VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

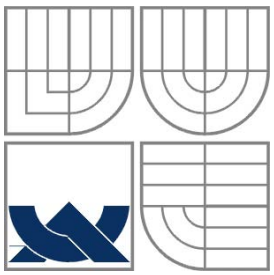
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OPTIMALIZOVANÉ SLEDOVÁNÍ PAPRSKU

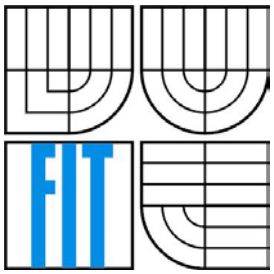
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN TROJAN
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

OPTIMALIZOVANÉ SLEDOVÁNÍ PAPRSKU

OPTIMIZED RAY TRACING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN TROJAN

VEDOUCÍ PRÁCE
SUPERVISOR

DOC. DR. ING. PAVEL ZEMČÍK

BRNO 2011

Abstrakt

Tato práce se zabývá optimalizačními algoritmy, které se snaží zefektivnit a zrychlit metodu sledování paprsku. Je zde řešeno několik akceleračních struktur a metoda pro snížení náročnosti výpočtu průsečíku se složitými tělesy. Část textu se rovněž zabývá samotnou metodou ray tracing a pojmy s ní spojené. Pro účely práce je vytvořena aplikace, na které je implementován algoritmus sledování paprsku za použití uniformní mřížky pro jeho optimalizaci.

Abstract

This thesis deals with optimization methods for ray tracing. There are discussed several methods of acceleration structures and method for reducing the complexity of calculating intersection with complex objects. Part of the text also deals with the ray tracing method and the concepts associated with this technique. For the purposes of this thesis have been created application, which is implemented ray tracing algorithm using a uniform grid for optimization.

Klíčová slova

realistické zobrazování, metoda sledování paprsku, počítačová grafika, optimalizační metody sledování paprsku, obalová tělesa, hierarchie obálek, dělení prostoru

Keywords

realistic rendering, ray tracing, computer graphics, optimization methods for ray tracing, bounding volumes, bounding volume hierarchy, spacial subdivisions

Citace

Trojan, Martin*: Optimalizované sledování paprsku*. Bakalářská práce, Brno, FIT VUT v Brně, 2011

Optimalizované sledování paprsku

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Pavla Zemčíka a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Trojan
15. května 2011

Poděkování

Tímto bych chtěl poděkovat Doc. Dr. Ing. Pavlu Zemčíkovi, za jeho věcné připomínky, rady a spolupráci na této bakalářské práci. Dále bych chtěl poděkovat všem, co mi pomohli s testováním, korekcí či užitečnou radou, zejména své mamince a přítelkyni. Svým rodičům a prarodičům za podporu během celého studia.

Obsah

Seznam obrázků

1 Úvod

Počítače, počítačové hry a animované filmy jsou pro značnou část lidí z rozvinutých zemí neodmyslitelnou součástí všedního života. Postupem času se u těchto odvětví klade čím dál větší důraz na realističnost a propracovanost. Co bylo pro hráče dobré před deseti lety, dnes nemusí platit. A tak se modelování 3D scén a syntetizace obrazu, což je odvětví počítačové grafiky, které se zabývá tvorbou obrazů napodobujících reálný svět, stávají nedílnou součástí dnešních počítačů. K dnešnímu stavu však vedla dlouhá cesta pokusů a omylů s cílem generování obrazů, co nejvěrněji odpovídajícím realitě.

Metoda sledování paprsku, jejíž vývoj probíhá již od konce sedmdesátých let, patří mezi stěžejní postupy moderní počítačové 3D grafiky. Pro účely této práce budeme používat anglický termín Ray tracing, jelikož je tak celosvětově uznáván a i v českých kruzích se užívá tohoto termínu. Algoritmus, jímž dochází ke generování obrazu, je ceněn nejen pro svou jednoduchost spojenou s elegantností, ale také pro kvalitu a množství optických jevů, které dokáže již ve své základní podobě syntetizovat, jako třeba stíny, odlesky, odrazy okolních těles a průchod světla skrze rozdílná optická prostředí. Díky tomu tato metoda generuje fotorealistické zobrazení generované scény.

Jedním z hlavních problémů, spojených s použitím této metody, je její značná časová náročnost. Mnoho let se hledají postupy a algoritmy, které zvyšují rychlost a efektivitu zpracování. Tato práce se těmito algoritmy a optimalizačními metodami zabývá a některé vybrané metody jsou implementovány v ray traceru, který byl zhotoven spolu s prací.

Existuje celá řada metod pro optimalizaci, ať už zrychlování výpočtů průsečíků paprsků s tělesy nebo různá dělení prostoru do struktur. Algoritmy se liší v tom, jak postupovat při dělení prostoru, jak provádět jeho třídění a jak postupovat při následném prohledávání. K nejrozšířenějším strukturám, uchovávající informace o rozdělení prostoru, patří pravidelné mřížky, binární stromy, kD–stromy, oktalové stromy a další. Přesnějším cílem této práce je tedy nastudovat a analyzovat některé algoritmy, které optimalizují základní algoritmus ray tracingu a vytvořit aplikaci schopnou zobrazovat grafické scény pomocí metody sledování paprsku s využitím optimalizačních struktur.

První část práce se zabývá základním pojednáním o ray tracingu a o základních pojmech z počítačové grafiky a matematiky použité při implementaci výsledné aplikace. Druhá kapitola je zaměřena na algoritmy pro optimalizaci. Následuje shrnutí současné situace okolo této metody a ve čtvrté kapitole je popsána implementace výsledné aplikace a výsledky z ní zjištěných. V závěru práce jsou zhodnoceny dosažené výsledky a uvedeny možnosti dalšího rozšíření aplikace.

2 Shrnutí současného stavu

Následující kapitola se věnuje pojmům, které byly použity při řešení této práce. Jde o shrnutí důležitých údajů a poznatků, avšak nejedná se o úplný přehled.

S vývojem počítačové grafiky postupně vznikalo velké množství algoritmů se zaměřením na zobrazování prostorových dat, kdy mnohé z nich byly prostými empirickými modely a velmi hrubě aproximujícími skutečnost. S příklonem k realističtější grafice se však nacházely nové metody, které se snažili o tzv. fotorealistické zobrazení. Jednou z nich je i sledování paprsku (ray tracing).

Obrázek 1: Obrázek generovaný metodou ray tracing [20]

V oblasti počítačové grafiky se pod pojmem ray tracing skrývá matematická metoda pro renderování (výpočtu a zobrazení) 3D počítačového obrazu. Umožňuje reálné zobrazení jevů za pomoci jiných technik vůbec, či jen stěží dosažitelných, jako jsou např. odrazy a odlesky objektů, lom světla v objektech, rozptyl a chromatické aberace. Dosahuje tím přímého šíření a sledování cesty světelných paprsků a simulaci účinků, které při interakci s povrchem modelovaného objektu s paprskem mohou nastat. To je však zaplaceno vysokou výpočetní náročností. Tím se stává ray tracing vhodný spíše pro aplikace, kde se obrazy vygenerují dopředu tj. fotografie, filmové a televizní efekty. Kdežto pro real-time aplikace, jako jsou počítačové hry, kde je rychlost zpracování obrazu nutná, je ray tracing bez jakékoliv optimalizace méně vhodný. Jacco Bikker ve své práci [4] z roku 2007 však popisuje možnosti interaktivního použití metody sledování paprsku a předkládá konkrétní implementaci zobrazující netriviální scény v reálném čase.

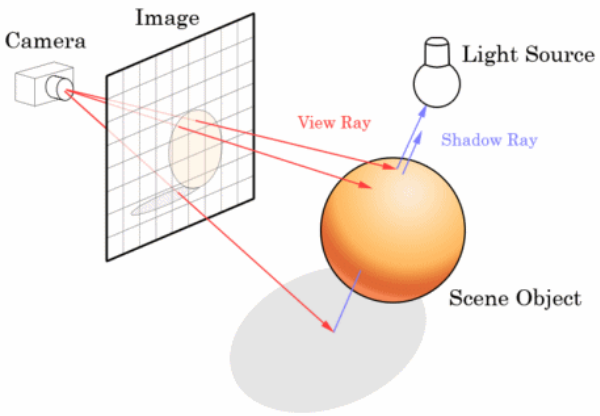
Obrázek 2: Turner Whitted (2005)

Metoda sledování paprsku byla poprvé představena v roce 1980 v práci [5] Turnera Whitteda. V této metodě vychází z jednodušší verze algoritmu, kterou je vrhání paprsku (Ray casting), a byla původně vytvořena pro zobrazování modelů vzniklých metodou Konstruktivní geometrie těles (Constructive Solid Geometry).

Ray tracing se řadí mezi metody vycházející od pozorovatele. Tyto metody jsou založeny na zpětném sledování paprsku, tedy z místa pohledu pozorovatele, resp. skrze rovinu generovaného obrazu zpět do scény.

Principy, na kterých jsou tyto algoritmy vystavěny:

- Většina paprsků, vyslaných ze světla, se nikdy nedostane k pozorovateli; nemusíme je tedy uvažovat, protože jsou pro výsledný obraz irelevantní
- Helmholtzův princip reciprocity, zaručující volnost ve směru sledování paprsku, tedy sledování od zdroje světla k pozorovateli nebo naopak, poskytuje stejnou informaci.
- Metody vycházející od pozorovatele jsou pohledově závislé, tj. změní-li se pozice pozorovatele, je nezbytné provést znovu celý výpočet osvětlení.



Obrázek 3: Ray tracing algoritmus generuje obraz rozšířením paprsků do scény [32]

Na základě principů, které byly popsány výše, tedy paprsek sledujeme zpětně, tzn. ve směru od pozorovatele. Na obrázku 3 můžeme vidět, že paprsek je vyslán z kamery (oka pozorovatele) do scény skrze stínítko představující matici bodů obrazovky. Je výpočetně velmi náročné sledovat všechny paprsky ze zdrojů světla. Projekční paprsky vysíláme přes pixely obrazu scény a hledáme, co je vidět v daném pixelu a jakou světelnou energii paprsek přenáší. V praxi však tento jev probíhá opačně.

3D scéna je matematicky popsána programátorem nebo pomocí vizuálních nástrojů. Objekty ve scéně i simulované paprsky jsou určeny parametrickými rovnicemi. Hledání průsečíku paprsku s jednotlivými objekty tedy spočívá v řešení soustavy rovnic objektu a paprsku.

Tato metoda zahrnuje efekty vznikající vzájemnou interakcí objektů ve scéně (tj. například odrazy ostatních těles na povrchu lesklého objektu a lom světla při průchodu průhledným tělesem) a dokáže určit stíny vržené různými tělesy; tyto stíny jsou však ostře ohraničeny.

2.1 Popis algoritmu

Pod pojmem sledování paprsku se rozumí algoritmus, popsaný T. Whittedem.
Základ metody je společný s metodou vrhání paprsku (ray-casting):

1. Vyslání primárního paprsku z kamery skrze mřížku generovaného obrazu.
2. Nalezení prvního průsečíku s objekty ve scéně.
3. Výpočet lokálního osvětlení v průsečíku
4. Vypočtená barva průsečíku je nastavena na pozici v mřížce, kterou byl vyslán primární paprsek.

V metodě sledování paprsku, po nalezení průsečíku primárního paprsku oka (kamery) s objektem, se ve 3. bodu algoritmu vytvoří řada dalších (sekundárních) paprsků, které lze zařadit do jedné z následujících kategorií:

- Stínové paprsky
- Odražené paprsky
- Lomené paprsky

Stínové paprsky zjišťují přímé osvětlení objektu světlem. Tento paprsek je vyslán od průsečíku primárního paprsku s objektem směrem ke světlu a je opět testován proti objektům ve scéně. Pokud je nalezen průsečík stínového paprsku s objektem a tento průsečík je blíže než světlo, nachází se objekt ve stínu; v opačném případě jej světlo osvětluje. Touto metodou lze získat přesné a ostré stíny z bodového zdroje světla.

Odražené paprsky umožňují zobrazit zrcadlový odraz. Směr odraženého paprsku je odvozen dle pravidel optiky ze směru primárního paprsku a normály povrchu. Odražený paprsek se po vyslání chová stejně jako paprsek primární. Po případném nalezení průsečíků paprsku s tělesem jsou pro tento průsečík opět vyslány stínové, odražené a lomené paprsky. Tento proces umožňuje zobrazení vzájemného zrcadlení lesklých těles. Vysílání odražených paprsků by vlastně mohlo pokračovat do nekonečna; z tohoto důvodu je stanoven horní limit počtu odražení.

Lomené paprsky simulují průhledné objekty. Princip zobrazení je obdobný jako u odražených paprsků, pouze místo směru odraženého paprsku se dle zákonů optiky odvozuje lomený paprsek. Algoritmus sleduje cestu paprsků z pomyslného oka nebo kamery prostřednictvím každého pixelu ve virtuální obrazovce a počítá barvy objektů, které jsou viditelné.

Na rozdíl od běžného života, kdy se paprsky pohybují od zdroje, odráží se a lámou, až se nakonec střetnou s okem pozorovatele, zde paprsky vycházejí z kamery. To protože ze zdrojů světla vychází nekonečné množství paprsků a nedalo by se v rozumném čase spočítat, které dopadnou na pixely plátna, přes které se oko dívá.

Naivní algoritmus testuje navzájem každý paprsek s každým objektem scény (a se všemi polygony v každém objektu), což vede ke značně časově náročnému výpočtu. Každý průsečík paprsku s objektem generuje další dva paprsky + stínový paprsek. V každém takovém průsečíku je zapotřebí provést ty samé výpočty; je tedy vhodné využít pro implementaci ray tracingu rekurzi.

Pro ukončení rekurze je možné použít následující kritéria:

1. Paprsek narazí na difúzní povrch
2. Je dosažena předem stanovená maximální hloubka rekurze
3. Energie paprsku klesne pod určitý práh

V každém průsečíku P paprsku a objektu platí:

$$\mathrm{I}(P) = \mathrm{I}_{local}(P) + \mathrm{I}_{global}(P) = \mathrm{I}_{local}(P) + \kappa_{rg}\mathrm{I}(P_r) + \kappa_{tg}\mathrm{I}(P_t)$$

kde:

- P - průsečík
- P_r - další průsečík odraženého paprsku
- P_t - další průsečík propuštěného paprsku
- κ_{rg}- globální koeficient odrazivosti (reflexe)
- κ_{tg} - globální koeficient propouštění (lomu, transmise, refrakce)

Rekurzivní charakter této rovnice potvrzuje vhodnost tohoto přístupu pro implementaci ray tracingu.

Pro potřeby algoritmu musíme definovat:

1. 3D Scénu
   a. Objekty a jejich pozice, tvar, barva, průhlednost a další vlastnosti materiálu, popřípadě textury
   b. Světelné zdroje - opět pozice, barva, tvar
   c. Případně: barva pozadí scény, vlastnosti prostředí
2. Pozici pozorovatele
3. Rozměry zobrazovací mříže

2.2 Nevýhody ray tracingu

[31]Nevýhod je u metody ray tracing několik. Největší a hlavní nevýhodou je rychlost zpracování. Nízká rychlost zpracování obrazů scény je způsobena vysokým počtem testů na hledání průsečíků, jelikož každý vyslaný paprsek odráží (vytváří) další a další paprsky. Metoda ray tracing není adaptivní, tzn., že zobrazení probíhá se stejným vzorkováním a nezávisle na situaci ve scéně.

Jako důsledek toho, že se ve scéně vyskytují pouze bodové zdroje světla, je nevýhodou vznik ostrých stínů. Plošná světla se většinou nahrazují mnoha bodovými zdroji, což má výrazný dopad na rychlost výpočtu. Tento postup však nevede k dosažení měkkých polostínů, ale k vržení mnoha překrývajících se stínů. Čím více je světel, tím více stínů se překrývá a tím lepší je napodobení měkkého stínu.

Další problém, jenž se vyskytuje u této metody, jsou zrcadla nebo jiné lesklé plochy, které sice odrážejí okolní scénu, ale neodrážejí světlo do okolí a nestávají se tak sekundárními zdroji světla a tudíž další odlesky nevrhají.

Poslední nevýhodou, souvisejí s rychlostí zpracování, je nutnost vyhodnotit celou scénu znovu při jakékoliv změně ve scéně (nové světlo, nový objekt, odebrání objektu, atd.) nebo při změně místa pozorovatele.

2.3 Reprezentace objektů

Objekty jsou v počítačové grafice zpravidla reprezentovány buď pomocí "množiny trojúhelníčků", které se snaží vystihnout jejich tvar, nebo jsou tělesa popsána pomocí matematického zápisu. Pro metodu ray tracingu je, mimo jiné, potřeba vyřešit problém, jak získat průsečík paprsku s určitým objektem.

Přímka

Směrnicová rovnice přímky

[1]Přímku můžeme implicitně vyjádřit pomocí směrnicové rovnice ve tvaru:

$$y = kx + q$$

kde $k = \tan\varphi$ je tzv. směrnice přímky, přičemž φ je orientovaný úhel s vrcholem v průsečíku přímky a první souřadnicové osy, jehož rameny jsou (kladně orientovaná) první osa souřadnicové soustavy a přímka, a q je tzv. úsek (vyťatý přímkou) na ose y, což je druhá souřadnice průsečíku přímky s osou y.

Pro $k > 0$ představuje rovnice přímky rostoucí funkci, pro $k < 0$ jde o funkci klesající. Pro $k = 0$ je přímka rovnoběžná s osou x. Je-li $q = 0$, pak přímka prochází počátkem souřadné soustavy.

Přímku rovnoběžnou s osou y nelze směrnicovou rovnicí vyjádřit.

Parametrická rovnice přímky

Parametrické rovnice přímky v rovině lze vyjádřit vztahy:

$$x = x_0 + a_1 t$$

$$y = y_0 + a_2 t$$

kde $[x_0, y_0]$ je libovolný bod přímky, a_1, a_2 jsou konstanty určující směrnici přímky a $t \in (-\infty, \infty)$ je proměnný parametr. Alespoň jedna z konstant a_1, a_2 musí být nenulová.

Vektor

Pro vyjádření v kódu programu je však vhodnější použití vektoru. Vektorová rovnice přímky má tvar:

$$\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$$

kde **p** je polohový vektor procházející všemi body přímky, **o** je polohový vektor jednoho z bodů přímky, **d** je vektor určující směr přímky a $t \in (-\infty, \infty)$ je proměnný parametr.

Průsečík paprsku a roviny

[2]Rovinu určuje vektor bodu **p** a normála **n**. Rovnicí roviny pak je:

$$\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) = 0$$

Dosazením rovnice přímky do rovnice plochy získáme:

$$\mathbf{n} \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{p}) = 0$$

$$t = \frac{\mathbf{n} \cdot (\mathbf{p} - \mathbf{o})}{\mathbf{n} \cdot \mathbf{d}}$$

Pokud je $t \geq 0$, pak paprsek protíná rovinu a hledaný průsečík je $\mathbf{o} + t\mathbf{d}$.

Průsečík paprsku a trojúhelníku

[1]Trojúhelník je definován pomocí tří bodů a, b a c. Pokud nejsou kolineární, pak definují rovinu, kterou můžeme vyjádřit pomocí barycentrických souřadnic jako:

$$p(\alpha, \beta, \gamma) = \alpha a + \beta b + \gamma c$$

kde konstanty

$$\alpha + \beta + \gamma = 1.$$

Barycentrické souřadnice jsou definovány pro všechny body v rovině, bod p leží v trojúhelníku pouze pokud platí, že

$$0 < \alpha < 1,$$

$$0 < \beta < 1,$$

$$0 < \gamma < 1$$

Pokud je jedna souřadnice nulová, pak se bod nachází na hraně trojúhelníku. Pokud jsou nulové dvě souřadnice, pak leží v jeho vrcholu.

Kombinací obou rovnic dostáváme rovnici:

$$p(\beta, \gamma) = a + \beta(b - a) + \gamma(c - a)$$

kde platí, že

$$\beta + \gamma < 1,$$
$$0 < \beta,$$
$$0 < \gamma.$$

Průsečík paprsku $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$ a roviny má tvar:

$$\mathbf{o} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}).$$

Ke zjištění t, β a γ rozepíšeme rovnici na soustavu rovnic pomocí vektorů:

$$o_x + td_x = a_x + \beta(b_x - a_x) + \gamma(c_x - a_x),$$
$$o_y + td_y = a_y + \beta(b_y - a_y) + \gamma(c_y - a_y),$$
$$o_z + td_z = a_z + \beta(b_z - a_z) + \gamma(c_z - a_z).$$

Normálový vektor získáme pomocí vektorového součinu dvou nerovnoběžných vektorů z roviny trojúhelníku:

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})$$

Průsečík paprsku a koule

[1]Kouli se středem $\mathbf{c} = c_x, c_y, c_z)$ a poloměrem R můžeme definovat jako množinu bodů (x, y, z), které splňují rovnici:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 + R^2 = 0$$

ve vektorovém vyjádření pak:

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - R^2 = 0.$$

Pokud do rovnice za $\mathbf{p}$ dosadíme rovnici přímky:

$$\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$$

Dostaneme:

$$(\mathbf{o} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{c}) - R^2 = 0.$$

Po úpravě této rovnice dostaneme kvadratickou rovnici:

$$(\mathbf{d} \cdot \mathbf{d})t^2 + 2\mathbf{d} \cdot (\mathbf{o} - \mathbf{c})t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2 = 0$$

Řešení této rovnice pak je:

$$t = \frac{-\mathbf{d} \cdot (\mathbf{o} - \mathbf{c}) \pm \sqrt{(\mathbf{d} \cdot (\mathbf{o} - \mathbf{c}))^2 - (\mathbf{d} \cdot \mathbf{d})((\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2}}{\mathbf{d} \cdot \mathbf{d}}$$

Při hledání řešení rovnice nám pomáhá diskriminant rovnice D.

Pokud je:

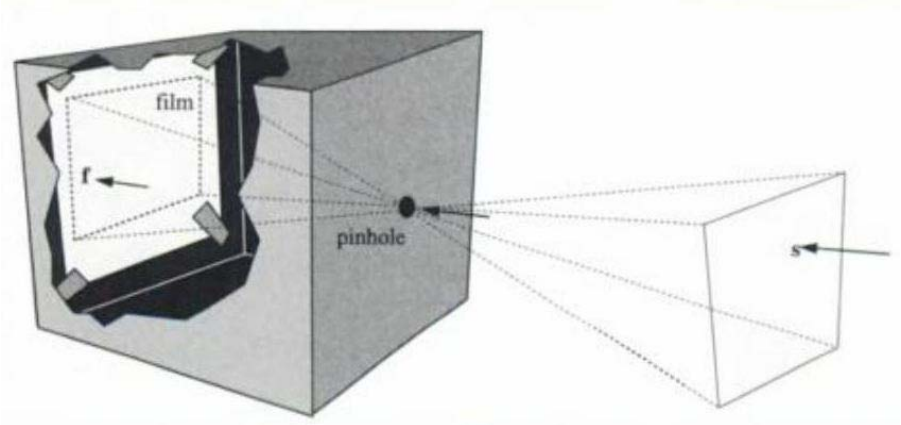
- $D < 0$, pak paprsek s koulí nemá žádný průsečík, tj. pokud je vzdálenost přímky od středu koule větší než poloměr kružnice
- $D > 0$, pak získáme dva průsečíky, jeden při vstupu paprsku do koule a druhý při jeho opuštění. Paprsek je tedy sečnou kulové plochy, tj. pokud je vzdálenost přímky od středu koule menší než poloměr kružnice.
- $D = 0$, pak se paprsek dotkl koule v jednom bodě. Paprsek je tečnou kulové plochy, tj. pokud je vzdálenost přímky od středu koule rovna poloměru kružnice

Normálový vektor $\mathbf{n}$ v bodě $\mathbf{p}$ je $\mathbf{n} = 2(\mathbf{p} - \mathbf{c})$. Jednotkový normálový vektor pak $(\mathbf{p} - \mathbf{c})/R$.

2.4 Model Kamery

Abychom byli schopni zaznamenat (nasnímat) obraz okolní scény, definujeme objekt kamery. S tím však vyvstává několik nutností, které je třeba řešit. Prostor, který se vykresluje na obrazovku, je dvourozměrný, avšak okolní scéna je definována v trojrozměrném prostoru. Tato transformace trojrozměrné scény do dvourozměrného obrazu se nazývá promítání. Průmětnou pak můžeme označit plochu, na kterou se obraz promítá.

[1] [2] [21] Kamera definuje polohu pozorovatele (kamery), směr pohledu a šíři záběru. Polohou pozorovatele uvažujeme bod, který je společným počátkem paprsků vysílaných do scény. Kamera je v podstatě obdobou štěrbinového fotoaparátu, pouze projekční plocha se zde nenachází za štěrbinou, ale před ní; tím výsledný obraz není převrácen.



Obrázek 4: Reálný štěrbinový fotoaparát [1]

Kamera je tedy objektem, definujícím pozici a orientaci pozorovatele ve scéně a směr, kterým se pozorovatel dívá, a tuto definici můžeme popsat tzv. pohledovým paprskem, jehož výchozí bod P_k představuje počátek souřadného systému kamery a směrový vektor $\vec{s}_k$ udává směr pohledu.

Výchozím bodem každého paprsku bude pozice příslušného pixelu v rámci trojrozměrné scény. Směrový vektor paprsku zjistíme na základě zvoleného modelu kamery. Při hledání tohoto vektoru budeme, na základě definice modelu kamery, hledat funkci φ, která danému pixelu na souřadnicích (i, j) přiřadí směrový vektor paprsku, jež z daného pixelu vychází.

Danou definici zapíšeme, jako vztah:

$$\varphi: O \rightarrow V$$
$$\vec{s}_{ij} = \varphi(i, j)$$

kde O je prostor obrazovky a V představuje vektorový prostor vztažený k souřadnicovému systému kamery.

2.5 Lokální osvětlovací model

Osvětlovacím modelem v kontextu počítačové grafiky rozumíme modely, které popisují intenzitu světla v bodech scény. Hlavními faktory jsou světelné zdroje, vlastnosti objektů ve scéně a samozřejmě také vlastnosti pozorovatele. Slovo lokální označuje fakt, že se vypočítává osvětlení jediného bodu na povrchu tělesa.

Dopadne-li světelný paprsek do bodu na povrchu tělesa, po odrazu se rozptýlí do všech směrů. [2] Funkce, počítající barvu osvětleného bodu, se nazývá osvětlovací model (stínovací funkce). Tato matematická funkce vyjadřuje intenzitu paprsku rozptýleného světla v závislosti na jeho směru a na směru, intenzitě a vlnové délce dopadajícího paprsku. Pomocí tohoto modelu můžeme vyjádřit vlastnosti jako barva, lesklost, drsnost, matnost atd.

Nejznámějším systémem s propracovanými stínovacími funkcemi (nazývanými shadery) je RenderMan od společnosti Pixar Animation Studios [22]. Primárně používá algoritmus Reyes, ale je také plně schopen používat metody sledování paprsku a globálního osvětlení. Tento software byl původně vyvinut pro vytváření filmů samotnou společností Pixar, nyní je však také k dispozici jako komerční produkt.

Phongův osvětlovací model

[15][16] Tato metoda byla vytvořena Bui Tuong Phong na University of Utah, kterou publikoval ve své disertační práci z 1973 dva roky před svou smrtí.

Původně byl Phongův osvětlovací model určen pro monochromatické světlo, ale jeho vzorce se dají použít i pro světlo barevné. Tento model patří mezi empirické osvětlovací modely, nemající

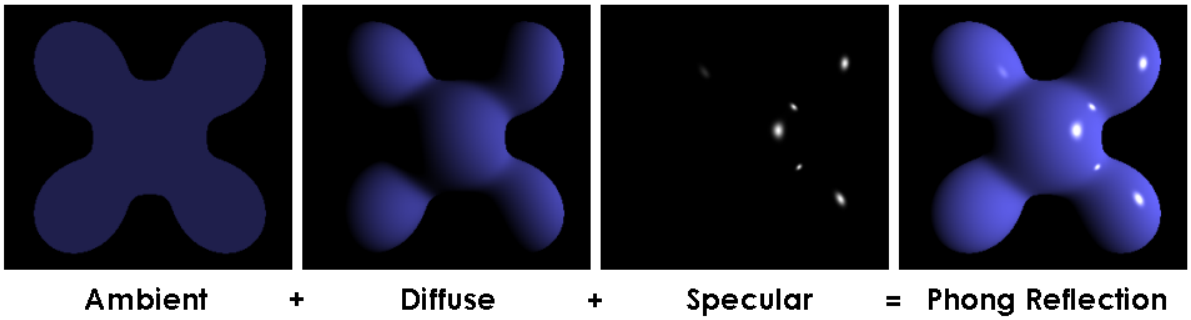
přímý vztah s fyzikální podstatou šíření a odrážení světla, ale jsou založeny na zkušenostech a pozorování.

Phongův model určuje množství světla v daném bodě podle směru dopadajícího paprsku, povrchové normály v daném bodě a směru ke zdroji světla. Model popisuje tři složky - okolní světlo (ambientní, které je konstantní), rozptýlené světlo (difúzní; učené dle normály povrchu) a odražené světlo (spekulární, tj. odraz světelného zdroje). Funkce je vypočítána pro všechny světelné zdroje a jednotlivé příspěvky jsou posléze sečteny.

Ambientní složka vzniká odrazem "ambientního" světla, což je světlo přicházející ze všech směrů se stejnou intenzitou. Samotné ambientní světlo nám, vzhledem ke své povaze, plastické zobrazení objektů ve scéně nezajistí, ale je užitečné ve spojení s dalšími složkami.

Difúzní složka světla představuje odraz světla pocházejícího z určitého daného zdroje nebo alespoň směru. Od povrchu tělesa se odráží do všech směrů stejně. Sama o sobě difúzní složka stačí k zobrazení matných objektů.

Spekulární složka je poslední ze složek odraženého světla, ta vytváří tzv. odlesky. Odlesky vznikají ze světla, které se odráží s jistým rozptylem podle zákona odrazu.



Obrázek 5: Složky světla [32]

Phongův osvětlovací model pro jeden světelný zdroj vyjadřuje rovnice:

$$I_p = \kappa_a i_a + \kappa_d i_d (\mathbf{L} \cdot \mathbf{N}) + \kappa_s i_s (\mathbf{R} \cdot \mathbf{V})^{\alpha}$$

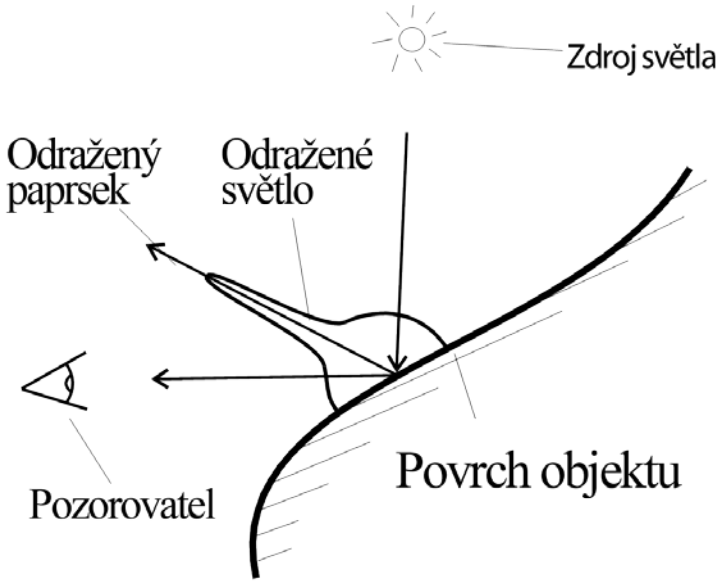
kde konstanty $\kappa_a, \kappa_d, \kappa_s$ se vztahují k materiálu objektu a konstanty i_a, i_d, i_s ke světelnému zdroji. Konstanta α pak vyjadřuje lesklost materiálu. Vektor **L** udává směr ke zdroji světla, **N** je normálový vektor v daném bodě povrchu, **V** je vektor směru k pozorovateli (obrácený směr příchozího paprsku) a **R** je směr dokonale odraženého paprsku ze světelného zdroje.

Odraz

Odražený paprsek se nachází vždy ve stejné rovině s paprskem dopadajícím a s normálou povrchu svírá úhel o stejné velikosti, ale opačném znaménku. Odražený paprsek můžeme vyjádřit rovnicí:

$$\mathbf{R} = \mathbf{I} - 2\mathbf{N}(\mathbf{I} \cdot \mathbf{N})$$

kde je **I** příchozí paprsek a N normála povrchu.



Obrázek 6: Osvětlení [17]

Lom

Pomocí metody sledování paprsku je lom světla další z efektů, který je snadno dosažitelný. Paprsek se při přechodu mezi dvěma prostředími láme úměrně k poměru indexů lomu těchto prostředí. Toto vyjadřuje Snellův zákon[12]:

$$\eta_1 \sin(\theta_1) = \eta_2 \sin(\theta_2)$$

Ve vektorové formě pak:

$$\mathbf{T} = \frac{\eta_1}{\eta_2}\mathbf{I} + \left(\frac{\eta_1}{\eta_2}\cos\theta_1 - \cos\theta_2\right)\mathbf{N}$$

Vektor **I** je příchozí paprsek a N normála povrchu. [18].
Nutné je rozeznat přechod dovnitř objektu a ven z objektu kvůli správným indexům lomu. Index lomu okolí lze předpokládat roven jedné.

Světlo, procházející průhledným objektem, obyčejně ztrácí svou intenzitu. Toto chování popisuje Lambertův-Beerův zákon [19], který říká: prochází-li světelný paprsek prostředím, které je

schopno absorbovat, je intenzita paprsku vstupujícího vyšší než intenzita paprsku prošlého tímto prostředím. Tento jev byl v roce 1729 poprvé formulován P. Bouguerem a později ještě jednou objeven Lambertem. Lze jej vyjádřit rovnicí:

$$I = I_0 \cdot e^{-bd}$$

kde:

I_0 - intenzita vstupujícího paprsku

I - intenzita paprsku po průchodu absorbujícím prostředím

d - vzdálenost od místa, kde paprsek vstupuje do absorbujícího prostředí

b - absorpční (napierovský) koeficient

2.6 Textury

Textury popisují vzhled materiálu, ze kterého je daný objekt vyroben, či kresbu nebo tapetu na povrchu objektu. Také je lze využít jako náhradu příliš složité geometrie objektu. Používají se trojrozměrné (3D) a dvourozměrné (2D) textury. 3D textura je vhodná k vyjádření materiálu, z něhož je objekt například vyřezán. 2D textury slouží k zobrazení obrázku na povrchu tělesa.

Z hlediska způsobu vytváření dělíme textury do dvou kategorií – rastrová a procedurální. U rastrových je texturou předem připravený rastrový obrázek. Pro dobrý vzhled výsledné scény je důležitá dostatečná detailnost textury. Často používanými formáty pro jejich ukládání jsou BMP, TGA a DDS (DirectDraw Surface).

Procedurální textury jsou vyjádřeny pomocí nějaké matematické funkce. Jejich výhodou je, že nezáleží na rozlišení, ale se přizpůsobí velikosti renderovaného obrazu. Nevýhodou však je, že ne všechny povrchy lze matematicky vyjádřit.

Obrázek 7: Textury [32]

2.7 Path Tracing

[28]Path tracing je renderovací technika, která podobně jako ray tracing, sleduje cestu paprsku od pozorovatele směrem ke scéně. Na rozdíl od klasického ray tracingu se na detekovaných průsečících paprsky dále nevětví. Podle [28] je tento postup na rozdíl od ray tracingu efektivnější

v oblasti paměťové náročnosti, což je při renderovaní v reálném čase důležitým faktorem. V klasickém ray tracingu je v každém průsečíku spočítaný osvětlovací model na základě rozmístění světel a vlastností povrchu. Při detekování průsečíku s objektem může nastat lom resp. odraz daného paprsku a následně jsou vygenerovány dva paprsky s odpovídajícím směrem na základě indexu lomu respektive reflexivních vlastností povrchu. Tento postup vede na větvení v každém paprsku a je z hlediska složitých scén neefektivní. Alternativu v tomto směru poskytuje path tracing. Algoritmus je založený na generování paprsku směrem od pozorovatele každým pixelem obrazu; tento krok vytvoří primární paprsky. Při detekci průsečíku paprsku s objektem je nutné stanovit událost, která vygeneruje sekundární paprsek.

2.8 Photon Maping

[30] Photon mapping je alternativou pro výpočet globálního osvětlení scény. Jde o dvouprůchodový algoritmus, jehož první část simuluje tok fotonů scénou a jejich interakce s povrchem objektů scény (nazývaná shooting) a druhá část, která ze zaznamenaných drah fotonů odhaduje výslednou intenzitu osvětlení jednotlivých viditelných bodů scény. Umožňuje simulaci jevů jako je kaustika (Caustics). Její vlastností je rychlost metody, její snadná paralelizace a nezávislost na geometrii scény. Pomocí této metody lze simulovat všechny druhy přímého a nepřímého osvětlení, difuzních a dokonale i nedokonale zrcadlových odrazů.

2.9 Aliasing

Při převodu spojité informace na diskrétní (nespojitou) – objekty byly popsány matematicky a vygenerovaný obraz je uložen v diskrétní podobě – vzniká aliasing. Takový převod se nazývá vzorkování, a aby nedocházelo k aliasingu, musí být vzorkovací frekvence větší než dvojnásobek nejvyšší frekvence obsažené ve vzorkovaném signálu - tzv. Shannonův teorém (Nyquistův nebo Shannon-Nyquistův teorém). Pokud tuto podmínku nesplňuje, dochází k překrytí frekvenčních spekter vzorkovaného signálu a tedy ke ztrátě informace. Při nedodržení Shannonova teorému je původní frekvence spojité informace vzorkováním zfalšována. Ukázku aliasingu si můžeme demonstrovat na filmovém záznamu rychle se otáčejícího předmětu, kterým je kupříkladu vrtule letadla. Obvykle divák, při sledování filmu, vidí vrtuli točící se nepřirozeně pomalu a někdy dokonce i opačným směrem oproti skutečnosti.

Aliasingu je nutné předcházet, protože pokud k němu dojde, jeho následky se odstraňují jen velmi těžce. Proto se před převodník spojitého signálu na diskrétní ve většině případů zařazuje tzv. antialiasingový filtr, který má za úkol odfiltrovat frekvence vyšší než odpovídají Shannonovu teorému.

Jako metodu pro anti-aliasing v ray tracingu můžeme použít metodu adaptivního nadvzorkování (adaptive supersamplingu), což je opačná metoda k metodě adaptivního podvzorkování (adaptive subsampling), která je popsána níže. Při použití této metody vyšleme každým pixelem 5 paprsků, 1 středem a 4 jeho rohy. Jestliže se výsledné barvy paprsků příliš neliší, pixel obarvíme jejich průměrem; v opačném případě rozdělíme pixel do 4 subpixelů a na každý opakujeme stejný postup. Tento algoritmus je poměrně snadný a dobře odstraní aliasing. Avšak zásah tělesa s malým průmětem není jistý ani při 5 vyslaných paprscích (těleso mineme a stále všechny paprsky budou mít podobnou barvu). Proto je lepší provádět nadvzorkování nepravidelně.

Při stochastickém vzorkování se výběr vzorků provádí buď zcela náhodně, nebo v každém pixelu stejný počet vzorků, náhodně posunutých v rámci pixelu vůči pravidelné mřížce - tzv. roztřesení (jittering). Takto efektivně odstraníme aliasing, přesněji řečeno aliasing stále existuje, avšak ve formě šumu, k němuž je lidské vnímání lhostejné.

2.10 Distribuovaný ray tracing

Nadvzorkování lze využít i v průběhu rekurzivního sledování. Využívá více paprsků, ze kterých pomocí distribuční funkce (fce rozložení) vypočte výslednou hodnotu. Generuje svazek primárních, odražených, stínových a lomených paprsků kolem původního přesného paprsku. Jejich směry mají náhodné rozdělení dané distribuční funkcí, která je specifická pro daný povrch a tím stochasticky (metodou Monte Carlo) aproximujeme integrál celkové intenzity světla I odrážející se od povrchu pod prostorovým úhlem (ϕ_r, θ_r)

$$I(\phi_r, \theta_r) = \int_{\phi_i} \int_{\theta_i} L(\phi_i, \theta_i) R(\phi_i, \theta_i, \phi_r, \theta_r)\, d\phi_i d\theta_i$$

kde R je funkce odrazivosti a L je funkce osvětlení (dopadající intenzity světla v závislosti na úhlu dopadu). Využívá se pro vznik měkkých stínů, odrazů na matných plochách, refrakce světla, motion blur (více paprsků v časovém intervalu) a supersampling AA.

2.11 Monte Carlo

Monte Carlo je metoda numerické integrace funkce. Pro vyhodnocení integrálu funkcí je možno použít i symbolické integrovaní nebo některou z numerických např. obdélníkovou metodu. V počítačové grafice musíme při řešení globálního osvětlení vyhodnocovat vícerozměrné integrály. Nevýhodou použití obdélníkové metody je v tomto případě potřeba vyhodnotit N^d funkčních hodnot pro d-rozměrný definiční obor. V tomto směru poskytuje metoda Monte Carlo univerzální nástroj pro numerické integrování více-dimenzionálních funkcí.

3 Optimalizace výpočtů v grafice

V této kapitole budeme diskutovat současný stav poznatků týkajících se zrychlení výpočtů ray tracingu. Zaměříme se na používané datové struktury a výpočetní náročnost vyhledávání průsečíků s tělesy. Na závěr si v krátkosti si uvedeme poznatky o paralelizaci výpočtů v souvislosti s ray tracingem a metodami výpočtu globálního osvětlení.

Jak již bylo popsáno, každý vyslaný paprsek je testován na průsečík se všemi objekty ve scéně a navíc při odrazu a lomu vznikají další paprsky, které je nutné taktéž testovat. Se zvyšující se složitostí scény tak značně stoupá časová náročnost, a proto je algoritmus nutné optimalizovat.

[3][6] V průběhu let bylo vymyšleno mnoho metod jak zobrazování pomocí metody sledování paprsku urychlit. Metody, kterými lze dobu vypočtu snížit, můžeme rozdělit do těchto možností:

- vysílat méně paprsků
- počítat méně průsečíků
- počítat průsečíky rychleji

Při generování obrazu pomocí metody sledování paprsku je největší část doby výpočtu strávena hledáním průsečíků paprsku a geometrických objektů. Existují dvě metody optimalizace tohoto problému:

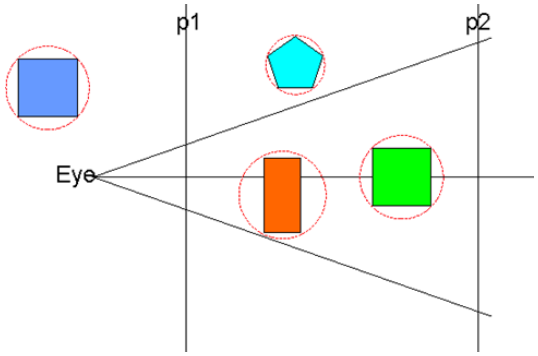
1. První metoda řeší přímo otázku vzájemné polohy konkrétních typů geometrických objektů s paprskem a hledá vhodnější algoritmy.
2. Druhá metoda spočívá v rozdělení prostoru do struktur, které umožňují zmenšení počtu objektů, které je nutné otestovat algoritmy na střetnutí s paprskem

3.1 Bounding volumes

Do češtiny bychom tuto metodu mohli přeložit jako obalová tělesa a zařadit ji mezi metody zrychlující výpočet průsečíku. Přirozený algoritmus testuje všechny tělesa a jejich průsečík s paprskem. Tento průnik však u některých těles může být velmi výpočetně náročný, zvláště pro některé tvary těles (NURBs [23] a další tělesa definované pomocí křivek, polygony s mnoha hranami a další). Proto je vhodné pro tyto tělesa definovat a uzavřít je do tzv. obalových těles s jednoduchým výpočtem průsečíku paprsku. Prakticky obalové těleso objektu o je buňka ν pro kterou platí, že $o \cap \nu \equiv o$. Pokud je průnik mezi paprskem a obalovým tělesem, pak může nastat i průnik s objektem. Pokud však paprsek obalové těleso neprotne, neprotne ani obalovaný objekt a tak v průměru nastane značné ušetření složitých výpočtů průsečíků.

Jednoduchými obalovými tělesy může být:

- koule, která má velmi jednoduchý algoritmus pro zjištění průsečíku [2].
- elipsoid – většinou těsněji obepíná obalované těleso
- krychle
- kvádr
- válec
- a další[24]

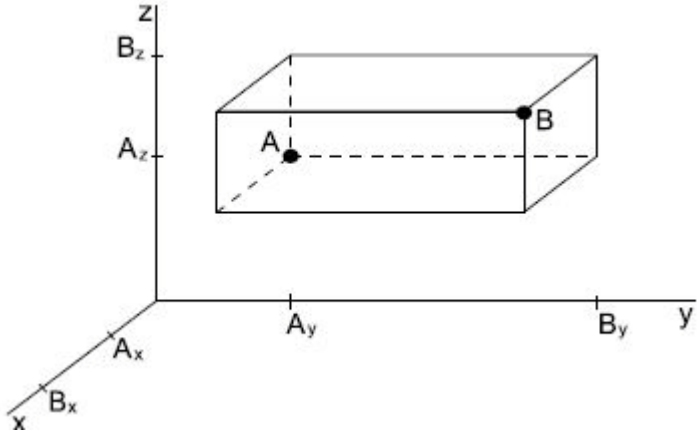


Obrázek 8: Obalová tělesa

Požadavky kladené na obalová tělesa jsou následující:

- *Těsnost*: protne-li paprsek obalové těleso, pak pravděpodobnost, že paprsek protíná také obalovaný objekt, je vysoká.
- *Účinnost*: časová složitosti testu na průnik paprsků a obalové těleso by měla být co nejmenší.

Aplikaci této metody můžeme naleznout v jiných optimalizačních metodách např. v metodě hierarchie obalových těles nebo v metodách dělení prostoru.



Obrázek 9: Axis -Aligned Bounding Box

Axis - Aligned Bounding Box

Další možností je použít Axis-Aligned Bounding Box (AABB). AABB Je obalové těleso tvaru kvádru nebo krychle, jehož strany mají normály, které jsou shodné s osami souřadnicového systému. AABB se popisuje dvěma extrémními body a_{min} a a_{max}, kde $a_{min} \leq a_{max}$, pro každé $i = \{x, y, z\}$.

3.2 Bounding volume hierarchies

Též hierarchie obalových těles (ang. Bounding volume hierarchies, ve zkratce BVH), navržena Rubinem a Whittedem [11] a pány Kay a Kajiya [10], je přirozeným rozšířením metody obalových těles, který využívá hierarchické koherentnosti. Udržuje obalová tělesa objektů ve stromu s n-árními kořeny. Každý vnitřní uzel v z BVH, který je také obalové těleso, pak obaluje další obalová tělesa, která jsou podstromem na v.

Hierarchie zajišťuje přirozené testování na průnik paprsku s objektem ve scéně. Pokud se paprsek nestřetne s uzavírajícím obalovým tělesem v kořenu stromu, nemůže neprotnout ani žádný objekt v jeho podstromu. V opačném případě se rekurzivně zanoří a testuje jen pro ty uzly BVH, jejichž obalová tělesa byla protnuta paprskem. Zajímavá vlastnost BVH je, že v každém uzlu stromu je uložena obálka obklopující všechny obálky a objekty svých potomků. Obálky potomků jsou tedy vždy obsaženy kompletně v obálce rodiče, mohou se však navzájem protínat. Metoda pro automatickou konstrukci BVH byla poprvé popsána Goldsmithem a Salmonem [9].

Pro algoritmus sestavení stromu BVH můžeme použít metodu, kde sestavení probíhá směrem zdola-nahoru nebo shora-dolů. Pokud jde o metodu sestavení shora-dolů je výrazně implementačně jednodušší, ale v některých případech sestavená struktura není optimální. Tento přístup v prvním kroku vytvoří obálku celé scény, ta je následně rozdělena při dodržení principů, které jsou definovány v [26] na 2 podstromy. Takto rekurzivně postupujeme, dokud nevyhovíme určené podmínce pro ukončení algoritmu. Algoritmus zdola-nahoru má na svém vstupu všechny data, nad kterými provede testování a podle podmínky ukončení algoritmu vytvoří listy, které následně rekurzívně zapouzdřuje do obálek vyšší úrovně, dokud se nedosáhne kořen struktury; pak se algoritmus ukončí. Metoda bottom-top vede v mnohých případech na vyváženější strukturu a v kontextu ray-tracingu nevznikají ve scéně tzv. hotspots, tzn. místa s výrazně vyššími částmi renderování, proto je vhodná na předzpracování ve statických scénách.

Při volení parametrů této struktury se musíme zaměřit na vícero kritérií, podle kterých sestavíme daný strom, a to na základě paměťových omezení a v případě ray tracingu také tvaru obalových těles. Výběr tvaru obálky je pro optimální řešení důležitou součástí požadavků, jako je minimalizování velikosti dat popisujících tvar z hlediska paměťové náročnosti, rychlost testovaní průsečíku s paprskem a rychlost určení velikosti a polohy při sestavovaní.

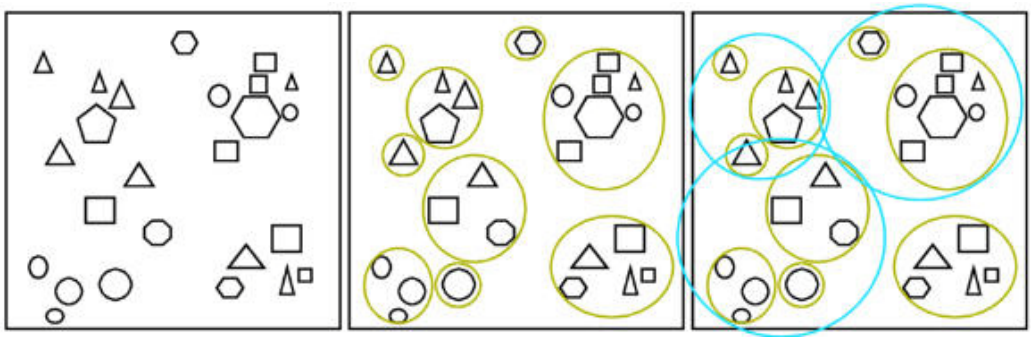
Podle [27] experimenty na typických scénách potvrdili, že ideální reprezentací obálky je minimální osově rovnoběžný kvádr. Optimálnost použití konkrétního útvaru je závislá na vstupních datech. Sestavení struktury podle [26] podléhá více kriteriím. Důležitým faktorem je celkový objem obálek, který by měl být minimální spolu s jejich počtem. V dynamických scénách je nutné zabezpečit rychlé přebudování struktury v čase zanedbatelném v porovnání s vyhledáváním.

Při budování struktury je důležité určit počet potomků nelistového uzlu. Různé experimenty [26] naznačují, že klasický model binárního stromu je jednak implementačně nenáročný a i ve složitých scénách přináší relativně uspokojivé výsledky. Navíc tento přístup je možné kombinovat s lineárním seznamem, který nám prakticky poskytne téměř okamžité vyhledání příslušného listu.

Vyhledání listu ve struktuře probíhá prohledáváním od kořene stromu, tím tedy můžeme eliminovat paprsky, které směřují mimo zvolená obalová tělesa. Podle implementace [26] je možno při použití souřadně osově zarovnaného kvádru dosáhnout v nejhorším případě časovou složitost:

$$O(\log^2 N)$$

kde N je počet primitiv v daném listě. V praxi je však takovýto scénář nedosažitelný. Důvodem je zjednodušení obálky objektu, detekce intersekce může nastat i v případě, že samotný paprsek obsaženou geometrii v listě mine. Důležitá je však schopnost struktury dosahovat vysoký prohledávací výkon i při dynamických scénách spolu s implementováním sledovaní svazku paprsků. V praxi také není možné předcházet překrytí jednotlivých obalových těles, avšak zrychlení hledání průsečíků je zanedbatelné oproti zpomalení vlivem redundantního prohledávání stromu.



Obrázek 10: Hierarchie obalových těles

3.3 Spacial Subdivisions

Další z možností optimalizace je rozdělit prostor scény tak, aby se při zobrazování minimalizovalo množství počítaných průsečíků s objekty. Na toto téma byla vyzkoušena řada algoritmů dělících prostor na vhodně uspořádané oblasti, jejichž cílem je efektivní nalezení objektů protínajících dráhu paprsku. V současné době tedy existuje velké množství akceleračních struktur, vzájemně se od sebe lišících rychlostí stavby, procházením, způsobem dělení a dalšími parametry. Z takových prostorových struktur se nejvíce prosazují kd-stromy, které jsou dostatečně jednoduché,

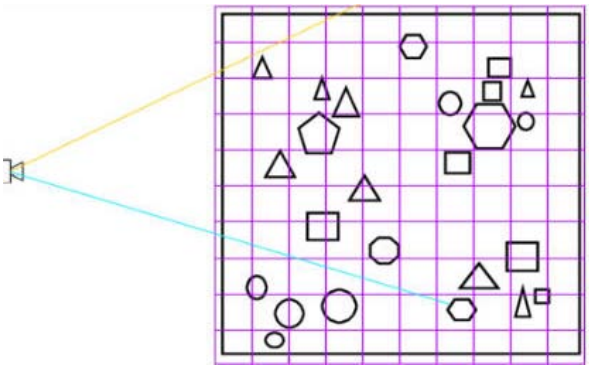
a přesto dosahují velmi dobrých výsledků na obecných scénách. Algoritmy dělení prostoru také zajišťují, že nejsou zbytečně testovány neviditelné objekty, tj. které jsou mimo záběr kamery, tím obyčejně není třeba žádné ořezávání scény ani LOD (level of detail).

Mezi metody by se dala zahrnout i metoda hierarchie obalových těles, která prostor určitým způsobem rozděluje. BHV jsme si však uvedli zvlášť, jelikož řeší dva problémy, a to, že se počítá méně průsečíků a počítání průsečíků u složitých těles se provádí rychleji. Hierarchie obálek (BVH) má, oproti kd-stromům (a obecně prostorově orientovaným hierarchickým strukturám), tu výhodu, že těsněji obklopuje jednotlivá primitiva.

Regular grid

Regular grid (pravidelná mřížka) byla poprvé presentována A. Fujimoto v práci[7] z roku 1986. Jedná se o datovou strukturu dělící prostor do pravidelných stejně velkých buněk. Výhodou tohoto přístupu je jednoduchost, rychlost stavby a jednoduchost procházení paprsku. Proto se mřížka používá pro dynamické scény v případě, že ostatní struktury neumožňují dostatečnou rychlou stavbu nebo úpravu. Hlavní nevýhodou je nemožnost měnit strukturu mřížky podle rozložení objektů ve scéně nebo nepřizpůsobení se geometrii scény, avšak počet buněk měnit lze. Může proto docházet ke vzniku zbytečných prázdných buněk, které neodkazují na žádná tělesa, a naopak ke vzniku buněk, které obsahují značné množství objektů a jejich další rozdělení by bylo vhodné.

Metoda používá rychlý 3DDDA algoritmus, což znamená, že paprsek může být vržen na mřížku a stačí testovat objekty obsažené pouze v tomto kvádru (voxel).



Obrázek 11: Pravidelná mřížka

Na obrázku 9 je znázorněna scéna, která je rozdělena pomocí pravidelné mřížky. Z kamery vycházejí dva paprsky, žlutý a modrý. Paprsky procházejí mřížku a hledají objekty, se kterými se střetnou. Jeden testuje, zda jsou v buňce nějaké objekty, druhý, které objekty v buňce jsou zasaženy. Tento přístup má určité nevýhody, jelikož procházení všech buněk a testování jejich průniku sice není výpočetně náročné, přesto však určitý výpočetní čas zabere. Další nevýhodou zmiňovanou výše je, že některé buňky mohou být prázdné, zatímco druhé mohou mít mnoho primitiv. V tomto případě nepomůže ani změna velikosti jednotlivých buněk.

Octree

Oktalový strom je stromová struktura, ve které každý uzel obsahuje osm poduzlů, není-li terminální čili koncový. Z hierarchického hlediska jsou jednotlivé oktanty reprezentované stromovou strukturou, jež každý podprostor je reprezentovaný uzlem stromu a elementární podprostor listem stromu. Jinými slovy uzel představuje krychlovou oblast prostoru, která je rozdělena na osm stejně velkých podkrychlí. Struktura octree vznikla rozšířením quadtree přidáním třetího rozměru. Algoritmus pro sestavení rozděluje prostor v každém kroku pomocí 2 hyperploch na 8 shodných podprostorů a právě díky pevně danému umístění hyperploch je sestavení takové struktury poměrně rychlé. Horní odhad časové složitosti vyhledávání konkrétního listu je podle [29]:

$$O(\log_8 N)$$

Podle [29] je možno strukturu linearizovat a dosáhnout tak časové složitosti vyhledaní uzlu na úroveň:

$$O(\log_2 N)$$

Metoda Octree není vhodná pro scény s nerovnoměrným rozložením objektů; vznikají prázdné listy, což v konečném důsledku prodlužuje čas vyhledání daného objektu.

KD - tree

Kd-strom je speciální variantou binárního dělení prostoru, která dělí prostor na dvě poloviny pomocí obecné roviny. Tato metoda je speciální případ BSP-tree, které detailněji popisuje další kapitola. Kd-tree, podobně jako octree, využívá při dělení hyperplochy, avšak prostor rozděluje na různě velké podprostory. Možnost orientace dělící roviny je omezena podmínkou, že dělící rovina musí být vždy kolmá na některou z prostorových os. Dále se liší v umístění dělící roviny. V případě BSP stromu je umístěna vždy doprostřed děleného voxelu, při použití Kd-tree je však pozice dělící roviny libovolná.

Vytvoření Kd-tree je, na rozdíl od octree, netriviální problém. Umístění hyperplochy je v tomto případě dosti variabilní vzhledem na podprostor a jeho rozměry. Možností, kam umístit dělící plochu, je nekonečně mnoho. Vždy se však hledá vhodná pozice, a to taková**,** která vytvoří co největší prázdný prostor. Tím se značně sníží počet testů průsečíků s objekty. Způsob stavby stromu výrazně ovlivňuje rychlost procházení paprsku tímto stromem. Existují různé způsoby určení osy dělící roviny a její polohy. Nejjednodušším z nich je varianta, při níž se pravidelně střídají osy použité pro výběr dělící roviny. Rovina je umístěna tak, aby buď rozdělila prostor na dvě stejně velké oblasti, nebo aby oblasti na obou stranách roviny obsahovaly stejný počet objektů. Tento postup však není vhodný pro stavbu stromu za účelem sledování paprsku. Pro účely určení polohy hyperplochy bylo vyvinuto vícero heuristik. Použití konkrétní heuristiky závisí na účelu struktury, která bude vybudována. Triviální určení polohy pro umístění hyperplochy je vyvážený počet primitiv v obou vzniklých podprostorech. Předpoklad, že takovéto rozmístění je optimální, vyvrací [13]. Opírá se o předpoklad,

že při hledání průsečíků v podprostoru nemá počet obsažených primitiv takovou důležitost, jako jejich celková plocha. Proto se téměř výlučně používají alternativy SAH (Surface area heuristics), která je založena na geometrické pravděpodobnosti a kde pro umístění hyperplochy je do výpočtu zahrnutá plocha jednotlivých primitiv. Heuristika vychází z předpokladu, že při použití svazku paprsků stoupá pravděpodobnost detekce průsečíku s vážnou plochou polygonu a počet testů je v rámci daného listu kd-stromu ve většině případů minimalizovaný. Metoda SAH je podrobně popsána v [25] [13]. Vyhledání optimální hyperplochy se však podle [25] asymptoticky přibližuje k:

$$O(N^2)$$

při naivním řešení, kde N je počet primitiv v daném prostoru. Pokud uvažujeme statické scény, je čas vybudovaní struktury míň důležitý jako rychlost a optimálnost obsahu jednotlivých listů stromu. Při použití modifikované SAH heuristiky je možné optimalizovat na:

$$O(N \log N)$$

Strukturu je možno implementovat jako binární strom, kde každý uzel, který není koncovým uzlem, obsahuje podstrom podprostoru daný hyperplochou rodiče, který tento uzel definuje.

Z hlediska povahy zpracovávaných dat trpí Kd-tree nedostatky při velkém počtu primitiv rovnoběžných se souřadnicovým systémem. Vznikají zde prázdné podprostory a naopak podprostory těsně obklopující geometrii.

BSP tree

[3] BSP (Binary space partitioning) je metoda, která rekurzivně dělí prostor na dva podprostory pomocí dělící roviny. Výsledkem tohoto dělení je datová struktura známá jako BSP strom. Tato metoda používá, v porovnání s Kd-tree, všeobecné hyperplochy a mohlo by se zdát, že díky tomu můžeme tuto metodu označit za optimální pro akcelerování ray-tracingu. Toto tvrzení však vyvrací [25]. Použití obecných hyperploch řeší problémy přetáhnutých polygonů a architektonických scén; na druhé straně je však při určení dané hypeplochy potřebné definovat 3 body ležící v této ploše, což může být časově náročné. Následné rozdělení podprostoru trpí chybami zaokrouhlení. Tyto mohou vést k zobrazovacím chybám resp. ke zvýšení výpočetní náročnosti specifických částí scény.

Konstrukce stromu probíhá rekurzivně; vstupem je osově zarovnaný kvádr (voxel) ohraničující scénu. Při každém kroku rekurze je rozdělen na polovinu podle nejdelší osy. Následně je provedeno rozdělení objektů podle toho, do které poloviny náleží. Pokud objekt protíná dělící rovina, je přiřazen do obou polovin. Rekurzivní dělení je prováděno do té doby, než je splněno jedno ze dvou ukončujících kritérií. Prvním je minimální počet objektů, druhým maximální hodnota rekurze.

Výsledný strom obsahuje dva typy uzlů:

- *Vnitřní uzel*: obsahuje rozměr dělení (X, Y, Z) a pozici dělící roviny
- *List*: obsahuje seznam objektů scény, které leží v daném podprostoru

Procházení struktury stromu při vykreslování scény je řešeno algoritmem TA^{B}_{rec}, který je podrobně popsán v [13].

Při budovaní BSP stromu je hledání optimální hyperplochy pro rozdělení prostoru v podstatě prohledávání trojrozměrného prostoru všech možných řešení. To vede při naivním řešení na časovou složitost podle [13]:

$$O(N^3)$$

kde N je počet primitiv v daném prostoru. Zrychlení opětovně přispívá využití heuristiky SAH, které v ideálním případě sníží složitost na úroveň subkvadratickou:

$$O(N \log^2 N)$$

BSP tree má využití při renderování rozsáhlých scén s nerovnoměrným rozložením objektů v nich.

3.4 Adaptive subsampling

Základní technikou zobrazování sledováním paprsku je vysílání jednoho paprsku pro každý bod obrazu zvlášť. Adaptive subsampling neboli adaptivní podvzorkování je skupina algoritmů, které mají za úkol snížit dobu potřebnou k vykreslení obrazu scény tím, že sníží počet počet pixelů, které jsou vyhodnocovány. Je-li v okolí bodu minimální rozdíl hodnoty a odstínu barev, lze předpokládat, že tento bod bude mít podobnou barvu a můžeme ji dopočítat jako průměr okolních barev.

Tedy základní myšlenkou této metody je rozdělení pixelů do skupin 8x8, 4x4, 2x2 a 1x1. V rozích těchto skupin jsou nejprve vyhodnoceny pixely, které pokud jsou vyhodnoceny jako podobné, jsou vnitřní pixely této skupiny interpolovány (jejich barva, souřadnice do textury apod.) mezi těmito rohovými pixely. V případě, že se tyto rohové pixely neshodují, je skupina rekurzivně rozdělena do čtyř podskupin, které jsou následně vyhodnocovány samostatně dle předchozího postupu.

Ve smyslu metody Adaptive subsampling pixely nazýváme podobné, pokud se v nich zobrazuje stejný objekt, pokud mají téměř stejnou barvu, pokud oba body, které jsou v daných pixelech promítnuty, leží ve stínu nebo mimo něj.

3.5 Paketové procházení

Při paketovém procházení se společně prochází vícero paprsků najednou. Pro tento paket se společně projdou všechny buňky akcelerační struktury, které by dané paprsky navštívily při individuálním procházení. Může se stát, že se jednotlivé paprsky neshodnou na dalším kroku v procházení. V tomto případě je nutno projít postupně všechny možnosti a některé paprsky po dobu vykonávání maskovat.

3.6 Paralelizace výpočtů

Jelikož je proces výpočtu každého výsledného pixelu obrazu na sobě nezávislý, je velkou výhodou ray tracingu jeho snadná paralelizace. Data se zde při syntetizaci nijak nemodifikují, a proto ani není nutné řešit výlučný přístup ke společným zdrojům mezi vlákny. Podle [31] je možná implementace využívající paralelní ray-tracing rozdělit na základě oblasti, kterou daný výpočet využívá. Datově orientované paralelní implementace jsou založené na rozdělení datové akcelerační struktury na vícero procesorů. Využívají vzájemnou komunikaci při výpočtu průsečíků paprsků s primitivami scény. Při tomto postupu se využívá složení a topologie datové struktury. Výpočet je paralelní a každá výpočetní jednotka je zodpovědná za určitý prostor daný částí akcelerační datové struktury. Urychlení výpočtu se v tomto případě dosahuje paralelním prohledáváním podprostorů daných uzly datové struktury, které jsou přidělené jednotlivým výpočetním jednotkám. Výhody datového paralelizmu můžeme vidět ve snížení paměťové náročnosti přerozdělením dat na jednotlivé výpočtové uzly systému. Nevýhodou tohoto přístupu je podle [31] distribuovaní jednotlivých částí výpočtu na výpočtové uzly systému a možná nevyvážená zátěž jednotlivých částí systému v určitých scénách.

4 Zhodnocení současného stavu

Metoda ray tracing je, dle mého názoru, směr, kterým se bude počítačová grafika ubírat. Přesto, že je metoda známa již přes třicet let, se pro svoji výpočetní náročnost používala pouze při statických nebo předpřipravených generováních scén, např. ve filmovém průmyslu. Za zmínku tady stojí uvést, že v roce 1995, při vytváření filmu Toy Story, trvalo vygenerování jednoho snímku průměrně 2 hodiny. V roce 2005, deset let po té, při filmu Cars, však generování trvalo 15 hodin přesto, že celkový nárůst rychlosti počítačů byl až 300 násobný. To vše je způsobeno úrovní detailů, které chtějí tvůrci ve filmu ztvárnit. Právě u těchto a dalších děl hrál ray tracing stěžejní roli.

Já osobně budoucnost vidím v použití této metody v počítačových hrách, kde je v poslední době trend zlepšovat právě realističnost, kterou tato metoda dokonale zvládá. Efekty, které přispívají k vysoké úrovni fotorealismu v počítačových hrách, jsou především reflexe (odrazy světla), refrakce (lámání světla), depth-of-field nebo vysoce kvalitní stíny. Všechny tyto efekty známe i ze současného typu rasterizovaného renderování, kde je jejich simulování daleko těžší. Pro ray-tracing jsou tyto efekty do značné míry přirozené. V dnešní době jsou výpočetní systémy natolik výkonné, že při optimalizovaném a paralelizovaném výpočtu je real – time aplikace možná a dokazují to již mnohé pokusy. Mám za to, že by pro rozvoj této metody pomohla implementace struktur pro výpočty přímo do hardware, podobně jako třeba rasterizace. Jak ray tracing, tak pasterizace, jsou aproximací fyzikálního jevu odrazu světla od povrchu. Ani jeden není podstatně lepší nebo horší – jsou prostě jiné.

Tři možné důvody pro osvojení ray tracingu jsou:

1.) jednoduchost programování
2.) rychlejší vizuální efekty
3.) možnost lepších vizuálních efektů.

Rasterizace je, oproti ray tracingu, opravdu rychlá a existuje spousta metod, jak tento typ renderování dále zefektivnit. Základem pro stavbu zobrazované scény nejsou paprsky, ale trojúhelníky, kde u každého z nich je možné určit pokrytí a následně i určit barvu každého jednotlivého viditelného pixelu. Současné grafické karty jsou postaveny tak, aby co nejefektivněji pracovaly právě při rasterizaci, kdy dokáží zároveň provést mnoho dalších kouzel pro rychlejší a efektivnější zpracování scény. Při tomto typu renderování je každý trojúhelník vykreslován vždy nezávisle. Ray tracing neustále využívá celou geometrii scény a je proto považován, na rozdíl od pasterizace, za globální rendering.

Jednoduchý příklad počtu vyslaných paprsků a testů na průsečík s primitivy nám ukazuje náročnost základního algoritmu:

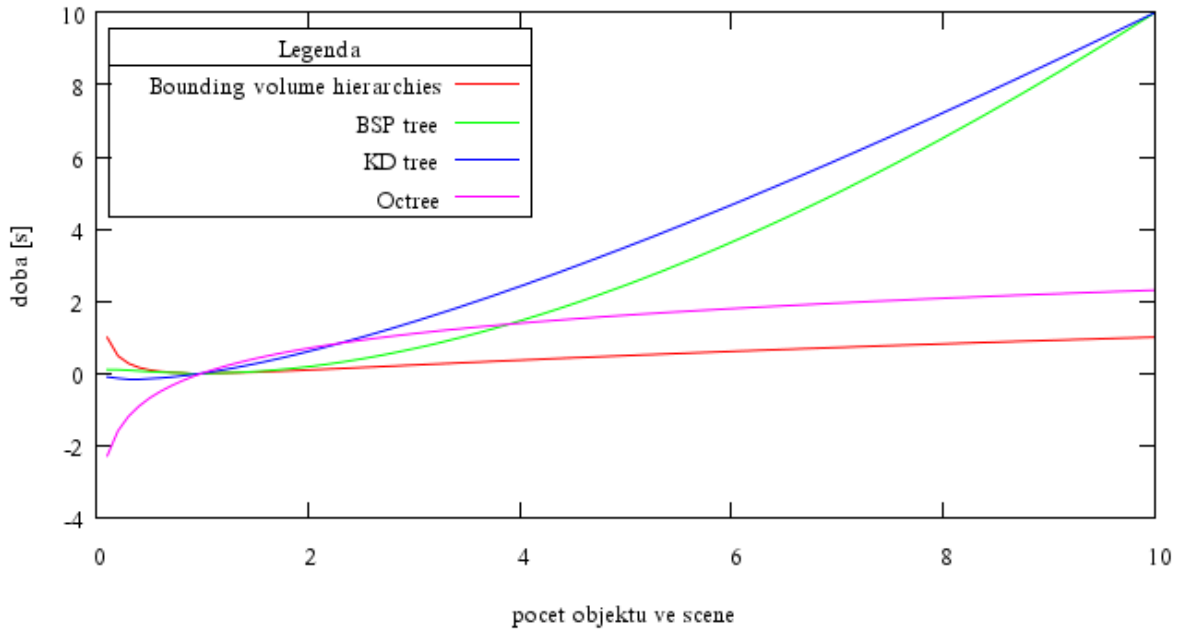
- 1 obrázek: ~ 100 000 000 paprsků
- 1 paprsek hrubou silou: ~ 1 000 000 testů primitiv
- Celkem tedy ~ 100 000 000 000 000 testů

Z příkladu vyplývá, že i pro velmi jednoduché scény trvá výpočet stovky hodin. Pro nalezení průsečíku není třeba testovat všechny objekty. Řešením jsou akcelerační struktury, kde složitost nalezení průsečíku je logaritmicky úměrná počtu polygonů. Logaritmická složitost je opak exponenciální. Exponenciální (geometrická řada) roste extrémně rychle, kdežto logaritmická roste extrémně pomalu.

Zadáním této práce bylo, mimo jiné, prostudovat dostupné optimalizační metody. Tomuto tématu se věnuje předešlá kapitola. Tabulka 1: Srovnání optimalizačních metod srovnává některé metody z hlediska časové složitosti prohledání prostoru. Graf 1. a 2 toto znázorňuje graficky (grafy se od sebe liší měřítkem). V některých případech je však dosažení udávané složitosti nereálné.

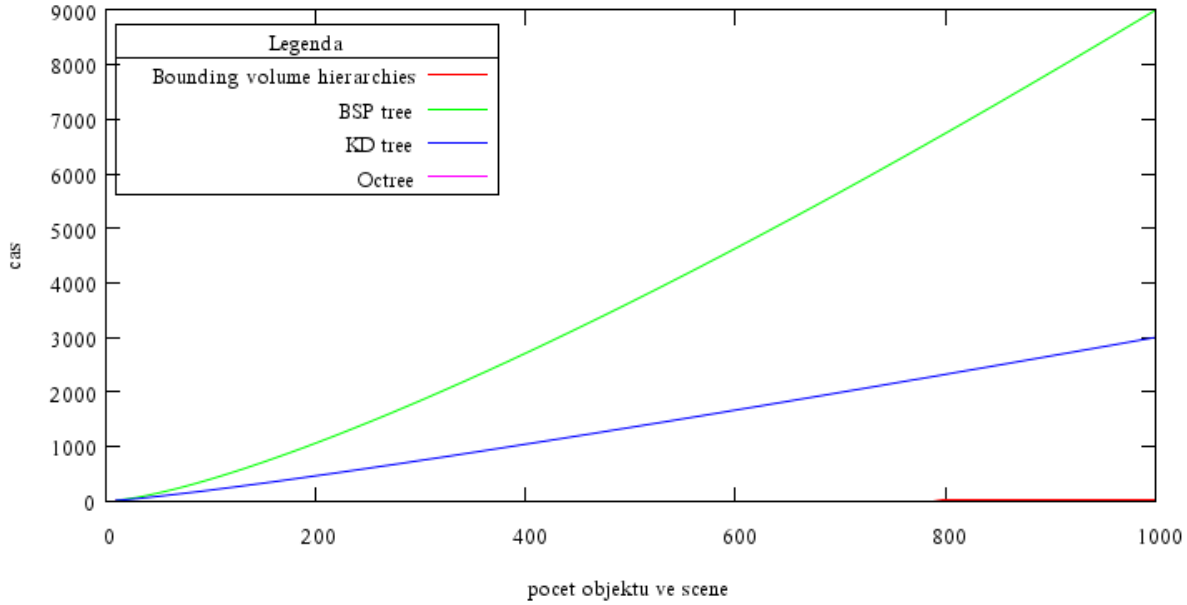
Metoda	Časová složitost
Bounding volume hierarchies	$O(\log^2 N)$
Octree	$O(\log_2 N)$
KD - tree	$O(N \log N)$
BSP tree	$O(N \log^2 N)$

Tabulka 1: Srovnání optimalizačních metod



Graf 1: Srovnání optimalizačních metod

Existují spousty software od velmi dobrých až po horší; za všechny bych jmenoval POV-ray (Persistence of Vision Raytracer) nebo Blender. Já jsem se však pro účely práce rozhodl sestavit vlastní ray tracer, což už je téměř na samotnou práci, kde mi bylo motivací vlastní řešení a implementace jednotlivých struktur. Implementace dosažených znalostí je popsána v následující kapitole.



Graf 2: Srovnání optimalizačních metod

5 Popis implementace

Tato kapitola se věnuje vlastní implementaci systému, který si bere za cíl implementovat ray tracer na základě známých a dostupných algoritmů. K tomu jsem použil struktury uvedeny v knize [1], dále jsem vycházel z ray traceru Aurelius od Adama Herouta a ray traceru od Jacco Bikkera [14].

Nejprve se zaměříme na celkovou strukturu systému a jeho hierarchii tříd. Poté projdeme jednotlivé struktury a algoritmy, řešící základní funkce systému. V neposlední řadě pak je postup pro sestavení a průchod strukturou optimalizační metody Regular grid. Na konci se podíváme na dosažené výsledky tohoto systému.

V implementovaném ray traceru může uživatel měnit osvětlování pomocí několika parametrů tzv. Phongovy parametry (odrazivost, průhlednost, index lomu, apod.), které nastavuje při definici scény.

5.1 Struktura systému a jeho třídy

Systém je složen ze tří hlavních částí - hlavního programu, renderovací oblasti a samotného ray traceru. První z nich, hlavní program, obsluhuje a definuje ray tracer, stará se o propojení jeho výstupů s knihovnou OpenGL, dále pak definuje kameru třídou `Camera` a renderovací scénu třídou `RenderArea`. Obsluhuje vstupy a výstupy na obrazovku. Druhou částí je `RenderArea`, která v sobě nese informace o všech primitivech, jejich vlastnostech a materiálech. Grafická primitiva definuje třída `Shape`. Každý z objektů si nese informace o své geometrii a materiálu (třída `Material`). Poslední, hlavní částí, je RT jednotka, reprezentovaná třídou `Raytracer`. Tato třída zapouzdřuje všechny datové struktury a operace nutné pro práci RT jednotky a její optimalizaci. Takto vypadá základní struktura.

5.2 Základní datové struktury

Mezi základní datové struktury, v našem systému, patří `rgb`, `Vector2` a `Vector3`. Tyto datové struktury jsou využívány všemi částmi systému, a proto je pro nás důležitá vysoká efektivita operací nad těmito datovými strukturami. Všechny, často používané metody, jsou definovány jako inline funkce. To sice poněkud zvětší výsledný program, ale zásadním způsobem ovlivní rychlost běhu. Všechny tři struktury jsou vlastně vektory reprezentující různá data. Třída `rgb` je struktura uchovávající barevnou složku. Je složena ze tří složek - červená, zelená a modrá (red, green a blue). Nad tímto vektorem jsou definovány základní vektorové operace sčítání, odečítání, násobení a dělení.

Dále pak násobení skalárem, dělení skalárem a ořezání (metoda `clamp()`) pro případ, že potřebujeme zajistit, aby nedošlo k opuštění intervalu [0;1].

Struktury `Vector2` a `Vector3` uchovávají body ve dvou-rozměrném respektive tří-rozměrném prostoru. Jejich struktura i definované operace jsou stejné. `Vector3` má však všechny metody o jednu souřadnici navíc. Skládá se tedy ze složek x, y a v případě `Vector3` z. Základní operace jsou stejné, jako u struktury `rgb`, plus metody specifické pro práci s vektory, jako výpočet délky vektoru, délka umocněná na druhou, výpočet jednotkového vektoru, normalizovaného vektoru. Dále pak vektorové operace skalární součin, vektorový součin, smíšený součin a operace pro porovnání.

5.3 Renderovací oblast

Výše zmíněná třída `RenderArea` v sobě nese informace o všech modelovaných objektech, světlech a optimalizační strukturu s informacemi pro optimalizaci pomocí metody Regular grid. Mezi její metody patří `init()`, která vygeneruje objekty a nastaví jim materiálové vlastnosti. Další důležitou metodou je `buildGrid()` jejíž funkcí je vygenerovat mřížku, která se používá při optimalizaci. Objekty jsou definované třídou `Shape`, která je abstraktní virtuální třídou. Abstrahuje rozhraní jednotlivých primitiv tak, že je možné s nimi pracovat jednotným způsobem. Třída `Shape` deklaruje několik metod, které musí mít každý potomek. Mezi ty nejdůležitější patří metoda `hit()`, která zjišťuje, zda došlo k protnutí paprsku s primitivem a vrací odpověď booleovského typu; průsečík s daným primitivem a další informace vrací ve struktuře `hitRecord`. Metoda `shadowHit()` pouze zjišťuje, zda došlo k protnutí. Pro optimalizaci je pak nutná metoda `intersectBox()` používající se pro sestavení optimalizační mříže. Je zřejmé, že všechny tyto funkce jsou pro metodu sledování paprsku stěžejní. Pro výpočet průsečíků se využívají výše definované soustavy rovnic, odvislé od tělesa, pro které je prováděno.

5.4 Materiál objektů

V programu je definována třída `Material`, která udržuje materiálové konstanty nutné pro výpočet osvětlení. Ambientní, difuzní a spekulární odrazivost, parametr udávající ostrost odlesku a jiné vlastnosti materiálu, jako je index lomu (refractiveIndex). Více se tématu osvětlení věnujeme ve druhé kapitole.

5.5 Ray tracer

Třída `RayTracer` je pak samotná implementace základního rekurzivního algoritmu a optimalizační metody Regular grid. Setkávají se zde všechny doposud uvedené algoritmy a datové struktury a tvoří funkční celek. Základní metodou třídy je `render()`, která prochází scénu bod po bodu, vygeneruje paprsek a ten předá funkci `rayTrace()`. Výslednou barvu pak uloží do rasteru, který je definován v třídě `Image`. Popsaný postup je znázorněn pseudokódem algoritmu:

```
Render()
  Pro každý pixel
        ray = getRay()  // sestav paprsek z kamera pro pixel
        barva = rayTrace(ray,depth) // sleduj paprsky, hloubka zanoření
        ulož barva do rasteru
```

Algoritmus 1: Renderování

Funkce `rayTrace()` sleduje paprsek a testuje jej na průnik s definovanými tělesy ve scéně. Pokud je nalezen průsečík, zjistí barvu a osvětlení v tomto místě a vyšle další paprsky způsobené odrazem. Tím se rekurzivně zanořuje, až do doby maximální hloubky zanoření, střetem se světelným zdrojem nebo dopadem na difuzní povrch. V případě, že se paprsek nesetká s objektem, je nastavena barva pozadí.

```
raytrace(paprsek, hloubka) : vrací vypočtenou barvu
     barva = barva pozadí
     je-li hloubka <= 0
     vrať = barva
     průsečík = najdi_nejbližší_průsečík(paprsek)
     je-li průsečík platný
     barva = vypočti_lokální_osvětlení(průsečík)
     je-li materiál průsečíku lesklý
        barva += koeficient_lesklosti(materiál průsečíku)
        *rayTrace(vytvoř_zrcadlový_paprsek(průsečík),
     hloubka-1)
     je-li materiál průsečíku průsvitný
        barva += koeficient_průhlednosti(materiál průsečíku)
        * rayTrace(vytvoř_lomený_paprsek(průsečík), hloubka-1)
 vrať barva
```

Algoritmus 2: Ray tracing

5.6 Optimalizační metoda

Pro optimalizaci základního algoritmu byla implantována metoda regular grid (pravidelná mřížka). K sestavení mřížky můžeme přistupovat dvěma směry. Nejprve je potřeba definovat obalové těleso, nejlépe AABB(Axis -Aligned Bounding Box) neboli obalová krychle, která je osově zarovnaná. Dále definujeme množinu těchto krychlí, která reprezentuje celou mříž. Jedním ze samotných algoritmů pak je:

```
Pro všechna x < velikost mřížky
 Pro všechna y < velikost mřížky
  Pro všechna z < velikost mřížky
    Zjisti pozici krychle
    Procházej všechna primitiva{
       Testuj, zda leží v krychli
       Pokud leží
          Přidej těleso do této krychle
```

Algoritmus 3: Sestavení mřížky I

Toto však není příliš efektivní přístup, neboť procházíme všechny krychle a u každé z nich procházíme všechna tělesa. Lepší a efektivnější přístup pak navrhuje algoritmus:

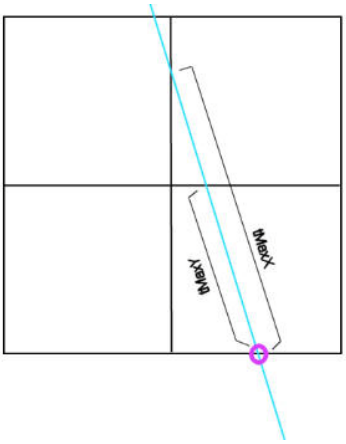
```
Procházej všechna tělesa{
   Dej obalovou krychli tohoto tělesa
   Zjisti, které krychle v mřížce by mohly obsahovat toto těleso
   Procházej kandidátní krychle{
      Testuj, zda leží v krychli
      Pokud leží
        Přidej těleso do této krychle
   }}
```

Algoritmus 4: Sestavení mřížky II

Zde musíme pro tělesa ve scéně definovat jejich obalová tělesa, pak pro každý prvek ve scéně hledáme, do které krychle patří. Prochází se tedy jenom tělesa, která nejsou prázdná. Na základě popsaného algoritmu se právě o sestavení se stará metoda `buildGrid` třídy `RenderArea`.

Poté, co máme mřížku sestavenou, můžeme ji procházet, tzv. traverzovat. Tato metoda používá 3D verzi algoritmu DDA[8]. Jako první musíme zjistit, ve které buňce paprsek začíná. Pokud je parsek mimo hranice mřížky, musí být posunut. Každá buňka má své souřadnice`(X,Y,Z)`. Dále

musíme zjistit délku, kterou paprsek urazí před zasažením buňky. Ta je pak uložena v `tMaxX, tMaxY, TMaxZ`.



Obrázek 12: Průnik paprsku mřížkou

Na obrázku 12 můžeme vidět místo střetu paprsku s mřížkou (kroužek). Od tohoto bodu budeme pohybovat `tMaxX` k nalezení průsečíku s vertikální osou nebo `tMaxY` k nalezení průsečíku s horizontální osou. Minimální z těchto tří hodnot nám určuje, jak daleko můžeme hodnoty posunovat. Konečně si zjistíme, jak daleko musíme posunovat paprsek k překročení buňky. Výsledek uložíme do `tDeltaX`, `tDeltaY` a `tDeltaZ`. Ve výše uvedeném obrázku se rovná `tDeltaY tMaxY`. Jakmile budeme mít proměnné nastaveny, můžeme začít krokování sítě. Krokování buněk je znázorněno na následujícím pseudokódu:

```
loop
{
        if (tMaxX < tMaxY)
        {
                tMaxX = tMaxX + tDeltaX;
                X = X + stepX;
        }
        else

        {
                tMaxY = tMaxY + tDeltaY;
                Y = Y + stepY;
        }
}
```

Algoritmus 5: Procházení strukturou mřížky

V tomto kódu je `stepX` nastaveno na hodnotu '-1' pokud znaménko x-ové souřadnice směrového vektoru paprsku je negativní, jinak je nastaveno na 1. `stepY` se počítá stejným způsobem. 3D verze je o něco delší, ale funguje stejným způsobem a přesahuje rámec této práce. Důležitou funkcí pro optimalizaci je `findNearest()` ze třídy `RayTracer`; ta hledá nejbližší průsečík s tělesem pomocí metody popsané výše. Pokud není optimalizace zapnuta, prochází se všechny objekty a testují se na průnik s paprskem.

5.7 Výsledky

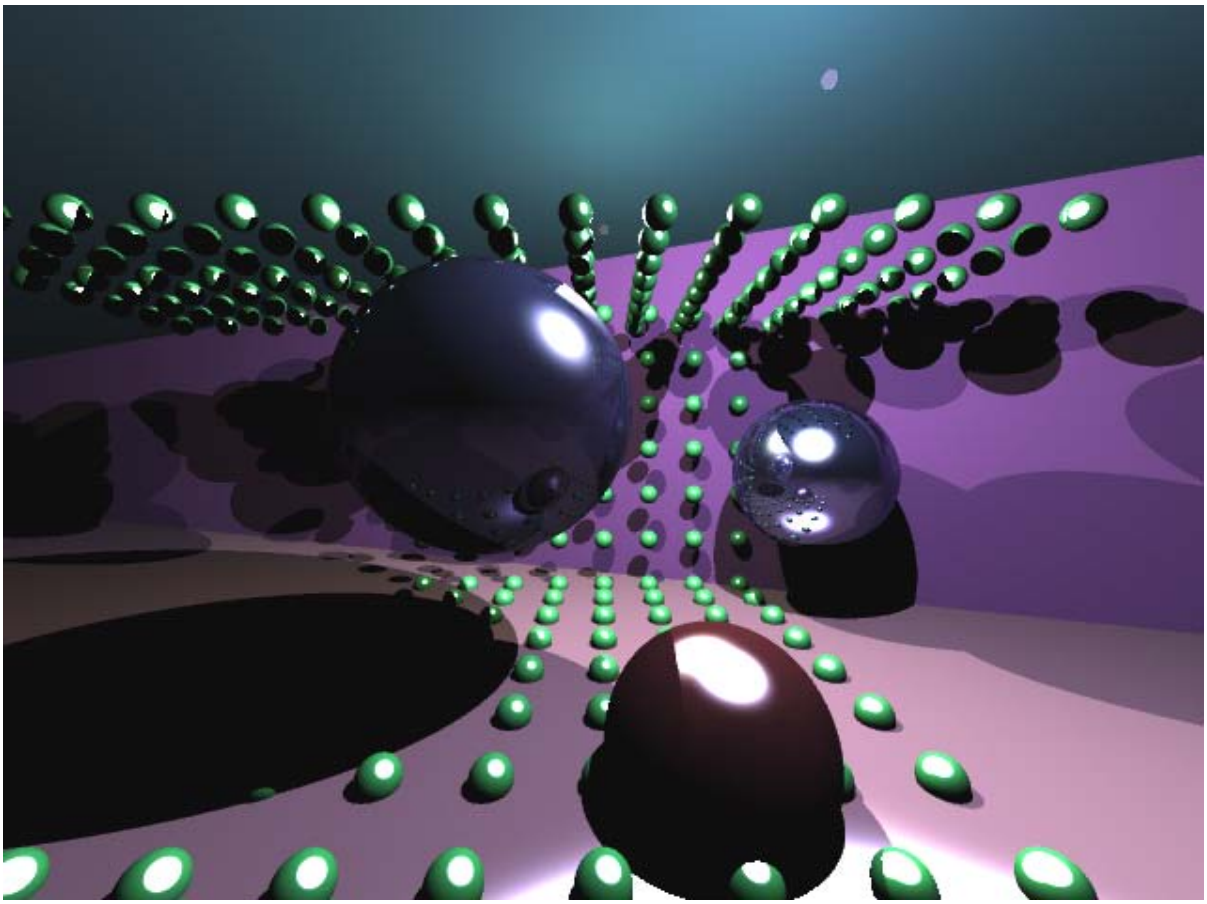
Systém byl postupně testován na scéně složené z 8, 64, 128 a 256 objektů. Jednou prováděné s optimalizací, podruhé bez ní. Velikost rasteru byla nastavena na 640 x 800. Dosažené výsledky jsou shrnuty v tabulce 2. Čas potřebný pro generování scény byl měřen na základě algoritmu 6. Z výsledků lze usoudit, že tato optimalizační metoda urychluje výpočet minimálně o řád. Na obrázku 11, je výsledný obraz scény vygenerovaný vytvořenou aplikací s 256 objekty. Definice scény byla přejata z [14].

Počet objektů	8	64	128	256
Bez optimalizace	5,4 s	31,4 s	64,4 s	115,5 s
S optimalizací	4,1 s	4,5 s	5,7 s	9,3 s

Tabulka 2: Dosažené výsledky

```
Cas_zacatku = clock()
//měřený úsek programu
Cas_konce = clock()
Delka trvani = Cas_konce - Cas_zacatku
```

Algoritmus 6: Měření doby trvání části programu



Obrázek 13: Výsledný obraz generované scény

6 Závěr

Cílem této bakalářské práce bylo především nastudovat, navrhnout a vytvořit aplikaci s implementovanou metodou pro optimalizaci metody sledování paprsku s ohledem na možnost budoucí rozšiřitelnosti aplikace. Tento cíl byl splněn. Po prostudování dostupné literatury na téma sledování paprsku, byla ověřena možnost rozdělení scény pomocí pravidelné mřížky, popřípadě na podprostory a urychlení výpočtu průsečíku paprsku s objektem. Na základě nově nabytých znalostí byla navrhnuta za pomoci objektově orientovaného přístupu struktura programu, která umožňuje implementaci různých optimalizačních technik sledování paprsků. Podle návrhu byla nakonec implementována výsledná aplikace, která nakonec byla podrobena sadě testů.

Z tabulky testu vyplývá, že úvahy v návrhu byly správné a vedly k celkovému zrychlení generovaného obrazu. Pro scény se stovkami objektů došlo u metody Regular grid (uniformní mřížka) ke zrychlení až o řád. Nicméně metoda je vhodná pro velké scény s pravidelným rozmístěním objektů.

Z nastudovaných materiálů dále vyplynulo, že v oblasti ray tracingu je stále mnoho příležitostí k dalšímu vývoji zejména v optimalizaci. V současnosti jsou intenzivně zkoumány možnosti použití ray tracingu pro interaktivní scény při real-time aplikacích.

Případný další vývoj aplikace by mohl být zaměřen několika směry. Prvním by mohl být vylepšení fotorealističnosti výsledného obrazu. Zde jde například o metody, jako jsou měkké stíny nebo hloubka ostrosti, popřípadě implementaci kaustiky, aby skleněné předměty vypadaly opravdu realisticky. Jako další oblast, která už byla započata při řešení této práce, by mohla být implementace algoritmů pro dělení prostoru kd-tree a octree nebo dalších optimalizačních struktur a technik, například akcelerace výpočtů přes GPU (grafické karty) a paralelizace. Vytvoření příznivějšího uživatelského rozhraní, kde bylo možno ovládání kamery, či možnost interaktivní změny scény uživatelem, je další oblast, kterým se může další vývoj ubírat.

Literatura

[1] SHIRLEY P., MORLEY R.K.: *Realistic RayTracing*, A. K. Peters, Ltd., Natick,MA, USA,second edition, 2003. ISBN 1-56881-198-5

[2] GLASSNER A. S., editor, *An Introduction to Ray Tracing*, Academic Press, 1989. ISBN 0-12-286160-4

[3] HAVRAN, V.: *Heuristic Ray Shooting Algorithms*, Dizertační práce, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Listopad 2000.
Dostupné na URL http://www.cgg.cvut.cz/~havran/phdthesis.html

[4] BIKKER J.: *Ray Tracing through the eyes of a Game Developer*, RT'07,IEEE Symposium on Interactive Ray Tracing, 2007

[5] WHITTED T.: *An improved illumination model for shaded display*, In ACM SIGGRAPH volume 23, issue 6, str: 343 – 349, 1980

[6] EISEMANN M., WOIZISCHKE C., MAGNOR M.: *Ray Tracing with the Single Slab Hierarchy*, Computer Graphics Lab, TU Braunschweig, 2008. Dostupné na URL http://graphics.tu-bs.de/publications/Eisemann08SSH.pdf

[7] FUJIMOTO A., TANAKA T., IWATA A K.: *Accelerated ray-tracing systém*, IEEE Compututer, Graphics and Applications, str.: 16–26, Duben 1986.

[8] AMANATIDES J., WOO A.: *A Fast Voxel Traversal Algorithm for Ray Tracing*, Dept. of Computer Science, University of Toronto, 1987. Dostupne na URL http://www.devmaster.net/articles/raytracing_series/A faster voxel traversal algorithm for ray tracing.pdf

[9] GOLDSMITH J. AND SALMON J.: *Automatic creation of object hierarchies for ray tracing*, IEEE,Computer Graphics and Applications, s. 14–20, Květen 1987.

[10] KAY T. L. AND KAJIYA J. T.: *Ray tracing komplex scenes*, SIGGRAPH 1986: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, str.: 269–278, New York, NY, USA, 1986. ACM Press.

[11] RUBIN S.M. AND WHITTED T.: *A 3-dimensional representation for fast rendering of complex scenes*, SIGGRAPH 1980: Proceedings of the 7th annual conference on Computergraphics and interactive techniques, str.: 110–116, New York, NY, USA, 1980. ACMPress.

[12] LANK V., VONDRA M.: *Fyzika v kostce pro střední školy*, Fragment 2007.

[13] WALD, I.; HAVRAN, V.: *On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)* [online], 2007, [cit. 2011-05-01]. Dostupné z URL: http://www.cgg.cvut.cz/members/havran/ARTICLES/ingo06rtKdtree.pdf.

[14] BIKKER J.: Ray *Tracing Theory and Implementation, series of articles*, Dostupné na: Flipcode.com, Devmaster.net

[15] KUŽELKA O.: *Java a 3D grafika – světla*[online], 2004[cit. 2011-05-01], Dostupné na URL: http://interval.cz/clanky/java-a-3d-grafika-svetla/

[16] PHONG B. T., *Illumination for computer generated pictures*, Communications of ACM 18 (1975), s. 6, 311–317.

[17] ZEMČÍK P.: *Přednášky k předmětu PGR - Počítačová grafika, Sledování paprsku*,VUT Brno, Fakulta informačních technologií: 2006

[18] *Snell's law - vector form*[online], Wikipedia, [cit. 2011-05-01],
Dostupné na URL< http://en.wikipedia.org/wiki/Snell's_Law#Vector_form >

[19] *Beer-lambert law*[online], Wikipedia, [cit. 2011-05-01], Dostupné na URL: http://en.wikipedia.org/wiki/Beer's_Law

[20] TRAN G. : *Glasses, pitcher, ashtray and dice* [online], Dostupné z URL http://www.oyonale.com/modeles.php?lang=en&page=40

[21] BORMAN S.: *Raytracing and the camera matrix - a connection. A tutorial on therelationships between raytracing formulations of projective geometry and the standard camera matrix representation*, červen 2003.

[22] BATALI D., BASHFORTH B., BERNARDI C., CHRISTENSENA P. H., LAUR D., HERYA C., QUARONI G., TOMSON E., JORDAN T. and WOOTEN W.: *RenderMan, Theory and Practice*, Červenec 2003, Siggraph 2003

[23] *Non-uniform rational B-spline* [online], Wikipedia, [cit. 2011-05-01], Dostupné z URL http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline

[24] *Bounding volume*[online], Wikipedia, [cit. 2011-05-01], Dostupné z URL http://en.wikipedia.org/wiki/Bounding_volume

[25] IZE, T.; WALD, I.; PARKER, S. G.: *Ray Tracing with the BSP Tree* [online], 2008, [cit. 2011-05-01]. Dostupné z URL: http://visual-computing.intel-research.net/publications/BSPRT08.pdf.

[26] KAY, T. L.; KAJIYA, J. T.: *Bounding Volumes* [online], 1986, [cit. 2011-05-01]. Dostupné z URL: http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtaccel.htm.

[27] HAVERKORT, H. J.: *Introduction to bounding volume hierarchies* [online], 2004, [cit. 2011-05-01]. Dostupné z URL: http://www.win.tue.nl/~hermanh/stack/bvh.pdf.

[28] KAJIYA, J. T.: *The rendering equation* [online], 1986, [cit. 2011-05-01]. Dostupné z URL: http://www.cs.princeton.edu/courses/archive/fall02/cs526/papers/kajiya86.pdf.

[29] KNOLL, A.: *A Survey of Octree Volume Rendering Methods* [online], 2003, [cit. 2011-05-01]. Dostupné z URL: http://www.cs.utah.edu/~knolla/octsurvey.pdf.

[30] JENSEN, H. W.: *Global illumination using photon maps. In Proceedings of the eurographics workshop on Rendering techniques*, 1996, London, UK: Springer-Verlag, 1996, ISBN 3-211-82883-4, s. 21-30

[31] KWON, C.-G.; SUNG, H.-K.: *An implementation of a paralell ray tracing algorithm on hybrid parallel architecture* [online], 2003, [cit. 2011-05-01]. Dostupné z URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00681796&tag=1.

[31] *Ray Tracing*[online], Wikipedia, [cit. 2011-05-01], Dostupné z URL: http:// http://en.wikipedia.org/wiki/Ray_tracing_(graphics)

[32] ENTHAS: *Začínáme s cinema 4d - úvod do 3d grafiky a základní pojmy* [online], TutorialArts, Dostupné z URL: http://www.tutoriarts.cz/zaciname-s-cinema-4d-uvod-do-3d-grafiky-a-zakladni-pojmy-1159

Dodatek A

Obsah DVD

- Elektronická verze textu bakalářské práce ve formátu PDF a její zdrojový text pro MS Word
- Vytvořené obrazy testovacích scén
- Zdrojový kód implementace popisované v práci včetně řešení pro Microsoft Visual C++ 2008 Express Edition a všemi potřebnými soubory pro přeložení a rendering testovacích scén
- Binární soubory
- Manuál aplikace