VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

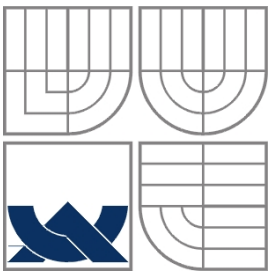
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RAYTRACING VIRTUÁLNÍCH GRAFICKÝCH SCÉN

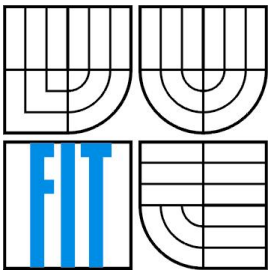
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. ANDREJ RYPÁK
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

RAYTRACING VIRTUÁLNÍCH GRAFICKÝCH SCÉN

RAYTRACING OF VIRTUAL GRAPHICS SCENES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. ANDREJ RYPÁK

VEDÚCI PRÁCE
SUPERVISOR

ING. JAN PEČIVA, PH.D.

BRNO 2012

Zadání diplomové práce/3423/2011/xrypak00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2011/2012

Zadání diplomové práce

Řešitel: **Rypák Andrej, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Raytracing virtuálních grafických scén**
Raytracing of Virtual Graphics Scenes
Kategorie: Počítačová grafika

Pokyny:

1. Nastudujte si grafickou knihovnu OpenSceneGraph a algoritmy používané při zobrazování metodami sledování paprsku.
2. Navrhněte aplikaci pro vizualizaci grafické scény, která bude používat některou z metod sledování paprsku.
3. Aplikaci implementujte a doplňte rozšíření dle domluvy s vedoucím. Zvažte integraci vašeho řešení s vizualizačním projektem Lexolights.
4. Vyhodnoťte vizuální výsledky a výpočetní náročnost vašeho řešení. Výsledky porovnejte s jinými řešeními a přístupy (POV-Ray, Lexolights, apod.).
5. Vytvořte demonstrační plakátek či video. Práci umístěte na internet.

Literatura:

- materiály k OpenSceneGraph a raytracingu dostupné na internetu

Při obhajobě semestrální části diplomového projektu je požadováno:

- Funkční prototyp aplikace.

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

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

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

Vedoucí: **Pečiva Jan, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 19. září 2011
Datum odevzdání: 23. května 2012

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

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce se věnuje problematice zobrazování a metodami sledování paprsků, zejména nejstarší metodě raytracing. Kromě přiblížení historie algoritmů z této rodiny, přináší detailní popis fyzikálního modelu, nástrojů a technik nezbytných pro vytvoření rendrovací aplikace. Značnou část práce pak tvoří samotná implementace aplikace pro fotorealistické zobrazování virtuálních 3D scén určených pro interaktivní grafiku, tedy zobrazování scén bez použití nestandardních informací z modelu. Obsahuje popis problémů a vysvětlení použitých řešení a postupů.

Abstract

This thesis is dedicated to ray tracing based rendering methods, primarily the original ray tracing. Besides introducing a brief historical overview of algorithms from the family, it presents all the essential tools, techniques and physics needed for designing a rendering application in detail. A significant part of the document consists of an implementation of a photorealistic rendering application for interactive graphics 3D virtual scenes. The focus is on rendering without using any additional model information. The thesis includes descriptions and explanations of specific problems and their solutions.

Klíčová slova

ray tracing, raytracing, OSG, OpenGL, grafické algoritmy, globální osvětlení, globální zobrazovací metody, optické jevy

Keywords

ray tracing, ray-tracing, OSG, OpenGL, graphics algorithms, global illumination, global rendering methods, optical phenomena

Citace

Andrej Rypák: *Raytracing virtuálních grafických scén*, diplomová práce, Brno, FIT VUT v Brně, 2012

Raytracing virtuálnych grafických scén

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Jana Pečivu.
Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Andrej Rypák
20.5.2012

Poďakovanie

Rád by som poďakoval predovšetkým svojmu vedúcemu, Ing. Janovi Pečivovi, Ph.D., za hodnotné a veľmi nápomocné konzultácie, za trpezlivosť a celkovo motivujúci prístup. Moja vďaka patrí aj Doc. Dr. Ing. Pavlovi Zemčíkovi za odbornú konzultáciu a Ing. Tomášovi Starkovi za názorné ukážky a cenné rady.

Obsah

1 Úvod

Táto práca sa venuje globálnej zobrazovacej metóde ray tracing a aplikácii, ktorá ju využíva na realistickú vizualizáciu virtuálnej 3D scény.

Fotorealistické techniky dnes hrajú vedúcu úlohu v CAD a CAM vizualizačných nástrojoch, medicínskych vizualizačných nástrojoch a špecializovaných simulátoroch, architektúre a modernom umení, slúžia pri vytváraní vizuálnych efektov vo filmoch či hrách. Dnešné animované filmy sú postavené takmer výlučne na veľmi pokročilých grafických algoritmoch. Fotorealistické algoritmy sa stále výraznejšie presadzujú aj v digitálnych hrách a v iných real-time aplikáciách v podobe shaderov a to najmä vďaka výraznému nárastu výkonu počítačov, nehovoriac o množstve scén, modelov a efektov pomocou takýchto algoritmov predpripravených.

Počítačová grafika je hlavne vďaka filmovému a hernému priemyslu jedným z najrýchlejšie sa vyvíjajúcich oborov spomedzi informačných technológií. Pokrok v hardvéri ide ruka v ruke s pokrokom v sofrvéri, preto niet divu, že stále vznikajú nové a dokonalejšie techniky ako v oblasti interaktívnych aplikácií, tak aj v oblasti aplikácií zameraných čisto na vizuálny výstup, ktoré sú schopné generovať obraz na nerozoznanie od fotografie.

V tejto práci prinášam pohľad na najstaršiu metódu, ktorá sa dá označiť za fotorealistickú, ray tracing. Hoci ide o najstaršiu metódu, nie je jej vývoj ani zďaleka mŕtvy, naopak, vďaka spomínanému pokroku v posledných rokoch vznikajú stále nové vylepšenia, ktoré prinášajú zrýchlenie výpočtu alebo zlepšenie vizuálnej kvality tejto metódy. Od jej vzniku inšpirovala mnoho algoritmov založených na podobných princípoch.

Táto práca nadväzuje na semestrálny projekt, z ktorého boli prevzaté, ale upravené a značne rozšírené kapitoly 2, 3 a 6. Na strane aplikačnej bol len čiastočne prevzatý systém spávy okien a kostra pre vrhanie primárnych lúčov, ktorá však bola prepracované do paralelnej podoby a implementovaná pomocou samostatných objektov.

1.1 Obsah práce

Druhá kapitola je uvádza princípy, históriu, nároky, výhody a nevýhody zobrazovacej metódy ray tracing. Tretia kapitola prináša detailný popis fyzikálneho modelu potrebného pri zobrazovaní sledovaním lúčov a štvrtá kapitola vysvetľuje osvetľovací model, tieňovanie, prácu s vektormi a ďalšie techniky, ktorých znalosť je pre renderovanie nutnosťou. Jedná sa o techniky, na ktorých závisí konečný vzhľad vyrenderovanej scény.

Piata kapitola obsahuje abstraktný návrh implementovanej interaktívnej aplikácie využívajúcej. Šiesta kapitola v krátkosti približuje grafickú knižnicu Open Scene Graph a grafické API OpenGL, na ktorých je aplikácia postavená. Napokon v siedmej kapitole sa nachádza rozbor použitých implementačných techník.

Ôsma kapitola porovnáva dosiahnuté výsledky s inými riešeniami, hodnotí výkon aplikácie z pohľadu rýchlosti aj kvality výstupu. Navrhuje možné vylepšenia a možný smer ďalšieho vývoja. V závere hodnotím priebeh práce.

2 Ray tracing

Táto kapitola približuje pojem generovania grafického výstupu počítačmi, renderovanie, zaoberá sa podstatou zobrazovacej metódy ray tracing a načrtáva históriu i súčasný stav zobrazovacích techník z tejto rodiny.

2.1 Renderovanie a zobrazovacie metódy

Grafické renderovanie je proces, pri ktorom sa využitím programu z virtuálneho digitálneho modelu alebo scény pozostávajúcej z viacerých modelov, prípadne iného popisu, napr. matematického, stáva digitálny obraz. Renderovanie scény môžeme tiež nazvať zobrazovaním scény, jedná sa o zobrazovanie 3D modelu pomocou 2D média.

Pri renderovaní sa väčšinou kladie dôraz na hodnovernosť a snahu kvalitou sa čo najviac priblížiť fotografickej snímke alebo filmu natočenému kamerou. Existujú však aj renderovacie procesy, ktoré sa naopak nesnažia priblížiť realite, ale chcú dosiahnuť nejaký iný vizuálny efekt, napríklad renderovanie animovaných filmových scén a pod.

Renderovanie je komplexný proces, v ktorom vystupuje veľké množstvo rôznych algoritmov. Tieto riešia napríklad usporiadanie objektov v scéne, ich viditeľnosť, farbu, textúru a kvalitu povrchu, optické vlastnosti materiálov, tieňovanie, šírenie svetla, nepriame osvetlenie, tiene, projekciu, potláčanie aliasingu textúr a hrán objektov a mnohé iné. Metódy renderovania určujú, ktoré algoritmy a v akej podobe budú použité. Podľa toho, či metódy zobrazovania pracujú len s časťou modelu či scény, alebo s celou či celou viditeľnou časťou scény, hovoríme **o zobrazovacích metódach**:

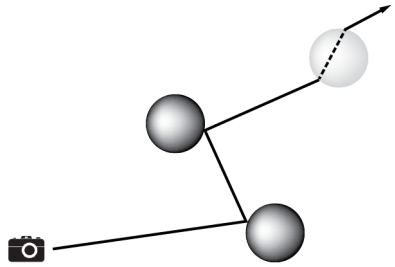
- **lokálnych**
  - algoritmy (hlavne riešenie viditeľnosti objektov a výpočet osvetlenia) prebiehajú iba na základe údajov práve spracovávaného objektu či primitíva a minima nevyhnutných globálnych dát (napríklad svetlá pri výpočte osvetlenia)
  - typickými predstaviteľmi sú fixné vykresľovacie reťazce a *scanline* algoritmy
- **globálnych**
  - algoritmy pracujú s celou dostupnou scénou, viditeľnosť objektov a osvetlenie je často výsledkom jedného procesu
  - typickými predstaviteľmi sú ray tracing a rádiozita spolu s príbuznými a hybridnými prístupmi

V mojej práci sa zaoberám technikami zobrazovania sledovaním lúčov, ktoré sa radia medzi globálne zobrazovacie metódy. [14]

2.2 Podstata techniky ray tracing

Ako už názov napovedá, jedná sa o sledovanie dráhy svetelných lúčov. Zjednodušene povedané, pomocou aproximovaného fyzikálneho modelu sledujeme dráhu lúča, ktorý dopadá do oka (kamery) od svetelného zdroja, pričom podľa pokročilosti fyzikálneho modelu berieme do úvahy jeho odrazy, ohyby, lom, útlm, rozklad a iné javy.

V praxi sa používa **reverzný prístup**, t.j. sleduje sa dráha lúčov od kamery k svetelnému zdroju, pretože zo svetelných zdrojov sa šíri nekonečné množstvo svetelných lúčov a len veľmi malá časť z nich sa dostane do kamery. Pri takomto reverznom prístupe používame **polpriamky smerujúce proti svetelným lúčom**, a hoci takéto polopriamky nepopisujú reálne svetelné lúče, pre jednoduchosť sú práve tieto v terminológii týchto algoritmov nazývame lúčmi.



Obrázok 2.1: spätné sledovanie lúčov

Lúče sú teda vysielané z priemetne ako predĺženie úsečiek medzi kamerou (okom) a priemetňou. Postupne putujú scénou a na základe odrazu, lomu a iných javov menia svoj smer. Technika ray tracing následne v bodoch kolízie lúčov s objektmi zbiera informácie o svetle dopadajúcom na tieto body a na ich základe počíta konečné osvetlenie bodu na priemetni, z ktorého bol lúč pôvodne vyslaný. [4], [15]

2.2.1 Typy lúčov

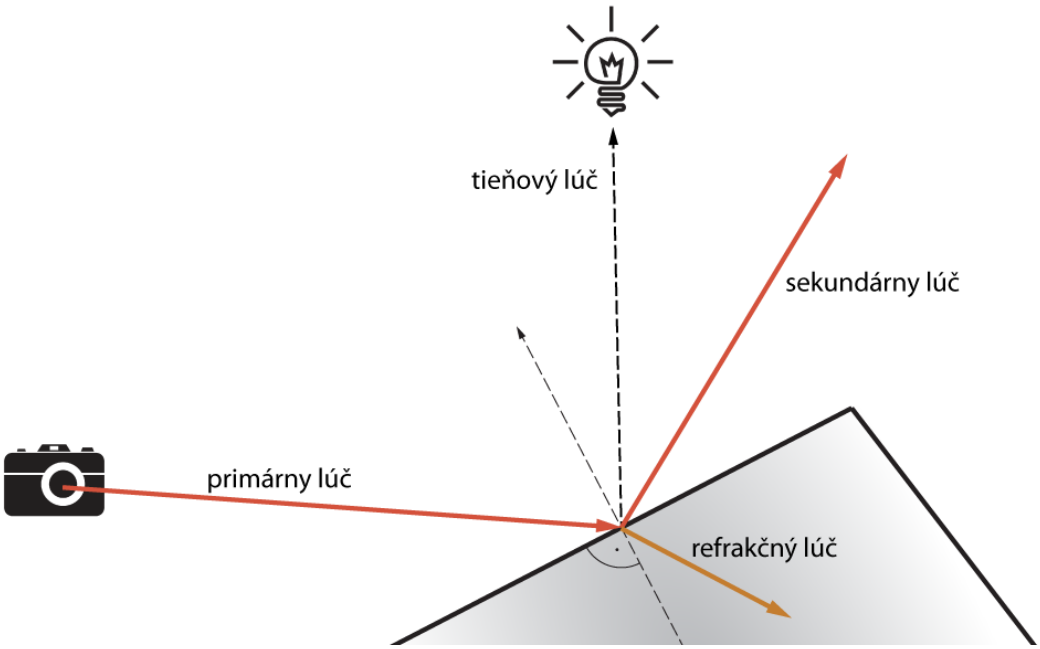
Ray tracing používa nasledujúce 4 typy lúčov:

- **primárne lúče**
  - lúče vyslané z priemetne do scény
  - v mieste dopadu na objekt môžu vzniknúť 3 nasledujúce typy lúčov, a to podľa vlastností daného objektu
- **sekundárne odrazené lúče**
  - sú odrazené lúče, ktoré pokračujú z miesta dopadu predošlého lúča smerom do scény
  - vznikajú, ak je povrch odrazivý, lesklý
  - ich smer sa vypočíta podľa zákona odrazu

- pre sekundárny lúč v mieste jeho dopadu na objekt v scéne platia tie isté pravidlá ako pre primárne lúče

- **refrakčné (lomené) lúče**
  - vznikajú, ak je objekt, na ktorý dopadol predošlý lúč, transparentný
  - smerujú dovnútra transparentného objektu a pohybujú sa v jeho vnútri
  - ich smer počítame pomocou Snellovho zákona
  - vnútri telesa sa pred jeho opustením môžu odrážať
  - po opustení telesa sa stávajú sekundárnymi lúčmi
- **tieňové lúče**
  - vo svojej podstate nie sú lúčmi, slúžia len na výpočet lokálneho osvetlenia
  - sú vysielané z miesta dopadu sekundárneho alebo primárneho lúča ku svetelným zdrojom

[13], [15], [4]



Obrázok 2.2: typy lúčov používané metódou ray tracing

2.2.2 Základný princíp výpočtu osvetlenia

Prvým krokom je vyslanie primárnych lúčov z prieметne do scény ako predĺženie úsečky medzi kamerou a konkrétnym bodom prieметne. Primárne lúče sú vrhané pre každý pixel prieметne, ktorou je výstupný buffer premietaný na displeji alebo ukladaný v podobe obrázka. Aplikácia musí vyhľadať priesečníky polpriamky lúča s geometriou scény. Vo väčšine prípadov stačí vyhľadať prvý priesečník. Tento bod nazývame bod kolízie.

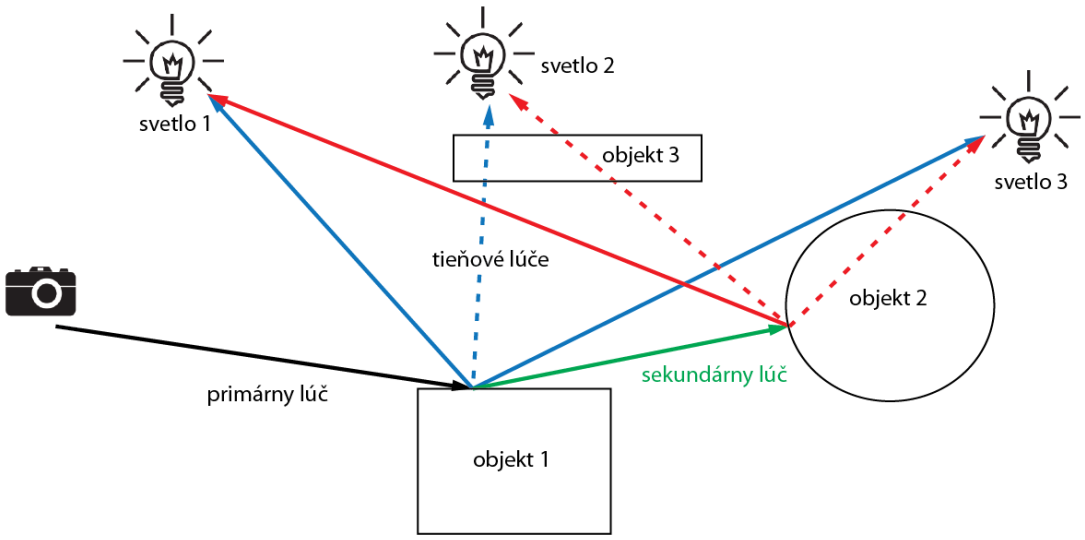
Následne sa v danom bode vypočíta lokálne osvetlenie vyslaním tieňových lúčov do svetelných zdrojov. Ak tieňový lúč po ceste k svetelnému zdroju pretne nejaké iné teleso v scéne, svetelný zdroj je zatienený a neberie sa do úvahy pre tento výpočet. Výpočet osvetlenia v bode kolízie prebieha podľa zvoleného lokálneho osvetľovacieho modelu, napr. Phongovho. V prípade transparentných objektov stojacich v ceste tieňovým lúčom je možné aplikovať rôzne techniky. Je napr. možné objekty ignorovať úplne alebo stlmiť vplyv príslušného svetla a pod.

Ďalším krokom je rekurzívny výpočet svetla prichádzajúceho od sekundárneho a refrakčného lúča, ktoré sú vyslané z kolízneho bodu, ak je objekt reflexný alebo transparentný. Vplyv týchto lúčov na konečnú intenzitu svetla v bode kolízie je daný vlastnosťami materiálu a uhlom dopadu lúča na rozhranie. Viac v podkapitole 7.3.4.

Sekundárne lúče sa po dopade na objekt v scéne správajú rovnako, ako primárne, t.j. prebehne výpočet osvetlenia pomocou tieňových lúčov a sú vyslané prípadné sekundárne a refrakčné lúče.

Refrakčné lúče prejdú vnútrom transparentných objektov a pretnú transparentné teleso. V tomto bode ho buď opúšťajú a stávajú sa sekundárnymi lúčmi, alebo sa odrážajú ďalej vo vnútri telesa.

Tento rekurzívny výpočet je nutné obmedziť maximálnou hĺbkou rekurzie, prípadne iným mechanizmom, inak by výpočet mohol byť nekonečný. [15], [4]



Obrázok 2.3: tieňové lúče, výpočet osvetlenia v scéne

Obrázok 4.9 zobrazuje princíp výpočtu osvetlenia v bode zásahu lúčom pomocou tieňových lúčov. Bod objektu 1 zasiahnutý primárnym lúčom je osvetlený svetlami 1 a 3, svetlo 2 je zatienené objektom 3, a preto toto svetlo nemá na výpočet osvetlenia v tomto bode vplyv. Obdobne, bod zasiahnutý sekundárnym lúčom je osvetlený len svetlom 1. Svetlo vypočítané v tomto bode je spätne propagované a spracované v bode zásahu objektu 1 primárnym lúčom.

2.3 Výhody a nevýhody metódy ray tracing

Lokálne zobrazovacie metódy potrebujú pre svoje výpočty informácie o aktuálne spracúvanom bode alebo primitíve a len minimálne množstvo globálnych dát (napríklad zdroje svetla). Takýto výpočet je veľmi rýchly, ale jednotlivé objekty scény sa navzájom nijako nemôžu ovplyvňovať, nevrhajú tiene a neodrážajú svetlo a ani sa nemôžu stať jeho sekundárnymi zdrojmi. Takto vyrenderovaná scéna nepôsobí realisticky.

Naproti tomu globálne zobrazovacie techniky berú do úvahy celé scény, alebo ich potrebné časti, a sú tak schopné realisticky zobraziť aj mäkké tiene, objekty implicitne navzájom vrhajú tieň a tiež sa môžu stať sekundárnymi zdrojmi svetla. Ray tracing vo svojej základnej podobe síce tiež nie je schopný zobraziť mäkké tiene, ale rôzne iné optické efekty, ako napríklad odrazy, odlesky, refrakciu a útlm lúčov, áno. Realistické mäkké tiene je však možné pridať pomerne jednoduchými úpravami [10].

Ray tracing má výhodu vo svojej škálovateľnosti. Je ho možné zapnúť a vypnúť na miestach, kde je alebo nie je potrebný. Je možné zvýšiť počet vzoriek na pixel tam, kde požadujeme hladké hrany, či zmeniť hĺbku rekurzie pre dosiahnutie vyššieho či nižšieho detailu a realistickosti.

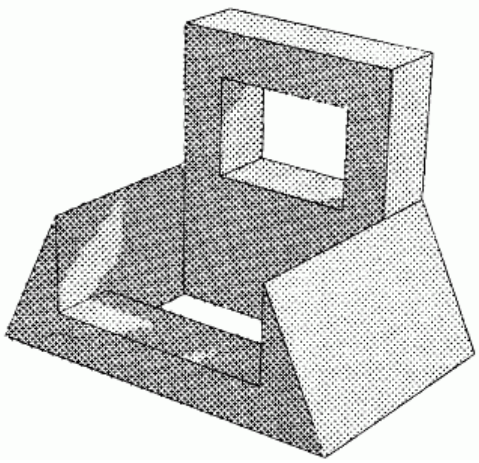
Rôznymi vylepšeniami a kombináciou ray tracing metódy s ďalšou metódou globálneho osvetenia, rádozitou, je možné sa hodnovernosťou veľmi priblížiť k reálnej fotografii. Tieto metódy patria k tomu najlepšiemu, čo súčastná grafika ponúka [13], za svoju realistickosť si však vyberajú daň v podobe vysokej výpočetnej náročnosti a sú teda nepoužiteľné v real-time aplikáciách. V týchto sa však využívajú na predprípravu scén, napríklad výpočet tieňových textúr, predpočítanie osvetlenia interiéru a podobne.

Na rozdiel od ray tracingu, ktorý skúma smer svetelných lúčov, rádiozita aproximuje distribúciu svetelnej energie v scéne. Z tohto základného rozdielu plynú výhody a nevýhody týchto algoritmov. V scéne zobrazovanej pomocou ray tracing algoritmu sa objekty nemôžu stať sekundárnymi zdrojmi svetla. Algoritmus taktiež nevie zobraziť mäkké tiene, pretože pracuje s bodovými svetlami[1]. Naopak, plošné svetlá a sekundárne zdroje svetla sú doménou rádiozity, ktorá však nie je schopná zobraziť odrazy, zrkadlenie, či ohyb svetla na rozhraní. V oboch prípadoch však už existujú riešenia, ktoré tieto nedostatky zakrývajú alebo odstraňujú úplne.

1 Svetelný zdroj v jedinom bode, geometricky. Takýto prípad je čisto teoretický.

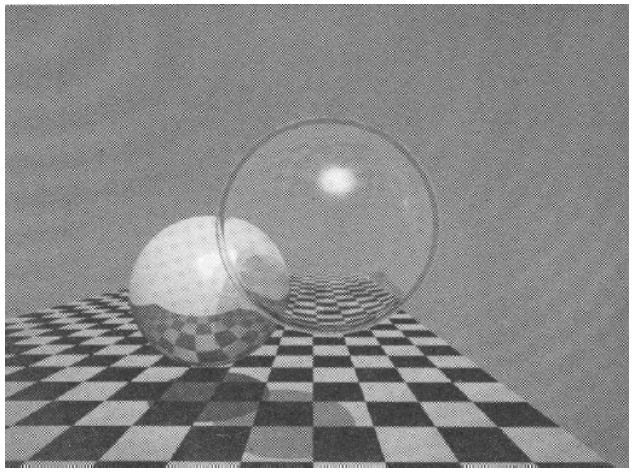
2.4 Vývoj techniky sledovania lúčov

Základy techniky ray tracing položil v roku 1968 Arthur Appel, [1], ktorý argumentoval potrebou zobrazovať vytieňované objekty a presné tiene pre lepšiu orientáciu v priestore. Svoju techniku demonštroval na jednoduchých objektoch v podobe strojových súčiastok. Nejednalo sa o pravý ray tracing, išlo o základ tejto metódy, dnes známy ako **ray casting** (vrhanie lúčov). Ray casting je v podstate zhodný s metódou ray tracing bez použitia sekundárnych a refrakčných lúčov.



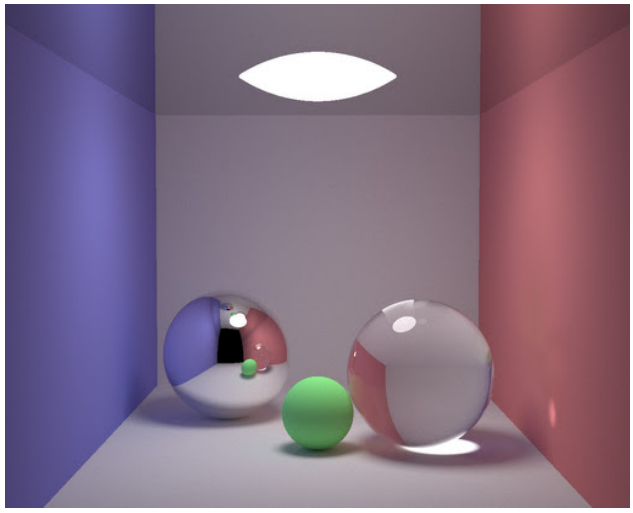
Obrázok 2.4: výsledok Appelovho algoritmu, tlač na plotteri

Na prelome osemdesiatych rokov došlo k revolúcii v podobe zrodenia plnohodnotnej techniky **ray tracing**, sledovanie lúčov, ktorá pre výpočet osvetlenia používala aj odrazené (sekundárne) lúče a tieňové lúče a bola schopná vytvoriť efekty lomu svetla, presné tiene a podobne. Otcom tejto techniky je Turner Witted, [4].



Obrázok 2.5: výsledok Wittedovho algoritmu

V roku 1986 James Kajiya prezentoval zobecnenú metódu ray tracing, s názvom **path tracing**, [5]. Metóda nahrádza lúče zobrazovacou rovnicou založenou na zachovaní energie svetla a na výpočet energie odrazeného lúča využíva BRDF (*Bidirectional Reclectance Distribution Function*), čo je funkcia distribúcie odrazovosti. BRDF udáva vlastnosti povrchu materiálu. Path tracing nepočíta s lúčmi ako s bodmi, ale ako s plochami. Taktiež svetlá sú plochami. Výpočet osvetlenia, zobrazovacia rovnica, je totiž založená na integráloch. Nakoľko počítanie niekoľkonásobných integrálov nie je výpočetne možné, v praxi sú integrály nahradené stochastickými metódami, primárne sa využíva metóda Monte Carlo. Takéto implementácie path tracingu sa nazývajú distribuovaný, monte carlo alebo stochastický ray tracing, prípadne

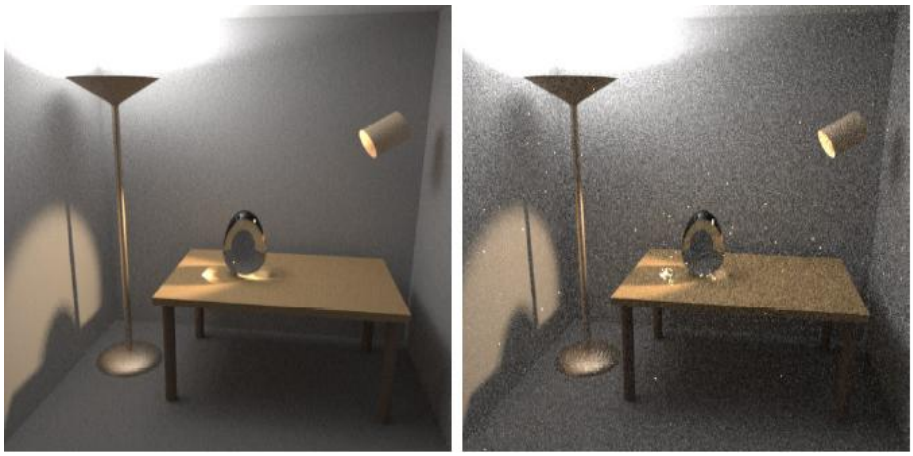


Obrázok 2.6: path tracing s kaustickými javmi a mäkkými tieňmi

path tracing. Path tracing vo všeobecnosti dosahuje lepšie vizuálne výsledky ako ray tracing a to hlavne vďaka schopnosti veľmi presne aproximovať aj javy ako sú mäkké tiene a kaustické javy, či efekty ako *motion blur* (rozmazanie pohybom) alebo *depth of field* (hĺbka poľa ostrosti). Nevýhodou tejto metódy je pomalá konvergencia výpočtu. Algoritmy path tracing totiž na rozdiel od ray tracing algoritmov, ktoré vysielajú (v princípe) jeden lúč na jeden pixel, vysielajú nepretržite množstvo vzoriek na každý pixel. Scéna je rozoznateľná až po niekoľkých vzorkách[2] na pixel. Vektory odrazu a lomu sú náhodne zvolené, pričom BRDF je distribučnou funkciou. Path tracing trpí pri nedostatočnom počte vzoriek veľmi výrazným šumom.

Rýchlejšiu konvergenciu, a teda lepšie vizuálne výsledky pri rovnakom počte vzoriek, dosahuje obojsmerné sledovanie cesty, tzv. **bi-directional path tracing**, uvedené v roku 1993, [6]. Táto technika vychádza z path tracing algoritmu, ale sleduje lúče ako od kamery smerom k zdrojom svetla, tak aj naopak. Cesty z oboch smerov sú spájané pomocou tieňových lúčov. Výstup metódy nie je zašumený, avšak nefunguje pre scény v exteriéri alebo pre scény so zložitou geometriou medzi svetlom a kamerou, alebo bodovými svetlami.

2 môže sa jednať o desiatky až stovky vzoriek, v závislosti od BRDF, zložitosti scény a iných faktorov



Obrázok 2.7: vľavo bi-directional path tracing (25 vzoriek na pixel), vpravo klasický path tracing (56 vzoriek na pixel) s rovnakým výpočetným časom

Nasledovali rôzne vylepšenia v podobe heuristických metód výpočtu vhodných ciest medzi zdrojom svetla a kamerou a monte carlo metód.

Od roku 2002, kedy Purcell uviedol svoj ray tracer implementovaný v programovateľnej GPU jednotke, [8], sa vývoj techník založených na sledovaní lúča zameriava na implementácie pre hardware grafických čipov, ktoré hlavne v posledných rokoch zaznamenali rapídny vzostup na výkone. Dnešné GPU vo výpočtoch s plávajúcou rádovou čiarkou ďaleko prekonávajú CPU, preto je tento trend viac než pochopiteľný.

Nebol to však prvý Hardvérovo implementovaný fotorealistický renderovací algoritmus. Reyes, vyvýjaný dnešným Pixarom, je platformou existujúcou už od skorých 90-tych rokov. Tento univerzálny systém je založený na shaderoch. Pixar ho dodnes používa pre renderovanie svojich filmov, známy RenderMan je založený na architektúre Reyes.

Medzi novšie hardvérové renderovacie systémy patrí nVidia Gelato, dnes dostupný na všetkých grafických čipoch od tejto firmy.

Dnes vývoj aplikácií na renderovanie fotorealistických snímok podporujú GPGPU nástroje a knižnice ako CUDA a OpenCL, či nástroj priamo určený na ich vytváranie, OptiX. Vznikajú optimalizované aplikácie využívajúce sledovanie lúčov, ktoré su schopné zobraziť scénu v reálnom čase (aj 20 a viac snímkov za sekundu), fyzikálne ešte presnejšie metódy založené na časticových princípoch šírenia svetla a podobne.

Taktiež vzniká veľké množstvo aplikácií, ktoré na základe programovateľných GPU shaderov vytvárajú efekty, vďaka ktorým sa ich výstup približuje k fotorealistickej kvalite, a to hlavne tiene, odrazy a odlesky, deformácie na nerovných povrchoch, či rozmazanie pohybom. Takéto aplikácie sú schopné bežať v reálnom čase.

2.5 Zhrnutie

Ray tracing vo svojej základnej podobe nie je optimálnou metódou fotorealistického zobrazovania, zato patrí k tým jednoduchším a rýchlejším, pričom jeho výstup je dostatočne kvalitný. Je schopný dobre aproximovať svetelné javy odrazu a lomu svetla. Jeho silnou stránkou sú lesklé povrchy.

3 Fyzikálna optika používaná pri sledovaní lúčov

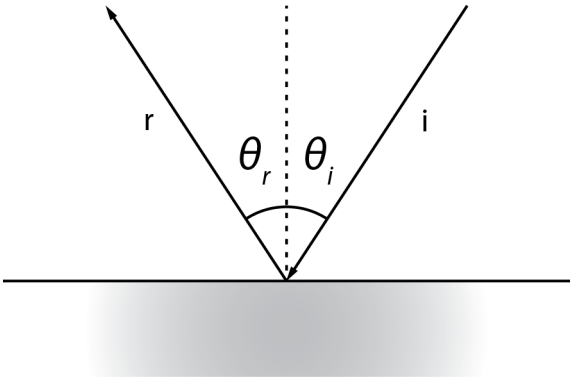
Pri technikách sledovania lúčov sa využíva prevažne geometrická optika, nazývaná aj optika lúčov, ktorá popisuje priamočiare šírenie svetla v podobe lúčov. Javy vyplývajúce z vlnovej či kvantovej podstaty svetla ako rozklad svetelného spektra, interferencia alebo ohyb svetla sú pre potreby zobrazovania zanedbateľné[3], až na experimenty zamerané na tieto javy. Geometrická optika je postavená na Fermatovom princípe šírenia svetla. Táto kapitola čerpá z [7] a [11].

3.1 Svetelné lúče

V izotropnom a homogénnom prostredí sa zväzky svetla šíria po priamkach. Takéto zväzky nazývame lúčmi. Tvrdenie vyplýva z Fermatovho princípu o najkratšom čase potrebnom na prekonanie vzdialenosti dvoch bodov lúčom. V počítačovej terminológii sa lúčmi nazývajú abstrakcie polopriamky, najčastejšie vyjadrené pomocou bodu a priestorového vektoru. V grafických aplikáciách spomenuté izotropné a homogénne prostredie medzi objektmi predpokladáme.

3.2 Odrazy a odlesky svetla na lesklých plochách

Pre odraz na zrkadliacich plochách platí empiricky už dávno dokázaný zákon odrazu – lúč dopadajúci na plochu sa odrazí pod rovnakým uhlom, pod akým dopadol na plochu, vzhľadom na normálu plochy.



Obrázok 3.1: odraz svetelného lúča

3 POV-Ray je schopný simulovať efekt rozkladu svetelného spektra

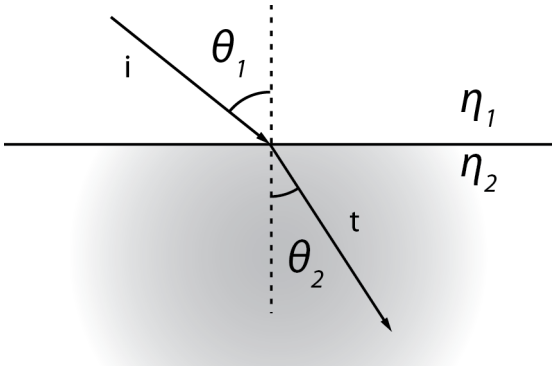
Platí vzťah

$$\theta_{\mathrm{i}} = \theta_{\mathrm{r}} \tag{3.1}$$

3.3 Lom svetla na rozhraní prostredí

Pre lom svetla na rozhraní dvoch prostredí s rôznou hustotou platí Snellov zákon, odvodený z Fermatovho princípu, ktorý hovorí, že pomer sínusov uhlu dopadu a refrakčného uhlu je rovný obrátenému pomeru indexov lomu prostredí.

$$\frac{\sin\left(\theta_1\right)}{\sin\left(\theta_2\right)} = \frac{\eta_2}{\eta_1} \tag{3.2}$$



Obrázok 3.2: Snellov zákon

Nezáleží na tom, ktoré prostredie je opticky hustejšie, η_1 je vždy prostredie, v ktorom sa pohybuje prichádzajúci lúč i a η_2 je prostredie, v ktorom sa pohybuje lomený lúč t .

3.4 Intenzita odrazeného a preneseného svetla

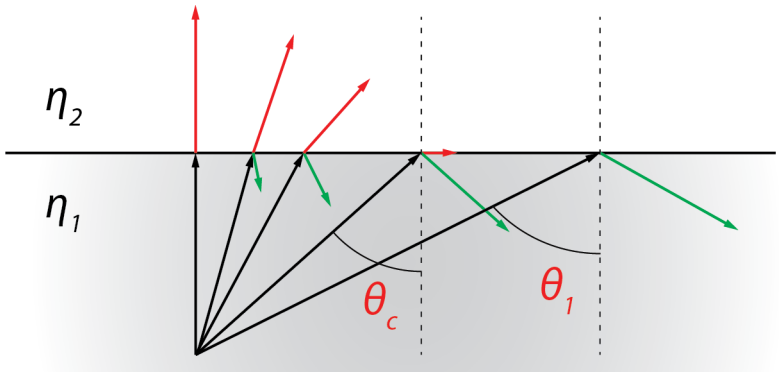
Pri dopade lúča na rozhranie s transparentným objektom s inou optickou hustotou v skutočnosti dochádza k obom javom popísaným v kapitolách 3.2 a 3.4 – dochádza aj k odrazu, aj k lomu svetla. Pre intenzitu svetla, ktoré sa odrazí (R) a svetla, ktoré vstupuje do transparentného objektu (T), platí zákon zachovania energia, a preto platí vťah:

$$\mathrm{T} + \mathrm{R} = 1 \tag{3.3}$$

Vzťah vyplýva z Fresnelových vzorcov pre elektromagnetické vlny prechádzajúce cez rozhranie [7].

Pri náraze lúča na rozhranie s telesom s nižšou hustotou môže nastať jav úplného vnútorného odrazu, t.j. lúč nevyletí z telesa von vôbec, ale úplne sa odrazí. Pri vzrastajúcom uhle dopadu totiž prudko klesá intenzita refrakčného lúča a stúpa intenzita odrazeného lúča. Úplný odraz nastáva, ak je uhol dopadu θ_1 väčší ako medzný uhol pre dané rozhranie θ_{c} .

Medzný uhol (anglicky *critical angle*) je uhol dopadu, pri ktorom lomený lúč a normála rozhrania sú navzájom kolmé.



Obrázok 3.3: medzný uhol a úplny vnútorný odraz

Medzný uhol sa dá vypočítať z indexov lomu materiálov podľa vzorca 3.4.

$$\theta_c = \arcsin\left(\frac{\eta_2}{\eta_1}\right) \tag{3.4}$$

Svetlo, ako elektromagnetická vlna, sa pri náraze na rozhranie rozkladá na dve zložky, *transverse electric (TE)* a *transverse magnetic (TM)*, navzájom kolmé. Tento jav sa nazýva polarizácia odrazom. Fresnelove vzorce počítajú s polarizáciou svetla. Pre intenzitu odrazeného svetla, platia tieto vzťahy:

$$\begin{aligned} R_\perp(\theta_i) &= \left(\frac{\eta_1 \cos(\theta_i) - \eta_2 \cos(\theta_t)}{\eta_1 \cos(\theta_i) + \eta_2 \cos(\theta_t)}\right)^2 \\ R_\parallel(\theta_i) &= \left(\frac{\eta_2 \cos(\theta_i) - \eta_1 \cos(\theta_t)}{\eta_2 \cos(\theta_i) + \eta_1 \cos(\theta_t)}\right)^2 \end{aligned} \tag{3.5}$$

Kde $R_\perp$ a $R_\parallel$ sú intenzity svetla s rovinou polarizácie kolmou na rovinu dopadu a rovnobežnou s rovinou dopadu.

Tieto vzorce sú však nepraktické pre ray tracing, kde sa polarizácia svetla kvôli zjednodušeniu výpočtov zanedbáva a všetko svetlo sa považuje za nepolarizované. V aplikáciách sa preto jednoducho vezme priemer oboch intenzít zo vzorca 3.5 a výsledná intenzita odrazeného svetla je daná vzťahom:

$$R = \frac{R_\perp + R_\parallel}{2} \tag{3.6}$$

Takto je pomocou vzorcov 3.3 a 3.6 možné vypočítať, v akom pomere je intenzita odrazeného svetla a svetla vstupujúceho do transparentného objektu.

4 Princípy a techniky používané pri zobrazovaní metódami ray tracing

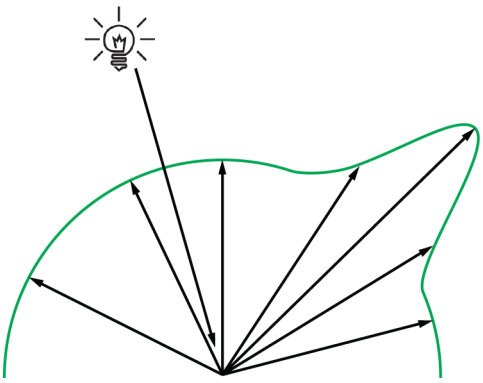
V tejto kapitole sa venujem technikám používaným pri zobrazovaní scény pomocou metód založených na sledovaní lúčov. Hoci popisované techniky sú pri ray tracing metódach esenciálne a stojí na nich kvalita výstupného obrazu, vo väčšej či menšej miere sa používajú aj pri ostatných zobrazovacích metódach.

4.1 Phongov osvetľovací model

Phongov model podobne ako ostatné lokálne osvetľovacie modely popisuje správanie sa svetla na povrchu materiálov pomocou troch svetelných zložiek, ktoré sú zhodné so zložkami svetla v OpenGL – *ambient*, *diffuse*, *specular*. [2], [3]

Jednotlivé typy svetla sa počítajú samostatne a ich súčtom vzniká celková výsledná intenzita svetla (farba) konkrétneho bodu v scéne:

$$I = I_a + I_d + I_s \tag{4.1}$$



Obrázok 4.1: svetlo na povrchu materiálu podľa Phongovho modelu ako súčet troch zložiek

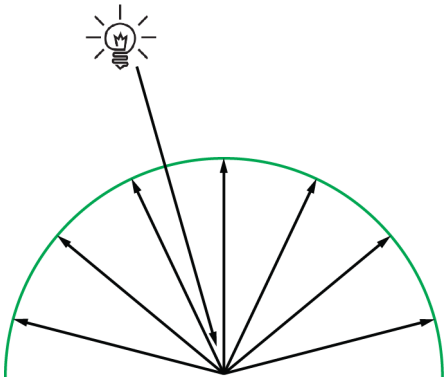
Intenzitu ambientného svetla prostredia vypočítame jednoducho vynásobením ambientných vlastností materiálu a svetelného zdroja. Ambientné svetlo vo phongovom modeli aproximuje nepriame osvetlenie.

$$I_a = k_a \; L_a \tag{4.2}$$

Obrázok 4.2: ambientné svetlo prostredia sa šíri do všetkých smerov rovnomerne a nezávisle od polohy zdroja svetla

Difúzne (rozptýlené) svetlo počítame ako funkciu uhlu dopadu svetelného lúča smerujúceho od svetelného zdroja. Odrazené svetlo sa šíri do všetkých smerov rovnomerne, teda aj smerom ku kamere. Difúzne svetlo je tlmené lineárnym, konštantným a kvadratickým faktorom (pozn., po anglicky *attenuation*). Difúzne svetlo simuluje správanie sa matných povrchov, ako napríklad papier, drevo, oblečenie, murivo, asfalt, hlina, steny...

$$I_d = \frac{k_d}{a + bd + cd^2} \left(d \cos\left(\theta_i\right) \right) L_d \tag{4.3}$$

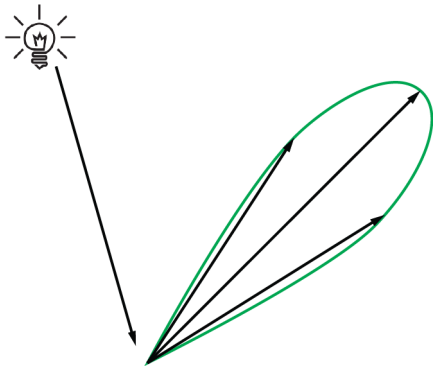


Obrázok 4.3: difúzne svetlo šíriace sa do všetkých smerov rovnomerne

Intenzitu odlesku (*specular*) spočítame ako funkciu uhlu medzi vektorom lúča odrazeného od povrchu a smerujúceho od svetelného zdroja a vektorom smerujúcim z miesta odrazu do kamery podľa vzorca 4.4. Vektor odrazu lúča sa počíta podľa zákona odrazu, vzorec 4.7. Zložka svetla *specular* vo Phongovom modeli aproxiumuje lesklé materiály, kovy, sklo, plasty, vosk, tekutiny...

$$I_s = k_s \, L_s \cos^{\alpha}\left(\phi\right) \tag{4.4}$$

Faktor α určuje ostrosť odleskov od povrchu, ostrejšie odlesky vznikajú pri vyšších hodnotách α,[2]. V OpenGL sa nazýva *shininess*.



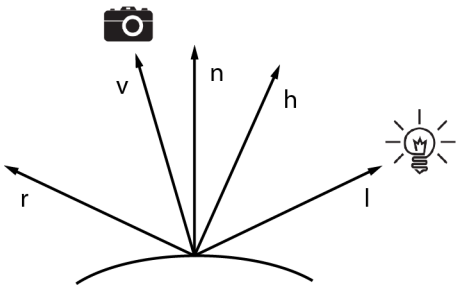
Obrázok 4.4: maximálna intenzita odlesku v smere odrazu od povrchu

Za predpokladu, že pozorovateľ a zdroj svetla sú v nekonečne, je možné použiť optimalizáciu, ktorej autorom je James F. Blinn a nazývame ju Blinn-Phongov osvetľovací model. Blinn zavádza pojem half-way vektor, teda vektor v polovici. Ide o vektor určený nasledujúcim vzorcom:

$$\vec{h} = \frac{\vec{l} + \vec{v}}{\left|\vec{l} + \vec{v}\right|} \tag{4.5}$$

Pri výpočte odleskov nahrádza half-way vektor ϕ , ktoré je konštantné pre celú scénu, kým sú svetelný zdroj a kamera v rovnakej pozícii. Pri použití tohto modelu je však nutné upraviť koeficient α .

$$I_s = k_s \, L_s \, \cos^{\tilde{\alpha}} \left(\rho\right) \tag{4.6}$$



Obrázok 4.5: vektory phongovho osvetľovacieho modelu

Tento model je možné s výhodou použiť pri scénach so svetlami umiestnenými v nekonečne, ale v iných prípadoch je výpočet pomocou pôvodného Phongovho modelu rýchlejší.

Doplnenie predchádzajúcich vzorcov:

- $\theta_i = \arccos\left(\vec{l}.\vec{n}\right)$, uhol medzi svetlom a normálou
- $\phi = \arccos\left(\vec{r}.\vec{v}\right)$, uhol medzi odrazeným lúčom a kamerou
- $\rho = \arccos\left(\vec{n}.\vec{h}\right)$, uhol medzi normálou a half-way vektorom
- k_a, k_d, k_s – parametre materiálu
- L_a, L_d, L_s – parametre svetla

4.2 Tieňovanie

Tieňovanie je všeobecný pojem, ktorý označuje aplikáciu svetelného modelu na objekt. Tieňovanie dotvára efekt priestorovosti a scéna získava na realistickosti.

Pri tieňovaní v 3D grafike zohráva veľkú úlohu smer normál povrchu. Ich smer a spôsob spracovania totiž zásadne menia výsledok aplikácie svetelného modelu.

Phongovo tieňovanie počíta normálu v mieste dopadu **interpoláciou normál** jednotlivých vrcholov príslušného geometrického primitíva. S touto interpolovanou normálou potom počíta osvetlenie pre každý jeho bod.

Týmto sa líši od **Gouraudovho tieňovania**, ktoré **interpoluje farbu** vypočítanú vo vrcholoch primitíva podľa príslušných normál, zjednodušuje tak výpočet, ale pre modely s nižším počtom polygónov je jeho vizuálny výsledok viditeľne horší.

Najjednoduchším typom tieňovania je **ploché tieňovanie** (anglicky *flat shading*), pri ktorom sa pre celé geometrické primitívum na základe jeho normály vypočíta jedna farba a **nič** sa **neinterpoluje**. Je výpočetne najmenej náročné, avšak produkuje vizuálne veľmi slabé výsledky s ostrými prechodmi medzi polygónmi.

Všetky tieto tieňovacie metódy v OpenGL využívajú Phongov alebo Blinn-Phongov osvetľovací model.

Obrázok 4.6: porovnanie výsledkov troch typov tieňovania,
zľava ploché tieňovanie (*flat shading*), Gouraudovo a Phongovo tieňovanie.
Prevzaté z [19]

4.3 Anti-aliasing

Anti aliasing je všeobecný pojem spojený s odstraňovaním javu zvaného aliasing. Dochádza k nemu pri vzorkovaní spojitej informácie, ak je frekvencia vzorkovania nižšia alebo rovná dvojnásobku najvyššej harmonickej frekvencie obsiahnutej vo vzorkovanom signáli (Shannonov vzorkovací teorém), teda napríklad na ostrých hranách objektov v scéne pri vzorkovaní rastrom (napr. bodmi priemetne). Jednoducho povedaná, pri hranách zvierajúcich s osami X alebo Y uhol vznikajú tzv. zubaté hrany (najviac viditeľné pri malých uhloch).

Anti aliasing (AA) v terminológii 3D aplikácií je pojem pre súhrn algoritmov, ktoré sa snažia tento jav zubatých hrán potlačiť.

Obrázok 4.7: scéna bez AA (vľavo) a s 8x FSAA, oba výstupy z Rayme

Základným typom AA je full scene AA (FSAA), ktorý funguje tak, že hodnota každého pixelu sa vypočíta z viacerých vzoriek z okolia myslenej pôvodnej vzorky, podľa ich počtu sa značí ako 4xAA, 8xAA, 16xAA. Ďalším, dnes už málo používaným, typom AA je multisampling (MSAA). Ten pracuje tak, že sa obraz vyrenderuje vo vyššom rozlíšení a potom sa zmenší na požadované rozlíšenie. Je rýchlejší, no dosahuje slabšie výsledky.

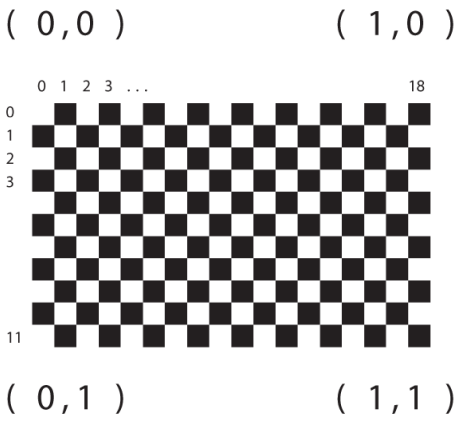
4.3.1 Adaptívny anti-aliasing

Implementáciou adaptívneho AA je možné značne znížiť počet potrebných vzoriek pri zachovaní zhruba rovnakej kvality výstupu.

Adaptívny anti-aliasing pracuje na základe predpokladu, že ak sa niekoľko vzoriek navzájom nelíši alebo sú rozdiely minimálne, nie je potrebné robiť ďalšie vzorky. Takto je napríklad pri adaptívnom 16x AA možné vyslať prvé 4 vzorky a ďalšie až v prípade, že sú medzi nimi značné rozdiely. Ak prvé vzorky nie sú dostatočne odlišné, farba pixelu sa vypočíta len z nich. V tomto prípade by sa ušetril výpočet ¾ vzoriek.

4.4 Textúry

Pri textúrovaní primitív je potrebné vypočítať, ktorá časť textúry sa použije. Tento proces sa nazýva mapovanie textúry. Namapovaný bod v textúre tvorí dvojica textúrovacích súradníc, v prípade 3D textúr trojica. Spojité textúrovacie súradnice je potrebné previesť na diskrétne pixely textúry (texely).



Obrázok 4.8: textúrovacie súradnice (v zátvorkách) a pixely textúry (texely)

Z obrázka 4.8 je vidno, že textúrovacie súradnice sú relatívne k rozmerom textúry. Nezáleží na pomere strán ani rozlíšení textúry, textúrovacie súradnice zostávajú rovnaké.

4.4.1 Filtrovanie textúr

Z faktu, že v 3D aplikácii môžu byť objekty v ľubovoľnej vzdialenosti a ich plochy pod ľubovoľným uhlom, vyplýva, že pri textúrovaní objektu jeden texel (pixel textúry) nemusí korešpondovať s jedným pixelom scény. Taktiež platí, že textúrovacie súradnice sú spojito mapované na diskrétnu textúru. Je teda istým spôsobom potrebné vybrať, ktoré body textúry sa použijú pri určovaní farby v mapovanom bode a s akou váhou.

Bežne sa používajú tieto štyri spôsoby, ktoré majú v dnešnom hardvéri už šandardne integrovanú podporu:

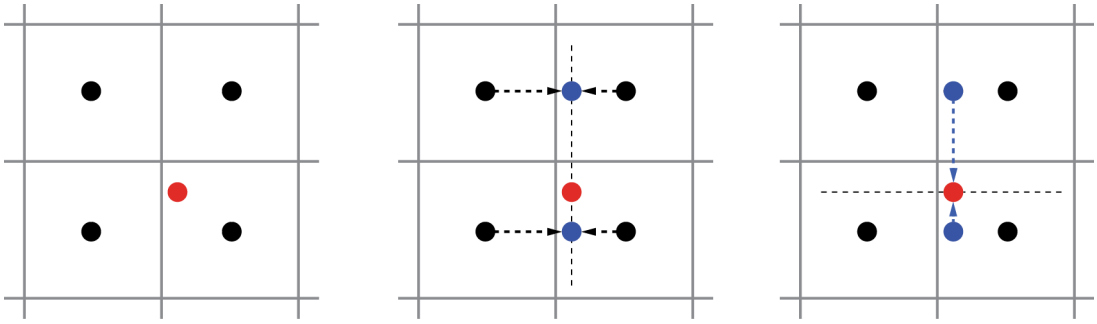
- metóda najbližšieho suseda
- bilineárne filtrovanie
- trilineárne filtrovanie
- anizotropné filtrovanie

4.4.1.1 Metóda najbližšieho suseda

Triviálny spôsob, akým možno postupovať, je výber texelu, ktorý je najbližšie k mapovanému bodu. Táto metóda je najrýchlejšia, ale pri zväčšovaní textúry dochádza k rozrastrovaniu, texely sa rozťahujú na niekoľko pixelov, vznikajú zubaté textúry. Pri zmenšovaní zase dochádza k aliasu textúry (moiré).

4.4.1.2 Bilineárne filtrovanie textúr

Ak sú dve dvojice bodov susediacich s mapovaným bodom lineárne interpolované podľa váhy určenej z polohy bodu a potom znova interpolované navzájom, hovoríme o bilineárnej interpolácii. Takto je možné zamedziť rozrastrovaniu textúry pri jej zväčšovaní. Bilineárne filtrovanie však tiež nie je schopné predísť aliasingu textúry pri jej zmenšovaní.



Obrázok 4.9: princíp bilineárneho filtrovania

4.4.1.3 MIP-mapping

Multum in parvo, mnoho v malom. Takto označujeme techniku, pri ktorej je spolu s textúrou v najvyššom rozlíšení distribuovaných aj niekoľko jej zmenšenín, vždy v pomere ¼ k predošlej (oba rozmery sú zmenšené na polovicu). Program potom podľa veľkosti, resp. vzdialenosti textúrovaného primitíva vyberá, ktorú úroveň MIP mapy použije. Tieto zmenšeniny sú predpočítané dopredu a tak nezaberajú čas procesora, ale pamäť.

Použitím MIP máp je možné zabrániť aliasu (*moiré*) pri zmenšovaní textúry, avšak za predpokaldu, že na predpočítanie je použitý zmenšovací algoritmus, ktorý zamedzuje jeho vzniku.

Nevýhodou MIP máp sú viditeľné prechody medzi úrovňami použitého detailu.

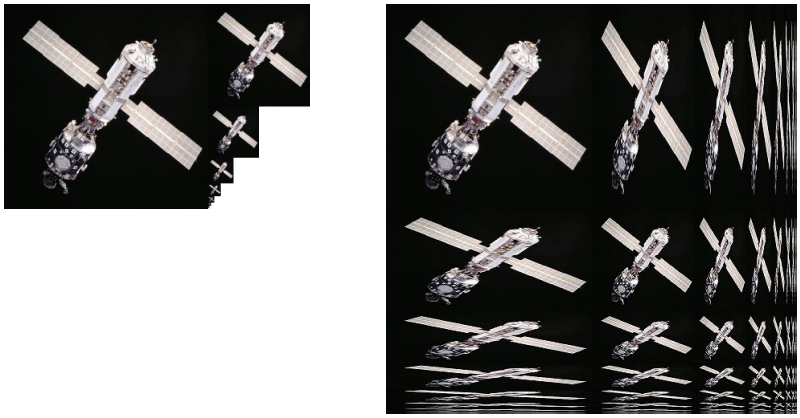
4.4.1.4 Trilineárne filtrovanie textúr

Trilineárne filtrovanie je založené na lineárnej interpolácii dvoch hodnôt získaných z dvoch susedných úrovní MIP mapy textúry. Tieto dve hodnoty boli získané bilineárnou interpoláciou nad vybraným detailom MIP mapy. Interpoluje sa medzi nimi na základe vhodnosti MIP mapy, t.j. na základe veľkosti, resp. vzdialenosti, textúrovanáho primitíva.

Trilineárne filtrovanie odstraňuje nedostatok MIP máp, avšak pri šikmých pohľadoch na objekty je rozostrenie, ktoré spôsobuje v oboch osiach textúry veľmi viditeľné.

4.4.1.5 Anizotropné filtrovanie textúr

Pri aplikovaní textúr na plochy pod veľkým uhlom je aliasing najviac viditeľný. Je možné ho odstrániť pomocou trilineárneho filtrovania, ale to so sebou prináša značné zmenšenie detailu textúry v oboch osiach. Pre zachovanie detailu v smere jednej osi je možné použiť anizotropné filtrovanie, ktoré je vylepšením trilineárneho filtrovania, založené na RIP mapách (neuniformných MIP mapách). Takto je možné zachovať detail v osi kolmej na smer pohľadu.



Obrázok 4.10: porovnanie MIP mapy (vľavo) a RIP mapy (vpravo)

Porovnanie vizuálneho výstupu týchto algoritmov zobrazuje obrázok A.1 v dodatku A.

4.5 Práca s vektormi

Lúče v terminológii ray tracing aplikácií sú abstrakcie polopriamky. Sú určené počiatočným bodom a vektorom alebo dvoma bodmi. Smer lúča udáva priestorový vektor. Pri výpočtoch odrazov, odleskov a lomu je potrebné vedieť vypočítať vektor odrazu a lomu. Tieto vektory sa použijú ako smerové vektory pre sekundárne a refrakčné lúče.

V tejto sekcii sú uvádzané výpočty pre vektory v 2D priestore, ale výsledné vzorce sú rovnako použiteľné pre 3D priestor.[18]

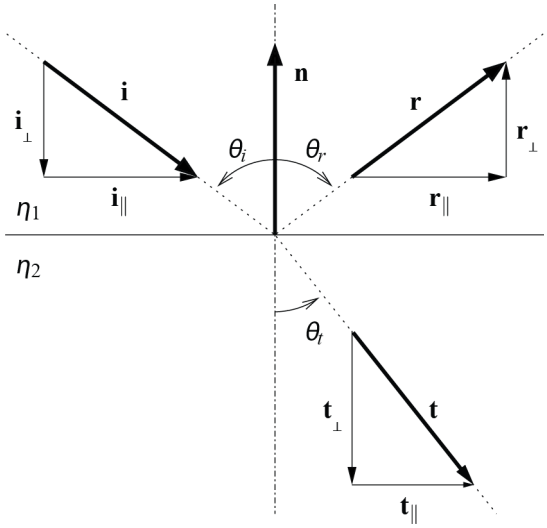
4.5.1 Základné vzorce pre rozhranie a odrazivé plochy

Základným predpokladom pri výpočte odrazených a lomených lúčov sú normalizované vstupné vektory $\vec{i}$ a $\vec{n}$:

$$\left|\vec{i}\right| = \left|\vec{n}\right| = \left|\vec{t}\right| = \left|\vec{r}\right| = 1 \tag{4.7}$$

Vektory $\vec{i}$, $\vec{r}$ a $\vec{t}$ je možné rozložiť na dve navzájom kolmé zložky – jednu kolmú a druhú paralelnú vzhľadom na rozhranie. Vzťahy pre $\vec{v}$ platia pre všetky 3 uvedené vektory.

$$\vec{v} = \vec{v}_{\perp} + \vec{v}_{\parallel} \tag{4.8}$$



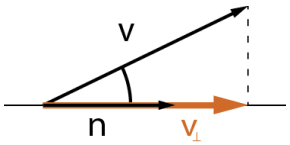
Obrázok 4.11: vektory rozhrania, upravené z [18]

Nie je ťažké dokázať, že ich skalárny súčin je rovný nule, čo potvrdzuje vzájomnú kolmosť.

$$\vec{v}_{\perp} \cdot \vec{v}_{\parallel} = 0 \qquad \Rightarrow \qquad \vec{v}_{\perp} \perp \vec{v}_{\parallel} \tag{4.9}$$

Ďalším poznatkom je, že všetky kolmé aj všetky rovnobežné zložky vektorov sú navzájom rovnobežné, navyše kolmé zložky sú rovnobežné s normálou rozhrania.

$$\begin{gathered} \vec{i}_{\perp} \parallel \vec{r}_{\perp} \parallel \vec{t}_{\perp} \parallel \vec{n} \\ \vec{i}_{\parallel} \parallel \vec{r}_{\parallel} \parallel \vec{t}_{\parallel} \end{gathered} \tag{4.10}$$



Obrázok 4.12: kolmý priemet vektora do priamky

Kolmý priemet získame pomocou vzorca 4.11, obrázok 4.12 znázorňuje použité vektory[4].

$$\vec{v}_{\perp} = \frac{\vec{n} \cdot \vec{v}}{\vec{n} \cdot \vec{n}} \vec{n} \tag{4.11}$$

Kolmá zložka je rovnobežná s normálou rozhrania (vzorec). Za predpokladu jednotkovej dĺžky normály (vzťah 4.7) je potom pre výpočet kolmej zložky možné vykonať nasledovné zjednodušenie:

$$\vec{v}_{\perp} = \frac{\vec{v} \cdot \vec{n}}{\left|\vec{n}\right|^2} \vec{n} = \left(\vec{v} \cdot \vec{n}\right) \vec{n} \tag{4.12}$$

Na základe vzorca 4.9 je možné použiť Pytagorovu vetu, a teda platí:

$$\left|\vec{v}\right|^2 = \left|\vec{v}_{\parallel}\right|^2 + \left|\vec{v}_{\perp}\right|^2 \tag{4.13}$$

4 Označenie vektorov vo vzorci 4.11a na obrázku 4.12 nemá priamy súvis s normálou na obrázku 4.11

Uhly dopadu, odrazu a lomu sú θ_i , θ_r , θ_t , v tomto poradí. Sú to menšie z dvojíc uhlov medzi príslušným lúčom a normálou plochy. Platí pre ne:

$$\begin{aligned} \cos\ (\theta_v) &= \frac{|\vec{v}_\perp|}{|\vec{v}|} = |\vec{v}_\perp| \\ \sin\ (\theta_v) &= \frac{|\vec{v}_\parallel|}{|\vec{v}|} = |\vec{v}_\parallel| \end{aligned} \tag{4.14}$$

Výsledok vzorca 4.14 vyplýva zo základnej trigonometrie, predpokladu 4.7 a vzorca 4.9.

4.5.2 Vektor pre odrazy a odlesky

Na základe zákona odrazu 3.1, dosadením 4.14 a upravením znamienka podľa obrázka 4.11 je zrejmé, že

$$\begin{aligned} \vec{r}_\perp &= -\vec{i}_\perp \\ \vec{r}_\parallel &= \vec{i}_\parallel \end{aligned} \tag{4.15}$$

Zo vzťahov 4.15 na základe 4.8 súčtom oboch vektorov získavame požadovaný smer vektora odrazu:

$$\vec{r} = \vec{r}_\parallel + \vec{r}_\perp = \vec{i}_\parallel - \vec{i}_\perp \tag{4.16}$$

Dosadením podľa 4.12 získavame po úprave výslednú rovnicu pre výpočet vektora odrazu.

$$\begin{aligned} \vec{r} &= \left[\vec{i} - \left(\vec{i}\ .\ \vec{n}\right)\vec{n}\right] - \left(\vec{i}\ .\ \vec{n}\right)\vec{n} \\ &= \vec{i} - 2\left(\vec{i}\ .\ \vec{n}\right)\vec{n} \end{aligned} \tag{4.17}$$

Využitím 4.8, 4.13, 4.15 a predpokladu jednotkovej dĺžky vektora $\vec{i}$ je možné dokázať, že aj vektor $\vec{r}$ má jednotkovú dĺžku.

4.5.3 Vektor pre lom

Vektor lomu je určený Snellovým zákonom 3.2. Zo vzorca vyplýva podmienka platnosti výrazu, ktorá je zároveň podmienkou pre úplný vnútorný odraz:

$$\sin\left(\theta_t\right) = \frac{\eta_1}{\eta_2}\ \sin\left(\theta_i\right) \quad \Leftrightarrow \quad \sin\left(\theta_i\right) \leq \frac{\eta_2}{\eta_1} \tag{4.18}$$

Opäť je vektor $\vec{t}$ možné rozdeliť na dve zložky, a potom je možné napísať:

$$\left|\vec{t}_\parallel\right| = \frac{\eta_1}{\eta_2}\ \left|\vec{i}_\parallel\right| \tag{4.19}$$

Nakoľko $\vec{t}_\parallel$ a $\vec{i}_\parallel$ sú rovnobežné a majú rovnaký smer, platí:

$$\vec{t}_\parallel = \frac{\eta_1}{\eta_2}\ \vec{i}_\parallel = \frac{\eta_1}{\eta_2}\left(\vec{i} + \cos\left(\theta_i\right)\vec{n}\right) \tag{4.20}$$

Využitím predpokladu o jednotkovej dĺžke vektorov a Pytagorovej vety, platí pre druhú zložku vektoru lomeného lúča nasledovné:

$$\vec{t}_{\perp} = -\sqrt{\left(1 - \left|\vec{t}_{\parallel}\right|^2\right)}\ \vec{n} \tag{4.21}$$

Súčet vektorov $\vec{t}_{\parallel}$ (4.20) a $\vec{t}_{\perp}$ (4.21) je rovný hľadanému vektoru lomu. Po úprave:

$$\vec{t} = \frac{\eta_1}{\eta_2}\ \vec{i} + \left(\frac{\eta_1}{\eta_2}\ \cos\left(\theta_i\right) - \sqrt{\left(1 - \left|\vec{t}_{\parallel}\right|^2\right)}\right)\ \vec{n} \tag{4.22}$$

Po nahradení $\left|\vec{t}_{\parallel}\right|$ podľa rovnice 4.14 získame:

$$\vec{t} = \frac{\eta_1}{\eta_2}\ \vec{i} + \left(\frac{\eta_1}{\eta_2}\ \cos\left(\theta_i\right) - \sqrt{1 - \sin^2\left(\theta_t\right)}\right)\ \vec{n} \tag{4.23}$$

Výraz pod odmocninou získame zo Snellovho zákona:

$$\sin^2\left(\theta_t\right) = \left(\frac{\eta_1}{\eta_2}\right)^2\ \sin^2\left(\theta_t\right) = \left(\frac{\eta_1}{\eta_2}\right)^2\ \left(1 - \cos^2\left(\theta_i\right)\right) \tag{4.24}$$

Z obrázka a definície skalárneho súčinu vyplýva:

$$\cos\left(\theta_i\right) = -\vec{i}\ .\ \vec{n} \tag{4.25}$$

Finálnu podobu vzorca pre výpočet vektora lomeného lúča získame nahradením zo vzťahov podľa 4.24 a 4.25 do 4.23:

$$\vec{t} = \frac{\eta_1}{\eta_2}\ \vec{i} + \left(-\frac{\eta_1}{\eta_2}\ \vec{i}\ .\ \vec{n} - \sqrt{1 - \left(\frac{\eta_1}{\eta_2}\right)^2\ \left(1 - \left(\vec{i}\ .\ \vec{n}\right)^2\right)}\right)\ \vec{n} \tag{4.26}$$

Vzorec vyzerá zložito, v skutočnosti je v praxi možné jeho opakujúce sa časti vyňať do premenných.

4.6 Hierarchické delenie priestoru

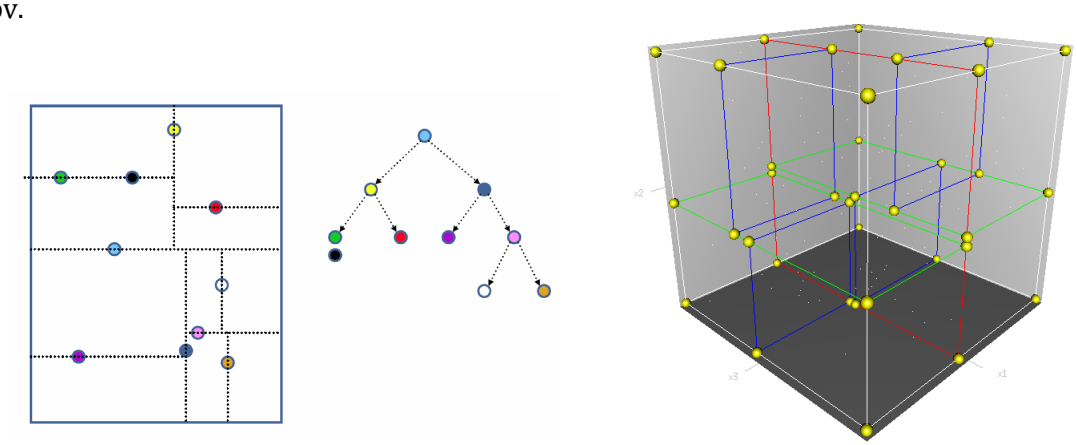
Hierarchické delenie priestoru scény je veľmi dôležitou súčasťou každej aplikácie, ktorá vo väčšej miere vyhľadáva priesečníky s geometriou scény. Usporiadaný hierarchicky rozdelený priestor sa totiž významne ľahšie prehľadáva, podobne ako je tomu pri vyhľadávaní utriedených a neutriedených polí.

k-D strom, po anglicky *k-D tree*, je priestorová dátová štruktúra určená na triedenie bodov v priestore. k-D strom je špeciálnym typom binárneho deliaceho stromu, ktorý delí priestor na polpriestory podľa hyperplochy kolmej na vybranú súradnicovú os. Takto predelí scénu vždy na dve časti. [9]

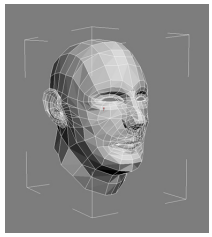
Táto štruktúra je veľmi potrebná pre určovanie priesečníku lúčov, pretože značne urýchľuje proces vyhľadania kolizného objeku, podobne ako binárne vyhľadávanie oproti sekvenčnému prístupu. Práve vyhľadávanie priesečníkov zaberá v algoritme väčšinu výpočetného času, ako už Witted poukázal v [4]. Witted nameral hodnoty pohybúce sa od 75% až po viac ako 95% času behu aplikácie.

Bounding box (BB), alebo obalové teleso, má za úlohu urýchliť hľadanie priesečníka so zložitou gemetriou, ktorú obaľuje, a tak jej tvar navonok zjednodušuje. S jednoduchými primitívami sa priesečníky hľadajú ľahšie ako so zložitými útvarmi. Najčastejšie sa využívajú BB v tvare kocky, ktorá má osi zarovnané so súradnicovým systémom, tzv. *Axis Aligned Bounding Box* (AABB). Ak nejaké teleso pretína BB, je potrebné urobiť test na priesečníky s geometriou, ktorú obaľuj, inak jednoducho môžeme prehlásiť, že priesečník neexistuje.

OSG používa obe techniky techniku obalových telies na úrovni uzlov a k-D stromy vrámci uzlov.



Obrázok 4.13: k-D strom, 2D nákres a 3D schéma



Obrázok 4.14: obalové teleso

5 Návrh aplikácie

Kapitola popisuje návrh aplikácie implementujúcej ray tracing algoritmus pomocou nástrojov popísaných v kapitole 6 na najvyššej úrovni abstrakcie.

Aplikácia je nazvaná **Rayme**.

5.1 Zameranie aplikácie

Aplikácia Rayme je zameraná na schopnosť vyrenderovať vizuálne krajšiu scénu z obyčajných modelov používaných pre bežné interaktívne 3D aplikácie, obsahujúcich poväčšine len informácie o povrchu určené pre Phongov osvetľovací model. Všetky výpočty teda musia čerpať informácie z dát poskytovaných zobrazovaciemu reťazcu OpenGL.

Rayme takto prináša do obyčajných scén opticky presné efekty zrkadlenia a lomu svetelných lúčov, ako aj Phongovo tieňovanie a tiene. Výpočetné jadro je schopné spracovať ľubovoľne veľké scény obsahujúce neobmedzený počet svetiel, je teda vhodné aj pre výpočet osvetlenia scén, ktoré bežné interaktívne 3D zobrazovacie aplikácie nezobrazia kvôli obmedzeniam počtu zobraziteľných svetiel v grafickom hardvéri.

Rayme je navrhnutý tak, aby bola jeho integrácia do iných vizualizačných projektov čo najjednoduchšia. Poskytuje rozhranie, pomocou ktorého je jeho integrácia otázkou niekoľkých riadkov kódu. Takto umožňuje zabudovanie aj do vizualizačného projektu Lexolights.

5.2 Hlavné časti aplikácie

Aplikácia Rayme je rozdelená na dve hlavné časti.

- výpočetné jadro
  - stará sa o výpočet komplexného osvetlenia scény
  - pomocou vláken rieši rozdelenie záťaže medzi výpočetné jednotky
- užívateľské rozhranie
  - slúži na nastavenie parametrov výpočetného jadra
  - poskytuje jadru potrebné údaje (graf scény, osvetlenie, vstupný a výstupný buffer)
  - umožňuje užívateľovi interakciu s aplikáciou
  - slúži ako vstupno-výstupné rozhranie pre zabudovanie do iných aplikácií

5.3 Užívateľské rozhranie

Užívateľské rozhranie Rayme je navrhnuté pre prácu v dvoch režimoch:

- interaktívny režim
  - umožňuje interaktívne upravovanie parametrov výpočetného jadra a polohy kamery
- konzolový režim
  - všetky nastavenia vrátane nastavenia kamery sú načítané z príkazového riadka
  - po skončení výpočtu sa aplikácia ukončí

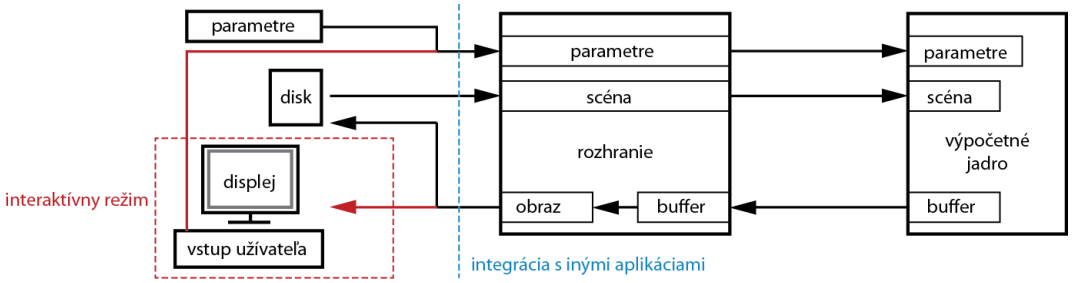
5.3.1 Interaktívny režim

Rayme je v interaktívnom režime po vizuálnej stránke rozdelený na dve okná, ktorých funkciou je zobrazovať model real-time spôsobom, pomocou OpenGL, a zobrazovať výsledok renderovania scény výpočetným jadrom.

Rayme sa v interaktívnom režime neukončí po vygenerovaní prvého obrázka, je možné vygenerovať ľubovoľný počet snímok.

Ďalšou funkciou interaktívneho režimu je uloženie stavu kamery pre potreby neskoršieho renderovania v konzolovom režime.

Pri renderovaní scény je podstatné nastavenie kamery, t.j. bodu, odkiaľ sú primárne lúče vrhané do scény. Tento bod aj so smerom pohľadu je možné nastaviť myšou v okne zobrazujúcom scénu real-time.



Obrázok 5.1: schéma toku dát v Rayme

6 Použité nástroje a technológie

Pri práci som sa zameral na prenosnosť a rozšíriteľnosť aplikácie, preto som zvolil open source knižnice a použil som programovací jazyk C++.

6.1 OpenGL

OpenGL (Open Graphics Library) je špecifikácia definujúca rozhranie pre programovanie grafických aplikácií, grafické API. Jeho úlohou je poskytnúť programátorovi jednotný prístup k rôznorodému hardvéru grafických akcelerátorov a skryť rozdiely medzi jednotlivými implementáciami. OpenGL vykresľovací reťazec na vstupe prijíma 3D popis scény a na výstupe vracia 2D raster (obraz). Pracuje ako stavový stroj a dnes[5] je už väčšinu jeho častí možné programovať alebo nahradiť mocou GLSL (OpenGL Shading Language).

V OpenGL pozostávajú všetky objekty z primitív – body, úsečky, trojuholníky a iné polygóny. Povrch môže byť popísaný pomocou textúr.

Jednou z výhod použitia OpenGL je jeho prenosnosť na množstvo platforiem, počnúc desktopovými (Windows, Unix, Mac ...), mobilnými, až po vstavané systémy.

Viac je o OpenGL možné nájsť napríklad v [16].

6.1.1 Svetlo v OpenGL

Nakoľko aplikácia pracuje s modelmi určenými pre interaktívnu real-time 3D grafiku, OpenGL, je podstatné popísať, akým spôsombom sú ich povrchové vlastnosti definované a ako je osvetlenie vyhodnocované v štandardnom vykresľovacom reťazci.

V OpenGL je reálne svetlo aproximované pomocou nasledujúcich 3 typov svetla:

- rozptýlené svetlo prostredia (**ambient light**), šíri sa do všetkých smerov rovnomerne a nedá sa určiť jeho smer
- difúzne svetlo (**diffuse light**), šíri sa od zdroja a po dopade sa odráža do všetkých smerov rovnomerne, slúži na matné vytieňovanie objektov
- odlesková zložka (**specular light**), šíri sa od zdroja a po dopade sa odráža v jednom smere, slúži na vytváranie odleskov

Jedná sa o zložky svetla používané Phongovým osvetľovacím modelom, bližsie popísané v kapitole 4.1.

Materiály majú 4 vlastnosti ovplyvňujúce osvetlenie: odrazivosť rozptýleného svetla prostredia (*ambient*), difúzneho svetla (*diffuse*), odleskového svetla (*specular*) a vyžiarené emisné svetlo (*emission*). Emisné svetlo v OpenGL nefunguje ako svetelný zdroj, ale priamo mení farbu objektu, ostatné tri reagujú so svetelnými zložkami svetelných zdrojov a svetla z prostredia.

5 OpenGL aktuálne vo verzii 4.2, GLSL uvedené v OpenGL verzie 2.0

Každý z týchto typov svetla je v OpenGL navyše vyjadrený pomocou 4 farebných zložiek (pre RGBA farebný model). Každú zložku farby je nutné vypočítať samostatne.

Farba konkrétneho bodu v scéne sa vypočíta z jednotlivých typov svetla, ktoré sú ovplyvnené ako materiálom, tak aj svetlom. Konkrétny postup výpočtu riešia osvetľovacie modely, v OpenGL je to Phongov osvetľovací model. [13], [16]

6.1.2 Tri typy svetiel v OpenGL

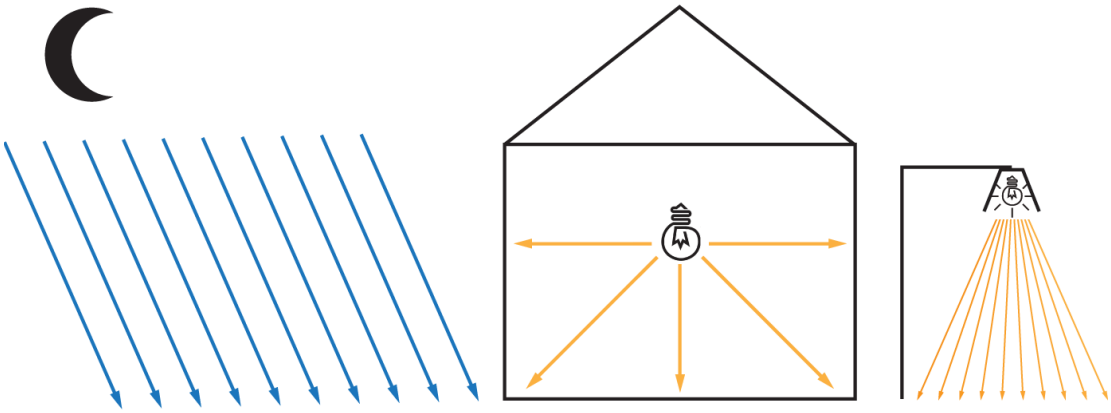
OpenGL rozoznáva 3 typy svetiel:

- smerové svetlo (*directional light*)
  - má smer, leží v nekonečne, určené vektorom
  - slnko, mesiac ...
- bodové svetlo (*point light*)
  - nemá smer, leží v určenom bode a svetlo vysiela do všetkých smerov
  - netienená žiarovka či žiarivka
- reflektorové svetlo (*spot light*)
  - má určený smer, leží v konkrétnom bode v scéne, svetlo vysiela len v kuželi určenom uhlom a vektorom udávajúcim smer svietenia
  - svetlomety automobilov, pouličné lampy, bodové interiérové osvetlenie

Výpočet osvetlenia je pre každé svetlo odlišný.

- smerové svetlo
  - počíta sa na základe smerového vektora, pozícia je v nekonečne
  - neaplikuje sa tlmenie na základe vzdialenosti
- bodové svetlo
  - počíta sa na základe polohy zdroja
  - aplikuje sa naň tlmenie so vzrastajúcou vzdialenosťou väčšie
- reflektorové svetlo
  - počíta sa na základe pozície, smeru svietenia a svetelného kužeľa, ktorý je určený uhlom medzi smerom svietenia a stranou kužeľa, takzvaný *cutoff* uhol
  - aplikujú sa dva typy tlmenia – narastajúce so vzdialenosťou a narastajúce s odchýlkou od smeru svietenia, ak je odchýlka vyššia ako uhol *cutoff*, príspevok svetla je nulový

Parametre tlmenia na základe vzdialenosti sú nastavované v modeli buď pre dané svetlo, alebo pre celú scénu. Parametrami sú konštaný, lineárny a kvadratický koeficient tlmenia.



Obrázok 6.1: tri typy svetiel v OpenGL, zľava smerové, bodové a reflektorové

6.2 Graf scény

Graf scény je všeobecná datová štruktúra používaná programami pracujúcimi s vektorovou grafikou. Štruktúra drží logické a veľmi často aj priestorové usporiadanie grafickej scény. Definícia grafu nie je jasná, pretože implementácie sa navzájom líšia prispôsobením na konkrétny problém. Grafom scény býva zväčša súbor uzlov v stromovom alebo grafovom usporiadaní, pričom uzol (node) môže mať viac potomkov, a pokiaľ sa nejedná o stromovú štruktúru, aj viac predkov. Vlastnosti predkov sú propagované na všetky deti, takže efekt operácie na skupine sa hneď prenesie na všetkých jej členov. [12]

6.3 Predstavenie nástroja Open Scene Graph

OpenSceneGraph alebo OSG je objektovo orientovaný open source nástroj (knižnica, framework) pre tvorbu 3D grafických aplikácií, ktorého základom je, ako už názov napovedá, štruktúra typu graf scény. Nástroj je dobre prenosný na mnoho platforiem, nakoľko je napísaný v jazyku C++ a postavený na OpenGL API, ktoré je samo o sebe veľmi dobre prenosné.

Graf scény použitý v OSG je **orientovaným grafom**, nie je teda stromom. Každý uzol môže mať viacero potomkov aj predkov. Každý uzol je teda jednoznačne logicky identifikovaný cestou od vrcholu grafu k danému uzlu a jeho priestorové usporiadanie a vlastnosti sú súhrnom vlastností a usporiadania uzlov v tejto ceste.

6.3.1 Hlavné prednosti OSG

OSG je framework s pomerne vysokou úrovňou abstrakcie, je schopný takmer úplne skryť nízkoúrovňové funkcie OpenGL a prináša všetky výhody objektovo orientovaného programovania. To sú pre programátora hlavné dôvody použiť OSG. Tento framework navyše programátorovi vôbec nebráni použiť nízku úroveň riadenia OpenGL tam, kde to vyžaduje, pretože ľubovoľnú časť API je možné doplniť alebo nahradiť vlastným kódom.

OSG ďalej poskytuje množstvo nástrojov, ktoré je možné využiť na rôzne rutiny, a tak urýchliť vývoj aplikácie. Pre moju prácu relevantné, spomeniem napríklad hľadanie priesečníkov lúča so scénou s podporou k-D stromov a obalových telies, manipulácia kamery, správa svetiel, získavanie textúr, normál, vlastností materiálu a ďalších potrebných informácií o céne, načítavanie a ukladanie scén.

6.3.2 Scény v OSG

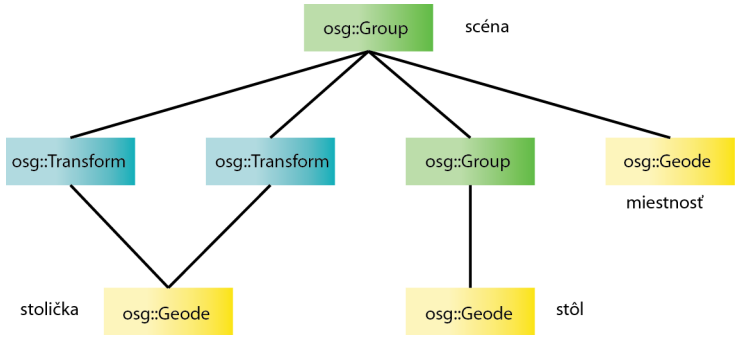
Graf scény v OSG sa skladá z uzlov. Veľké množstvo typov uzlov a ich schopnosti implicitne organizovať priestor poskytujú možnosti ukladania dát nedostupné v renderovacích API nízkej úrovne. OpenGL sa sústredí na sprístupnenie prvkov grafického hardvéru, a hoci poskytuje isté formy ukladania dát pre použitie neskôr (*dispaly lists*), podpora logickej a priestorovej organizácie je minimálna a nedostačujúca pre väčšinu 3D aplikácií [17].

Prvým uzlom scény je vrchol grafu. Pod týmto uzlom sa nachádzajú skupinové uzly (*group node*), ktoré organizujú geometriu a renderovací stav, ktorý určuje vzhľad geometrie. Na opačnej strane grafu, úplne na konci, sú listy (*leaf node*), ktoré obsahujú vlastnú geometriu vytvárajúcu objekty v scéne.

6.3.3 Základné typy uzlov a objektov v OSG

OSG obsahuje množstvo uzlov, ktoré je možné využiť pri stavbe grafu scény. Uvádzam dva základné, potrebné pre pochopenie kapitoly 7:

- `osg::Group`
  - základný skupinový uzol, slúži na organizáciu priestoru
  - dedia od neho transformačné uzly, ktoré menia polohu, veľkosť, natočenie potomkov
  - dedia od neho aj uzly, ktoré prepínajú časti grafu, skrývajú alebo zobrazujú, menia úroveň detailu, alebo nesú informácie o zdroji svetla
  - nemôže byť listom grafu, nenesie priamo informáciu o geometrii
- `osg::Geode`
  - listy grafu sú instanciami tohto uzlu
  - nosič geometrie



Obrázok 6.2: príklad jednoduchého grafu scény miestnosti so stolom a dvoma stoličkami

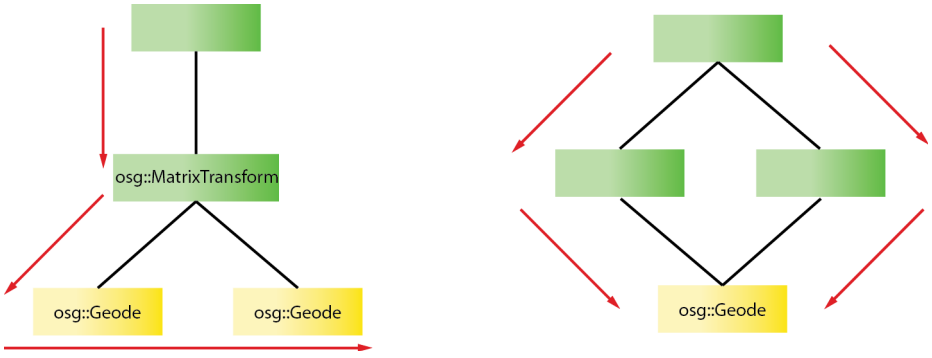
Niektoré dôležité objekty:

- `osg::Drawable`
  - objekt, pomocou ktorého sa objekty scény vykresľujú
  - obsahuje príkazy pre OpenGL
  - každý uzol `Geode` obsahuje jeden alebo viac `Drawable` objektov
- `osg::StateSet`
  - obsahuje údaje korešpondujúce s parametrami stavu stavového stroja OpenGL
  - každý uzol môže mať `StateSet`, a tiež `Drawable` majú vlastný `StateSet`

6.3.4 Renderovanie pomocou OSG

Graf scény renderuje v prechodoch. Základným prechodom je vykresľovací prechod, ale typicky grafy ponúkajú ďalšie prechody – prechod, v ktorom sa aktualizuje geometria a stavy, prechod, pri ktorom sa triedia uzly a nepotrebné sa z vykresľovania vylučujú a vykresľovací prechod.

Prechodom sa rozumie sled príkazov, ktorý má za následok iteráciu cez všetky (potrebné) uzly grafu po všetkých možných prípustných cestách od vrcholu po listy.



Obrázok 6.3: dva príklady renderovania stromu

Počas takéhoto prechodu aplikácia zbiera informácie a propaguje ich smerom k potomkom. Takto sa zbierajú napríklad transformácie a zmena transformačnej matice v uzle `osg::MatrixTransform` po ceste k listom v grafe na obrázku 6.3 má za následok pohyb oboch jej potomkov.

7 Detaily implementácie

Kapitola popisuje a vysvetľuje konkrétne postupy použité pri implementácii aplikácie Rayme.

Na začiatok uvádzam, čo je implementované v Rayme a čo Rayme poskytuje:

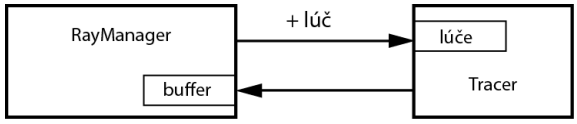
- načítanie ľubovoľného modelu v podporovanom formáte[6]
- výstup výsledného vygenerovaného obrázku a nastavenia kamery
- interaktívne a aj konzolové ovládanie a beh aplikácie
- fyzikálne presný komplexný svetelný model založený na Fresnelových rovniciach, odrazy svetla, lom svetla
- Phongov osvetľovací model, Phongovo tieňovanie s interpoláciou normál
- textúrovanie s podporou viacerých textúrovacích režimov a bilineárnym filtrovaním
- 8x alebo 4x FSAA anti-aliasing, alebo bez AA
- výpočetné jadro pracujúce paralelne vo viacerých vláknach

Nasledujúce podkapitoly popisujú a vysvetľujú detaily použitia týchto techník.

7.1 Model výpočetného jadra

Výpočetné jadro sa skladá z objektu vytvárajúceho a spravujúceho lúče, `RayManager` a pomocného objektu, ktorý spúšťa proces vyhodnocovania svetla pre každý lúč, `Tracer`. Takýto návrh som zvoli preto, lebo objekty `Tracer` môžu byť instancované a môže ich tak existovať niekoľko naraz, každý vyhodnocujúc vlastnú sadu lúčov.

Okrem vytvárania lúčov má `RayManager` na starosti sledovanie času výpočtu a predspracovanie potrebných údajov o zdrojoch svetla.



Obrázok 7.1: schéma výpočetného jadra

7.2 Lúče a priesečníky

Lúče v aplikácii Rayme sú dedičmi OSG triedy `LineSegmentIntersector`, ktorá je abstrakciou úsečky. V scéne zbiera informácie o objektoch, ktoré pretína, a ukladá si okrem iného polohu týchto priesečníkov. Na výpočet týchto priesečníkov využíva zabudovanú podporu k-D stromov v OSG.

6 čokoľvek, čo je schopné OSG načítať do grafu scény

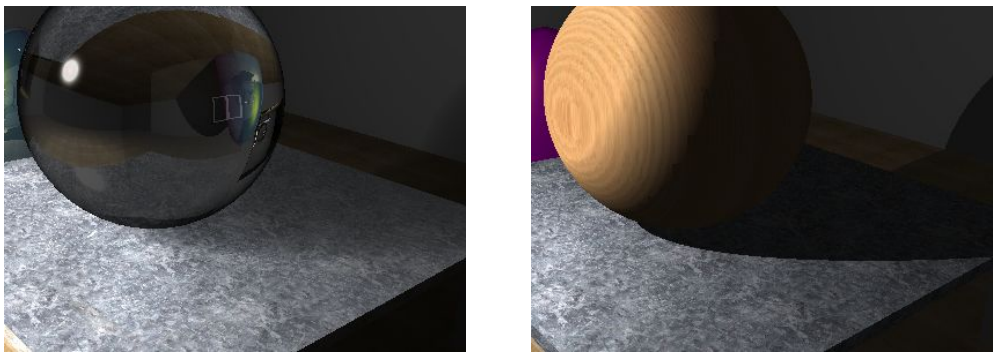
k-D stromy sú v OSG podporované, ale aplikácia ich zostaví pre každý `Drawable` v scéne zvlášť. Toto prináša nutnosť iterovať cez jednotlivé `Drawable` v grafe a takto spomaľuje výpočet priesečníkov s grafom scény. Nakoľko výpočty priesečníkov zaberajú najväščie množstvo výpočetného času, je toto spomalenie pri scénach s väčším množstvom `Drawables` veľmi citeľné a výpočet je hrdlom v celkovom výpočte osvetlenia scény.

7.3 Svetelný model

Ak lúč pretne geometriu scény, priesečník sa označí za kolízny bod a vypočíta sa normála povrchu. V kolíznom bode sa následne vypočíta intenzita osvetlenia podľa Phongovho svetelného modelu (kapitola 4.1) pre každé relevantné svetlo.

Na zistenie príspevku toho-ktorého svetla, sú z kolízneho bodu vyslané tzv. tieňové lúče (*shadow rays*) smerom ku svetelným zdrojom, ktoré majú za úlohu zistiť, či medzi kolíznym bodom a daným svetlom nie je nejaký nepriehľadný objekt. Tieňové lúče nie sú abstrakciami reálnych lúčov.

Rayme vyhodnocuje priesečník tieňového lúča a transparentného telesa tak, že zníži príspevok svetelného zdroja, a to podľa hrúbky telesa. Takto je zabezpečené, že tenké priehľadné objekty (napr. okná) nezatienia kolízny bod.



Obrázok 7.2: porovnanie mäkkého tieňa transparentnej gule a ostrého tieňa nepriehľadnej

Posledným krokom je výpočet odrazivosti povrchu kolízneho bodu a vyslanie odrazeného sekundárneho lúča. V prípade transparentného telesa, aj vyslanie refrakčného lúča.

7.3.1 Svetelné zdroje a optimalizácia

V kapitole 6.1.2 sú popísané tri typy svetiel používané v OpenGL a teda aj v Rayme. Z kapitoly a obrázka 6.1 plynie, že niektoré objekty neosvetľujú určitý bod v scéne, bez ohľadu na zatienenie inými objektmi v scéne. Ide o prípad, kedy je spomínaný bod mimo kužeľa reflektorového svetla, alebo je príliš ďaleko od bodového alebo reflektorového zdroju svetla.

Vzhľadom na to, že počítanie priesečníkov tieňových lúčov je časovo náročná operácia, kým výpočet vzdialenosti a vzájomnej polohy bodu a svetelného zdroja je pomerne jednoduché, kôli optimalizácii je najskôr vyhodnotená poloha a typ svetelného zdroja – ak je svetlo ďaleko alebo svieti iným smerom, je vyradené z výpočtu – a až keď vyhovuje tejto podmienke, zisťujú sa priesečníky so scénou, nie v opačnom poradí. Táto optimalizácia prináša zrýchlenie hlavne

pri rozsiahlejších scénach, kde sú svetlá ďaleko od seba alebo je využitých mnoho reflektorových svetiel a podobne. Na druhej strane sa jedná o spomalenie, avšak zanedbateľné, pre malé scény a scény s malým počtom svetiel.

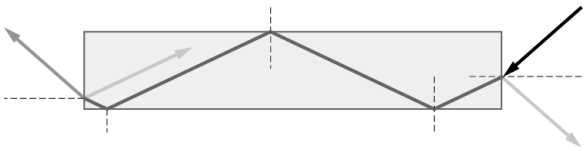
7.3.2 Riešenie sekundárnych lúčov, rekurzia

Sekundárne lúče sú lúče vyslané z bodu kolízie predchádzajúceho (otcovského) lúča pod uhlom vypočítaným podľa vzorca 4.17 pre odrazy vektorov. Intenzita svetla vypočítaná v bode kolízie sekundárneho lúča sa pripočíta k intenzite svetla otcovského lúča, pričom sa aplikuje tlmenie na základe vzdialenosti, ktorú sekundárny lúč prekonal.

Rekurzia končí, ak aplikácia dosiahne požadovanú maximálnu hĺbku rekurzie, alebo ak je príspevok sekundárneho lúča príliš malý, aby bol postrehnuteľný[7].

7.3.3 Riešenie priehľadných objektov

V prípade zásahu transparentného objektu sa do jeho vnútra vyšle refrakčný lúč. Refrakčný lúč prejde vnútrom objektu a zvnútra narazí na jeho povrch. V tomto bode sa vyšle do scény sekundárny lúč (za predpokladu, že nedôjde k úplnému vnútornému odrazu) a taktiež ďalší refrakčný lúč. Vektory oboch lúčov sú vypočítané podľa rovníc 4.17 a 4.26. Pre oba vyslané lúče platí taktiež podmienka o ukončení rekurzie z predchádzajúcej podkapitoly (7.3.2).



Obrázok 7.3: rozdeľovanie príspevku lúčov medzi refrakčné a odrazené lúče. Pri úplnom vnútornom odraze žiaden lúč neopúšťa teleso

7 Prah postrehnuteľnosti príspevku sekundárneho lúča je možné aplikačne nastaviť.

7.3.4 Miera odrazivosti a priepustnosti materiálu

V kapitole 3.4 je popísaný fyzikálny model pre výpočet intenzity svetla odrazeného a svetla pokračujúceho do transparentného objektu. Rayme kvôli rýchlejšiemu výpočtu používa Schlickovu aproximáciu Fresnelovej rovnice. Jeho rovnica pre odrazivosť je:

$$\mathrm{R}\left(\theta_{\mathrm{i}}\right) = \mathrm{R}_0 + \left(1 - \mathrm{R}_0\right)\left(1 - \cos\left(\theta_{\mathrm{i}}\right)\right)^5 \tag{7.1}$$

kde

$$\mathrm{R}_0 = \left(\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2}\right)^2 \tag{7.2}$$

Je potrebné dbať na to, aby platilo $\eta_1 \leq \eta_2$. V prípade, že $\eta_1 > \eta_2$ stačí jednoducho vymeniť hodnoty η_1 a η_2 .

Výsledky Schlickovej aproximácie sú veľmi podobné s výsledkami Fresnelových rovníc, avšak len pre malé rozdiely indexov lomu jednotlivých rozhraní η_1 a η_2 . Našťastie v reálnom svete sú vysoké hodnoty indexu lomu zriedkavé. Aby bolo použitie Schlickovej aproximácie prínosné, je pri implementácii treba dať pozor, aby vo vzorci 7.1 bola nákladná funkcia umocnenia rozložená a nahradená za násobenie. [18]

7.4 Textúry a textúrovanie

Po vypočítaní intenzity osvetlenia v kolíznom bode je nutné aplikovať textúru povrchu.

Textúrovací reťazec v OpenGL disponuje širokou paletou nastavení textúrovacích režimov, formátov a textúrovacích funkcií. V OpenGL sa tieto nastavujú pomocou `glTexEnv` funkcie.

Rayme zisťuje nastavenie textúrovacieho režimu a funkcie textúrovaného telesa zo `StateSet` objektov grafu a podľa toho zmiešava farbu podkladu s textúrou. Takto implementuje podmnožinu[8] komplexného textúrovacieho reťazca prítomného v OpenGL.

Počas textúrovania Rayme využíva implementovaný algoritmus bilineárneho filtrovania.

7.5 Phongova interpolácia

OSG poskytuje v bode kolízie implicitne normálu zasiahnutého primitíva, ale pre účely ray tracing aplikácie je nedostačujúca. Bez ďalšieho spracovania je takto scéna vykreslená s normálami *per-face*, namiesto *per-vertex*, t.j. obsahuje viditeľné artefakty a ostré prechody, tak ako scéna vykrelsená tieňovacím algoritmom *flat shading* (kapitola 4.2).

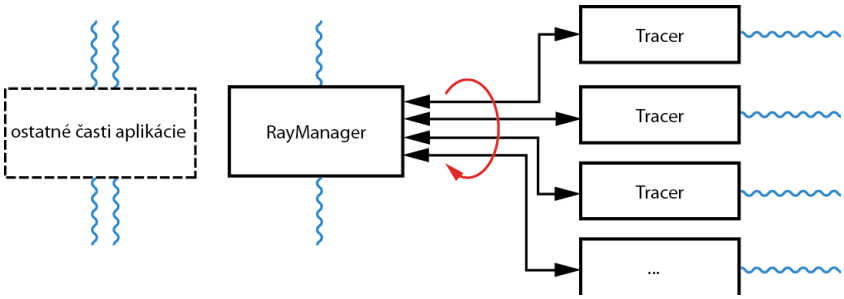
8 Implementované sú režimy GL_MODULATE, GL_REPLACE, GL_BLEND a GL_DECAL

Namiesto použitia implicitnej normály je možné s minimálnymi stratami na výkone interpolovať normály z vrcholov primitíva. OSG totiž implicitne poskytuje aj pomery vzdialeností kolízneho bodu od jednotlivých vrcholov zasiahnutého primitíva.

7.6 Výpočetné vlákna a paralelizácia výpočtu

Osvetľovanie scény pomocou algoritmu ray tracing je výpočetne náročné. Preto je tento proces potrebné optimalizovať a paralelizovať, aby bolo možné využiť maximum výpočetného výkonu poskytnutého platformou, na ktorej výpočet prebieha.

Vzhľadom na to, že vlnové vlastnosti svetla sú zanedbané, môžeme predpokladať, že dva a viac lúčov sa navzájom neovplyvňujú, sú teda úplne nezávislé a výpočet osvetlenia pre každý bod obrazu, resp. pre každý vysielaný lúč, je možné masívne paralelizovať.



Obrázok 7.4: schéma výpočetného jadra z pohľadu vláken

Obrázok ukazuje model vláken aplikácie Rayme. Správca lúčov `RayManager` alokuje lúče a priraďuje ich voľným výpočetným objektom typu `Tracer`, ktoré spúšťajú výpočet priesečníkov lúčov so scénou a následné vyhodnocovanie osvetlenia v kolíznych bodoch. `RayManager` v intervaloch iteruje cez všetky objekty `Tracer` a voľným objektom priradí novú sadu lúčov. Počet výpočetných vláken Rayme určí podľa počtu jadier CPU, ale je možné nastaviť ho manuálne. Pre optimálny výkon je vhodné nastaviť počet výpočetných vláken vyšší ako počet dostupných procesorov (jadier).

Rayme využíva knižnicu OpenThreads, ktorá je súčasťou OSG. Knižnica tvorí minimálnu ale kompletnú vrstvu potrebnú pre prácu s vláknami na rôznych platformách.

7.7 Systém súradníc

Ďalším z problémov typických pre 3D aplikécie sú prevody medzi súradnicovými priestormi. Využívajú sa na ne transformačné matice, ktorými sa body a vektory scény násobia. V Rayme sú všetky výpočty osvetlenia robené v priestore súradníc modelu, tzv. *model coordinate space*, je do nich teda nutné previesť aj vrhané lúče. Presnejšie ide o súradnice celej scény, *world coordinate space*, ktoré OSG odlišuje od súradníc lokálnych pre jeden objekt scény (*model coordinate space*).

Pri zobrazovaní objektov na priemetni je potrebné ich pretransformovať zo súradníc scény do súradníc okna. Na toto slúži matica M_{trans} vytvorená násobením matíc kamery – pohľadovej *(viewing)* a projekčnej *(projection)* – a matice, ktorá premietnuté body prevedie do súradníc okna *(window matrix[9])*. Tieto transformácie sú vykonávané aj v OpenGL.

$$M_{trans} = M_{viewing}\ M_{projection}\ M_{window} \tag{7.3}$$

Počiatočný bod primárnych lúčov leží na priemetni a ich smerový vektor smeruje do scény. Aby sme dokázali priemetňu umiestniť do scény, a tým aj primárne lúče, potrebujeme transformáciu inverznú k vyššie popísanej. Na tento účel vypočítame maticu inverznú k vyššie uvedenej matici M_{trans} – M_{trans}^{-1} – a následne ňou násobíme body priemetne.

Sekundárne lúče sú už implicitne vytvárané v súradniciach scény, so začiatkom v kolíznom bode otcovského lúča.

7.8 Integrácia s Lexolights

Rayme bol od začiatku vyvíjaný tak, aby ho bolo možné integrovať do projektu Lexolights alebo do iných projektov založených na OSG. Na obrázku 5.1 je zobrazená schéma toku dát v Rayme a tiež čiara znázorňujúca prístupový bod pre ostatné aplikácie. Rayme je možné integrovať do ľubovoľnej aplikácie, a to hneď dvoma spôsobmi:

- zabudovaním priamo do aplikácie
  - Rayme využíva vlastný `namespace`, nebude teda prekážať a pre svoj chod potrebuje len graf scény a nepovinné nastavenia
  - výstup poskytuje štandardne vo forme `osg::Image`
- využitím konzolovej funkcionality
  - možnosť spustiť Rayme pomocou dávkového súboru alebo priamo volaním systému poskytuje možnosť spolupráce s ľubovoľnou aplikáciou
  - výstup do súborového systému vo forme obrázka

7.9 Užívateľské dáta v modeloch

Lexolights pracuje s modelmi, ktoré majú niektoré parametre pre realistické zobrazovanie ručne vkladané do popisu jednotlivých častí modelu – v OSG sa tieto užívateľom definované informácie vkladané do uzlov nazývajú užívateľské dáta, pre potreby tejto práce ich budem nazývať anotáciami.

Rayme používa dve rovnaké anotácie ako Lexolights – `Material.castShadow` a `Material.reflectiveColor`. K nim som kvôli možnosti nastaviť aj index lomu pridal ešte jednu – `Material.refractiveIndex`.

- `Material.castShadow`
  - táto anotácia určuje, či teleso vrhá tieň alebo nie

9 window matrix je matica, ktorá sa používa v OSG

    - používa sa hlavne pri geometrii svetiel, tieň nevrhajú napríklad tienidlá svietidiel
    - keď tieňový lúč zasiahne uzol s touto anotáciou nastavenou na `true`, tento priesečník sa ignoruje
- `Material.reflectiveColor`
    - určuje farbu a intenzitu odrazov od objektov s touto anotáciou odrážajú lúče
    - slúži pre nastavenie zrkadiel a iných reflexných plôch
- `Material.refractiveIndex`
    - určuje index lomu tohto objektu
    - platí len pre transparentné objekty

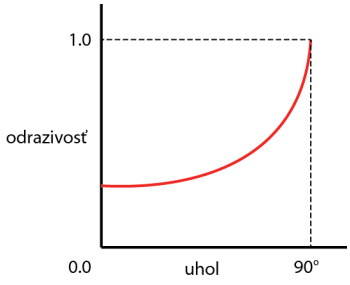
Rayme však bol navrhnutý tak, aby pracoval ešte univerzálnejšie – ak nie sú tieto anotácie prítomné, snaží sa ich odhadnúť sám.

7.10 Aproximácia povrchových vlastností modelu

Ak nemá Rayme k dospozícii anotácie nesúce údaje o odrazoch a indexe lomu, musí ich sám nejako získať. Nanešťastie, väčšina ray tracing aplikácií je postavená tak, že sa pre ne skonštruujú špeciálne modely, ktoré obsahujú množstvo informácií o správaní sa povrchu materálov (napríklad BRDF), a tie sa následne renderujú. Výstup takýchto aplikácií je veľmi kvalitný. Ale čo keď potrebujeme fotorealisticky vygenerovať nejakú obyčajnú scénu bez týchto informácií? Rayme je primárne určený ako renderovací nástroj pre Lexolights a iné projekty založené na OSG, má poskytovať možnosť vytvárať pekné snímky bez nutnosti prerábania existujúcich modelov.

Nakoľko som nenašiel žiadne zdroje, ktoré by sa zaoberali takouto tématikou, rozhodol som sa pre experiment. Vychádzajúc z Phongovho svetelného modelu sú pre odrazy použiteľné dva atribúty – odlesková zložka (*specular*) a index ostrosti odleskov, *shininess*.

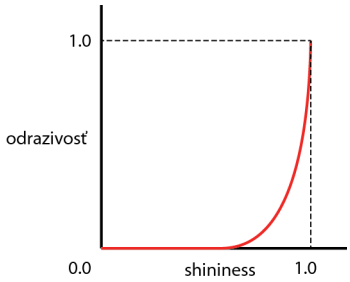
Prvé pokusy s odleskovou zložkou svetla viedli k zostrojeniu rovnice odrazivosti na základe uhla. Princíp bol podobný, ako je princíp Fresnelových vzorcov pre rozhranie – pod malým uhlom bol odraz slabý a pod veľkým uhlom bol odraz intenzívny. Farba sekundárneho lúča následne modulovala odlsekovou zložkou materiálu. Vizuálne výsledky tohto experimentu sú zverejnené v dodatku B, obrázky B.1 a B.2.



Obrázok 7.5: približý priebeh odrazivosti na povrchu – funkcia uhla

Toto správanie som však po niekoľkých pozorovaniach zavrhol. Hlavným dôvodom bol fakt, že odlesková zložka slúži na modelovanie odleskov a nie odrazov a tiež nič nehovorí o intenzite odrazu a kvalite povrchu.

Preto som sa priklonil k druhej možnosti – *shininess*. Táto premenná predstavuje pre materiál stupeň vyleštenia. Vyššia hodnota spôsobuje ostrejšie odlesky a nižšia hodnota menej ostré. Táto premenná teda už o niečo viac hovorí o kvalite a povahe povrchu telesa. Zmenu intenzity odrazu na základe uhla lúča voči normále som tiež zavrhol, pretože takto sa dá aproximovať povrch len niektorých materiálov, tie sú väčšinou transparentné a Rayme na ne používa Fresnelove vzorce.



Obrázok 7.6: približý priebeh odrazivosti na povrchu – funkcia premennej shininess

V tomto druhom prípade je teda odrazivoť funkciou len jednej premennej, *shininess*. V prvej polovici je odrazivosť nulová, pretože kvalita povrchu je slabá. V druhej polovici rozsahu *shininess* však rastie kvadraticky. Pre hodnotu *shininess* 1 je rovná odrazivosti zrkadla.

Index lomu by bolo možné aproximovať na základe stupňa priehľadnosti objektu. Avšak tento odhad by nekorešpondoval s realitou a navyše v bežných scénach sa najviac vyskytuje sklo a voda, oboje takmer 100% priehľadné. Ako základnú hodnotu indexu lomu som teda zvolil kompromis medzi indexmi lomu spomínaných látok. Tento sa použije pre všetky transparentné objekty, pokiaľ, samozrejme, nebude nastavený v modeli manuálne pomocou anotácií.

Uvedené parametre nemusia užívateľovi vyhovovať, preto mu Rayme poskytuje možnosť dodatočne upraviť intenzitu odrazov a lomených lúčov pomocou dvojice premenných `reflectionTrimmer` a `refractionTrimmer` dostupných z UI aj príkazového riadka.

7.11 Renderovanie scén s veľkým počtom svetiel

Jednou zo zaujímavých vlastností Rayme je jeho schpnosť pracovať so scénami, kde počet svetiel prekračuje maximum pre danú platformu. Grafické karty majú totiž limit počtu svetiel, hoci dnes relatívne vysoký.

Tento problém je možné vyriešiť pomerne jednoduchým trikom, kedy sa model upraví tak, aby obsahoval malé množstvo svetiel, následne sa s týmto okresaným modelom pracuje v interaktívnom režime, kde je možné uložiť želané nastavenie kamery. Táto kamera sa následne použije pre renderovanie plnohodnotnej scény v konzolovom režime, ktorý nepotrebuje grafickú kartu na zobrazovanie, spomínaný limit preto pre neho neplatí.

8 Vyhodnotenie výstupu a výkonu

Kapitola hodnotí výstup aplikácie Rayme z estetického aj fyzikálneho hľadiska. Porovnáva výsledky s vlajkovou loďou medzi ray tracermi, s programom POV-Ray[10], a s real-time zobrazovacou aplikáciou pre blízko fotorealistické osvetlovanie, Lexolights[11]. Taktiež hodnotí výkon Rayme z pohľadu výpočetnej náročnosti. V záverečnej časti kapitoly sa nachádzajú návrhy ďalšieho možného vývoja aplikácie.

8.1 Rayme, Lexolights a POV-Ray v skratke

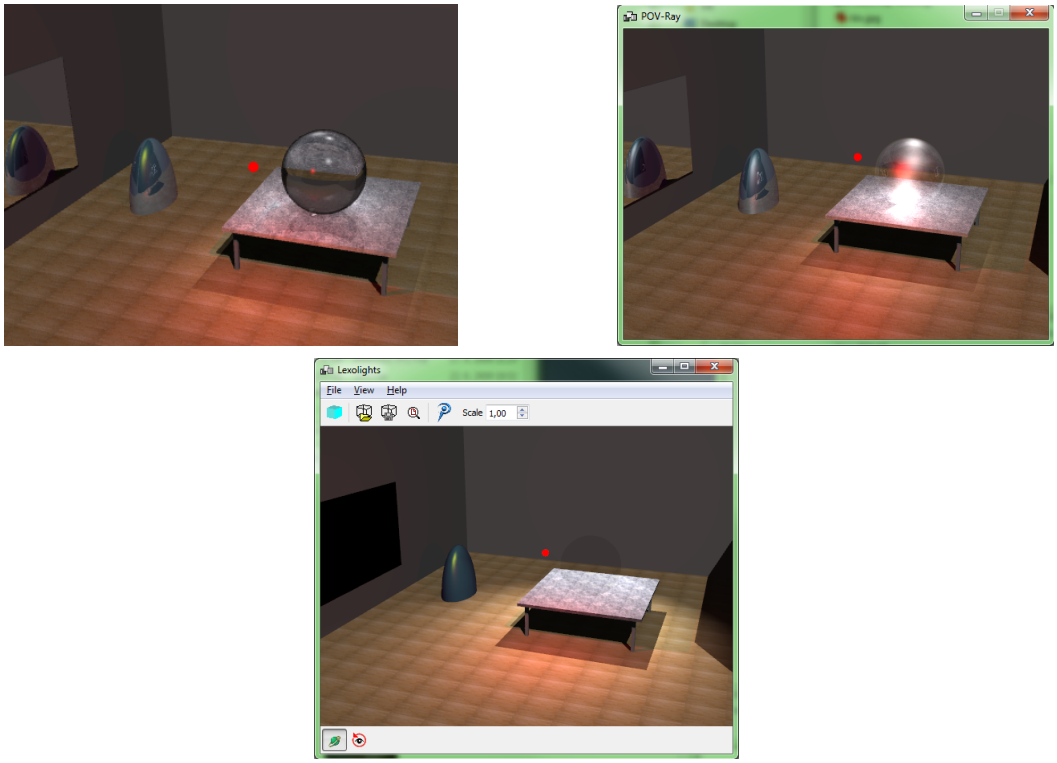
Rayme a Lexolights si nie sú veľmi podobné aplikácie, hoci ich cieľ je rovnaký – vizuálne čo najlepší výstup. Kým Lexolights generuje niekoľko snímkov za sekundu a využíva silu grafických kariet prostredníctvom shaderov, Rayme je off-line renderovací nástroj, ktorý generuje snímky pomaly. Lexolights disponuje funkciami pre realistické zobrazovanie tieňov, ktoré Rayme implicitne generuje. Lexolights však nie je schopný zobrazovať zrkadlenie a lom svetla, čo je doménou Rayme a preto Rayme nechce byť súperom Lexolights, ale jeho doplnkom.

POV-Ray je už niekoľko desaťročí vyvíjaný nástroj pre off-line generovanie snímok s veľmi vysokou úrovňou realistickosti. POV-Ray disponuje celou sadou mocných algoritmov, ktoré v kombinácii podávajú presvedčivý výkon. Je založený na jadre pozostávajúcom z ray tracing algoritmu, snímky však obohacuje photon mapping algoritmom i rádiozitou. Je schopný veľmi dobre pracovať s optickými javmi lomu svetla a odrazov. Pri lome svetlo dokonca rozkladá na farebné spektrum. Renderuje kaustické javy, hmlu, rozostrenie ohniskom či pohybom a iné efekty. Podporuje najrôznejšie triky s textúrami (bump mapping, ripple mapping a pod.).

Avšak, hoci je POV-Ray naozaj veľmi pokročiný nástroj, na svoj chod potrebuje model doplnený o dodatočné informácie. V tomto ohľade podáva Rayme lepší výkon. Pre porovnanie ponúkam dve snímky, ďalšie je možné nájsť v dodatku B.

10 URL <http://www.povray.org/>

11 URL <http://sourceforge.net/projects/lexolight/>



Obrázok 8.1: porovnanie výstupu Rayme (vľavo), POV-Ray a Lexolights (naspodu), rovnaká scéna

Na obrázku 8.1 je vidno rozdiely v generovaní tej istej scény. Jedná sa o jednoduchú scénu bez dodatočných informácií. POV-Rayu zrejme chýba nastavenie indexu lomu sklenej gule na stole a Lexolights nezobrazuje odrazy na zrkadle ani na objektoch v scéne.

8.2 Časová náročnosť výpočtov

Z popiskov k obrázkom v dodatku B je vidno vysokú časovú náročnosť výpočtov. Hoci som sa snažil kód optimalizovať, hrdlom aplikácie zostáva výpočet priesečníkov s geometriou, hlavne iterovanie cez uzly grafu a testovanie na bounding box, obalové telesá, pre každé `Drawable`. OSG evidentne nie je stavaná na tento typ využitia. k-D stromy sú vytvárané pre každý uzol s geometriou zvlášť (pre každé `Drawable`). Ak lúč BB pretne, použije sa k-D strom na zistenie zásahu konkrétneho primitíva, tento výpočet je rýchly.

Aplikácia beží relatívne rýchlo pre scény s nízkym počtom `Drawables` a spomaľuje s každým ďalším pridaným `Drawable`. Počet polygónov sa vďaka k-D stromom veľmi neprejavuje. Nasledujúce tabuľky problém demonštrujú.

Model:

- mirror2.iv
  - 129 Drawables, 16 tisíc vrcholov, 5311 trojuholníkov

Rozlíšenie	Nastavenia, derivát	Čas
400x300	Bez rekurzie	70 -80 s
400x300	Vysoký počet odrazov	130 – 150 s
400x300	Sklenená doska stolu, pohľad zblízka	150 - 250 s
400x300	4x FSAA, transparentná guľa	cca 350 s

Model:

- treppe.osg
  - 14 Drawables, 248 tisíc vrcholov, 83 tisíc trojuholníkov

Rozlíšenie	Nastavenia, derivát	Čas
400x300	Bez rekurzie	20-25 s
400x300	Vysoký počet odrazov	40 s
400x300	Sklenená doska stolu, pohľad zblízka	150 - 250 s
800x600	4x FSAA	cca 360 s

Model:

- belaria.iv
  - 12 tisíc Drawables, 1 milión vrcholov, 360 tisíc trojuholníkov

Rozlíšenie	Nastavenia, derivát	Čas
400x300	Bez rekurzie	200 s
400x300	Normálny režim, veľké množstvo transparentných objektov v zábere kamery	200-250 s
400x300	8x FSAA	6000 s
800x600	8x FSAA	Cca 7 hodín[12]

Z časov v druhej tabuľke je vidno, že na rýchlosť renderovania má vplyv počet `Drawable` objektov a nie počet polygónov scény.

Bohužiaľ konkrátnejšie profilovacie informácie som nezískal, pretože profiling pomocou gprof nič neukázal. Zrejme kvôli tomu, že nie je schopný merať prácu OSG knižnice. Medzi priloženými súbormi na CD je výstup behu programu.

12 všetky merania prebehli na nevýkonnom notebooku

8.3 Možné vylepšenia a ďalší vývoj

Namiesto k-D stromov vybudovaných OSG pre každý Drawable zvlášť by bolo vhodné zostaviť jeden k-Dstrom obsahujúci celú geometriu scény.

Problém by však nastal zrejme v práci s grafom scény, pretože takýto strom obsahujúci celú geometriu v jednom uzle nie je schopný pojať `StateSety` jednotlivých uzlov originálneho stromu, t.j. vlastnosti materiálov, textúry, textúrovacie módy a pod., a bolo by nutné vytvoriť a udržiavať reláciu medzi originálnym stromom a k-D stromom s geometriou. Otázne je, či by v konečnom dôsledku došlo k zrýchleniu, alebo by réžia bola príliš veľká.

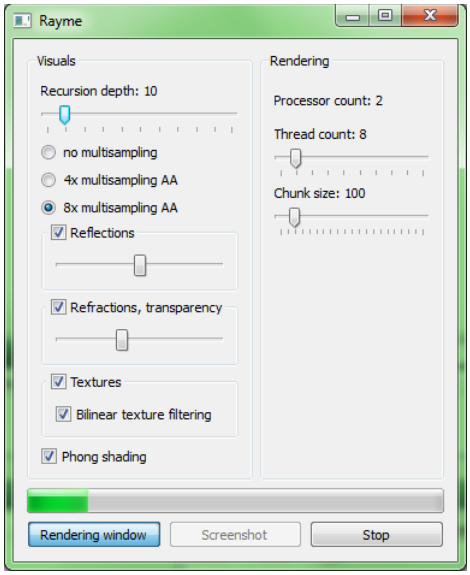
Ďalším vylepšením by mohlo byť použitie zásobníkov a medzistupňových bufferov namiesto rekurzie v prípade sekundárnych lúčov. Takto byt tiež bolo možné scénu zobraziť pomerne rýchlo a jej vzhľad by sa postupne zlepšoval, podobne ako tomu je pri path tracing algoritmoch.

Počas práce na tejto aplikácii ma napadla myšlienka pretvoriť tento nástroj na zobrazovanie modelov a renderovanie na komplexnejší nástroj s možnosťou nastavovania parametrov svetiel a objektov, prípadne schopnosťou meniť ich polohu.

Rayme disponuje debugovacím módom, v ktorom sa namiesto celej scény naraz renderuje po bodoch, body volí užívateľ. Bolo by zaujímavým rozšírením tieto lúče zobrazovať v scéne, takto by program mohol slúžiť aj ako dobrý výukový alebo názorný demonštračný softvér pre potreby výuky metód založených na sledovaní lúčov.

Pre Rayme som navrhol a zostrojil grafické užívateľské rozhranie v QT, bohužiaľ sa mi nepodarilo sprevádzkovať spoluprácu OSG a QT, nemohol som preto rozhranie napojiť na jadro Rayme.

Vylepšiť by si zaslúžilo aj renderovacie jadro, vhodné by boli napríklad efekty rozostrenia na šošovke alebo mäkké tiene a kaustické javy.



Obrázok 8.2: GUI v QT pre Rayme

9 Záver

V tejto práci som preskúmal jednu z techník fotorealistického zobrazovania, pôvodnú rekurzívnu metódu ray tracing, ktorej autorom je Turner Witted. Ide o veľmi agilnú metódu, ktorej vývoj stále pretrváva.

Fotorealistické metódy, ray tracing nevynímajúc, majú však problém s výpočetnými nárokmi. Hoci má metóda dnes už aj masívne hardvérom urýchľované implementácie schopné podávať niekoľko snímok za sekundu, pri zložitejších scénach pri plných detailoch je výpočet stále pomalý. Pri výpočte osvetlenia scény zaberie najväčšie množstvo času výpočet priesečníkov lúčov s objektmi scény. Ako som sa mal možnosť presvedčiť, ide naozaj o veľmi pomalú operáciu, ktorú je však možné výrazne zrýchliť použitím štruktúr na organizáciu priestoru, napríklad k-D stromov, ktoré sa používajú najbežnejšie.

Navrhol som a implementoval plnohodnotný program, ktorý využíva práve spomínaný ray tracing algoritmus na výpočet globálneho osvetlenia scény. Svetelný model rieši odrazy svetla, lom a jeho útlm v prostredí. Tento realistický svetelný model je založený na Phongovom osvetľovacom modeli a Fresnelových rovniciach na výpočet odrazu a lomu svetla. Okrem toho disponuje funkciami na zlepšenie a vyhladenie výstupu, anti-aliasingom a filtrovaním textúr. Program pri osvetľovaní scény vďaka implementácii viacerých paralelných výpočetných vláken využíva všetky dostupné výpočetné jednotky.

Program je vďaka dvom režimom – interaktívnemu a konzolovému – použiteľný ako v demonštračných aplikáciách, tak aj na dávkové výpočty a je možné ho pripojiť k alebo integrovať do iných aplikácií.

Nakoľko zámer programu bola schopnosť spracovať ľubovoľné scény určené pre interaktívnu 3D grafiku, je tento program schopný celkom dobre s takýmito scénami pracovať a výstup vyzerá slušne aj bez dodatočných informácií poskytnutých modelom.

Program je založený na OSG, ktoré je výborným základom pre tvorbu 3D grafických programov, avšak v mojom prípade využitie jeho integrovanej podpory zisťovania priesečníkov viedlo k relatívne vysokej časovej penalizácii výpočtu. Zrýchlenie tohto výpočtu by malo byť primárnym námetom na ďalšie pokračovanie vývoja programu. Taktiež by bolo zaujímavé implementovať rôzne efekty, napríklad rozostrenie šošovkou objektívu alebo rozmazanie pohybom.

Hoci implementácia ray tracing algoritmu nie je prevratná, práca na tomto projekte bola pre mňa veľmi prínosná, hlavne v podobe načerpaných informácií z prostredia fotorealistického zobrazovania a práce s OSG. Zrejme to z tejto práce nevyplýva priamo, ale práve práca so spomínaným frameworkom mi zabrala najviac času, nakoľko som ho potreboval na pokročilé činnosti.

Literatúra

[1] Appel, A.: *Some techniques for shading machine renderings of solids*, AFIPS 1968 Spring Joint Computer Conference, 37-45, 1968

[2] Bui-Tuong Phong: *Illumination for computer generated images*, Comm. ACM 18, 311-317, 1975.

[3] Blinn, J. F.: *Models of light reflection for computer synthesized pictures*, SIGGRAPH 1977, San Jose, Calif., 192-198, 1977.

[4] Witted, T.: *An Improved Illumination Model for Shaded Display,* CACM, 343-349, 1980

[5] Kajiya, J. T.: *The rendering equation*, SIGGRAPH 1986, 143-150, 1986

[6] Lafortune, E. P., Willems, Y. D.: *Bi-directional Path Tracing*, Department of Computer Science, Katholicke Universiteit Leuven, Leuven, Belgium, 1993

[7] Halliday, D., Resnick, R., Walker, J.: *Fyzika. Vysokoškolské učebnice obecné fyziky. Část 4: Elektromagnetické vlny – optika – relativita*. Brno a Praha: VUTIUM a Prometheus, 2000. ISBN 80-214-1868-0.

[8] Purcell, T. J., Buck, I., Mark, W.R., Hanraham, P.: *Ray tracing on programmable graphics hardware*, SIGGRAPH 2002, 703-712, 2002

[9] Wikipedia: *k-d tree* [online][cit. 2012-1-10], 2011.
URL <http://en.wikipedia.org/wiki/K-d_tree>

[10] Parker, S., Shirley, P., Smits, B.: *Single Sample Soft Shadows* [online], University of Utah, Utah, USA, 1998.
URL <http://www.cs.utah.edu/~bes/papers/coneShadow/shadow.html>

[11] Wikipedia: *Optics* [online][cit. 2012-1-10], 2011.
URL <http://en.wikipedia.org/wiki/Optics>

[12] Wikipedia: *Scene Graph* [online][cit. 2012-1-10], 2011.
URL <http://en.wikipedia.org/wiki/Scene_graph>

[13] Herout, A.: *Počítačová grafika,* Opora a slajdy k predmetu PGR, Vysoké učení technické v Brně, 2008

[14] Wikipedia: *3D rendering* [online][cit. 2012-1-10], 2011.
URL <http://en.wikipedia.org/wiki/3D_rendering>

[15] Wikipedia: *Ray tracing* [online][cit. 2012-1-10], 2011.
URL <http://en.wikipedia.org/wiki/Ray_tracing>

[16] Wright, R. S., Lipchak, B., Haemel, N., Sellers, G.: *OpenGL SuperBible: Comprehensive Tutorial and Reference*, 5.edícia, Addison-Wesley, 2010, ISBN 978-0321712615

[17] Martz, P.: *OpenSceneGraph Quick Start Guide*, 2007,
URL <http://www.osgbooks.com/books/osg_qs.html>

[18] De Greve, B.: *Reflections and Refractions in Ray Tracing*, 2007,
URL <http://www.flipcode.com/archives/reflection_transmission.pdf>

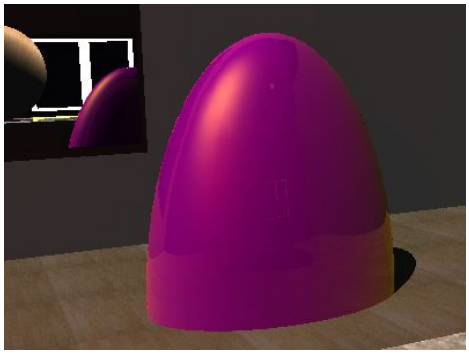
[19] Heckbert, P.: *Facet-Gouraud-Phong,* obrázok, autor modelu Parke, F., [online]
[cit. 2012-5-20]
URL <http://www.cs.cmu.edu/~ph/nyit/>

[20] Wikipedia: *Anisotropic Filtering* [online][cit. 2012-5-20], 2011.
URL <http://en.wikipedia.org/wiki/Anisotropic_filtering>

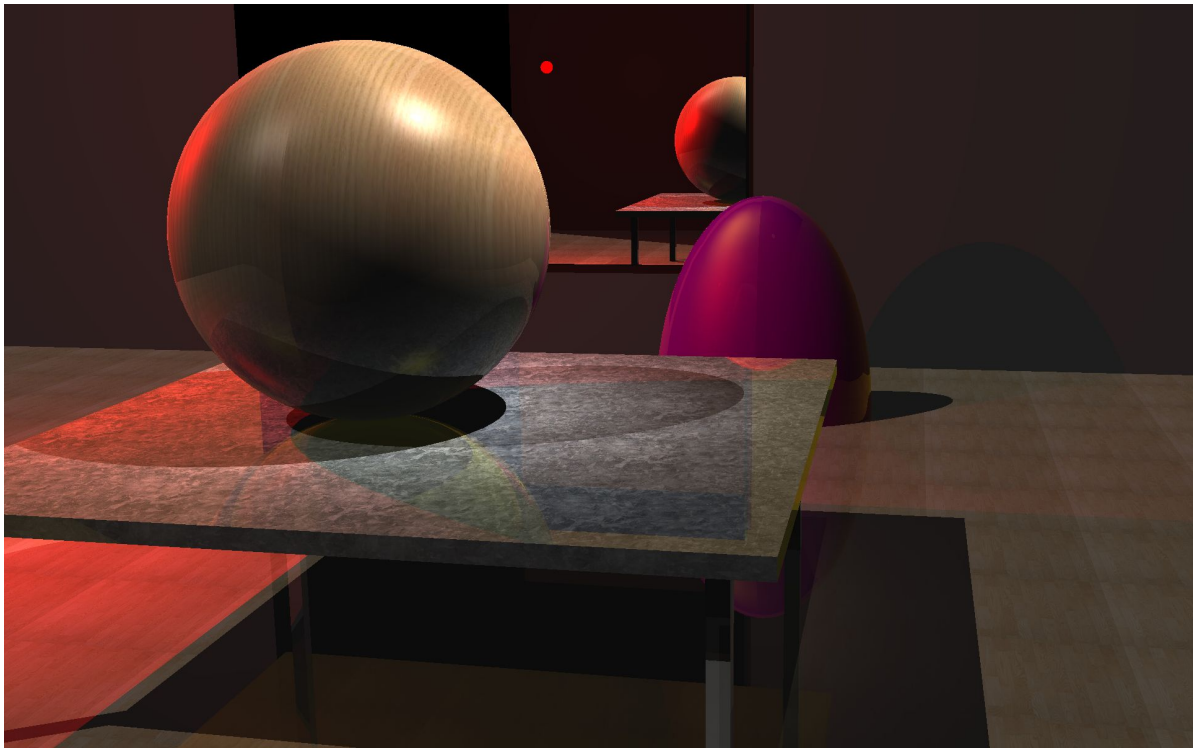
Dodatok A – Doplňujúce obrázky

Obrázok A.1: porovnanie výsledku trilineárneho a anizotropného filtrovania textúry, prevzaté z [20]

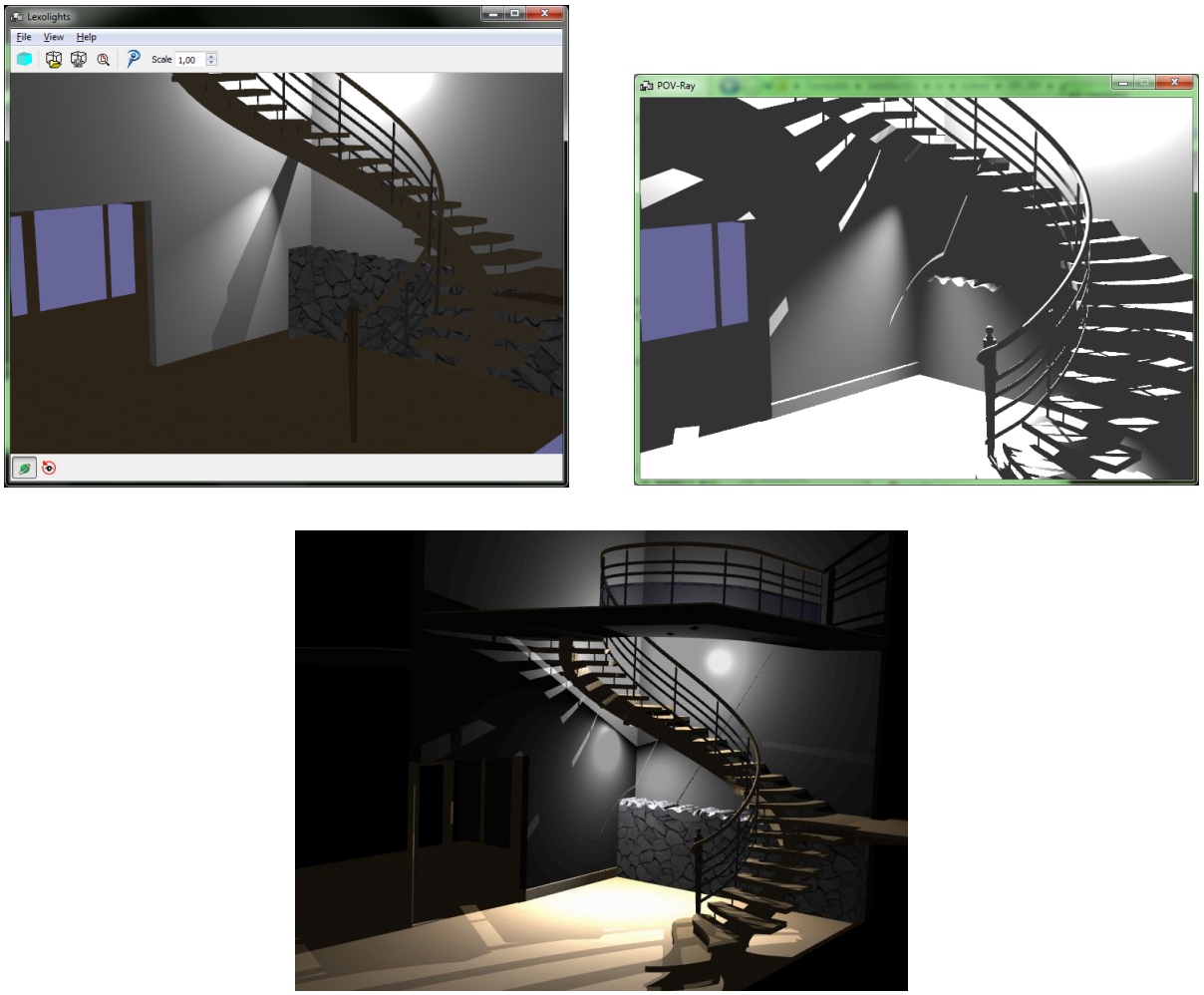
Dodatok B – Doplňujúce snímky scén



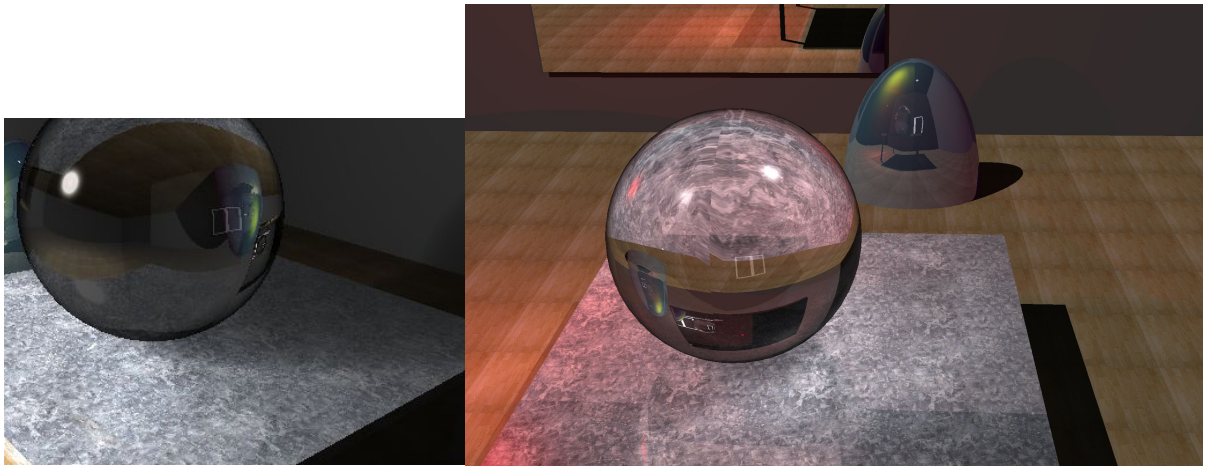
Obrázok B.1: odrazy na objekte, intenzita je funkciou uhlu dopadu, odrazy v smere normály sú minimálne, pri väčšom uhle sa ich intenzita zvyšuje



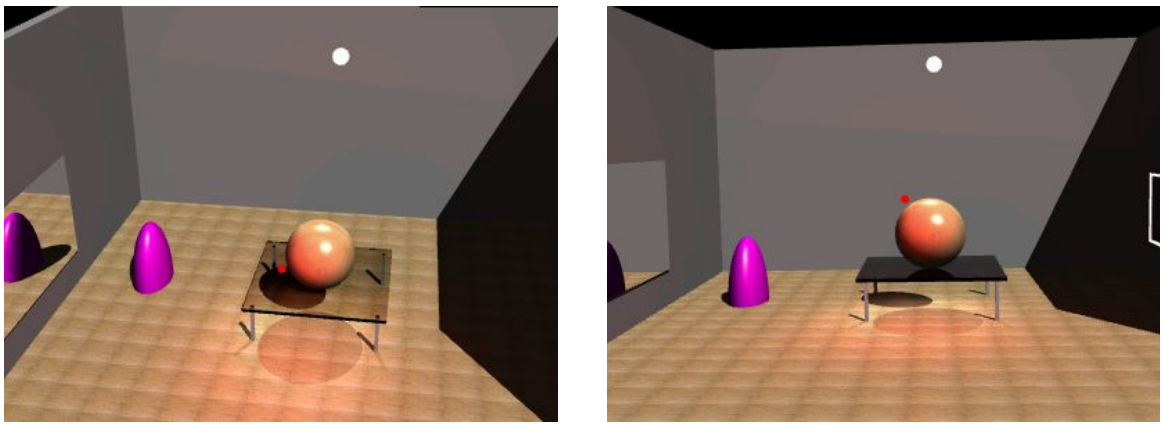
Obrázok B.2: na guli postrehnuteľná premenlivá intenzita odrazov



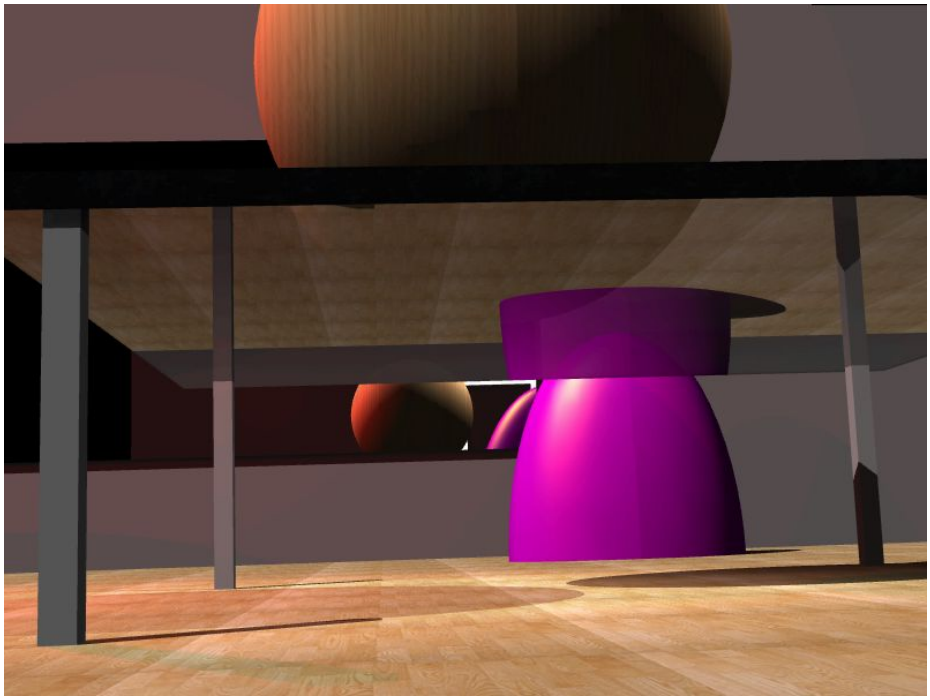
Obrázok B.3: porovnanie vizualizácie tej istej scény, Lexolights, POV-Ray, Rayme



Obrázok B.4: riešenie priehľadnosti v Rayme



Obrázok B.5: ukážka Fresnelových vzorcov v praxi, tá istá scéna rovnaké nastavenia, iný uhol kamery



Obrázok B.6: priehľadný stôl pod veľkým uhlom

Zoznam použitých skratiek a výrazov

AA	Anti aliasing, technika odstraňujúca efekt aliasingu
API	Application Programmable Interface, rozhranie pre programovanie aplikácií
BB	bounding box, technika obalových telies
GLSL	OpenGL Shading Language
OpenGL	Open Graphics Library
OSG	Open Scene Graph
UI	angl. user interface, užívateľské rozhranie
real-time	označenie aplikácií bežiacich v reálnom čase

Zoznam príloh

Príloha 1.: CD s programom a dokumentáciou, väčšie množstvo vygenerovaných scén
Príloha 2.: Manuál na obsluhu programu Rayme