VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

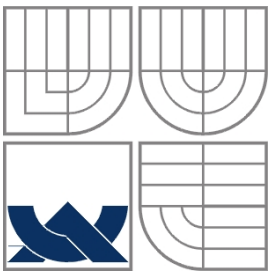
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ KULEČNÍKU POMOCÍ DISTRIBUOVANÉHO SLEDOVÁNÍ PAPRSKU

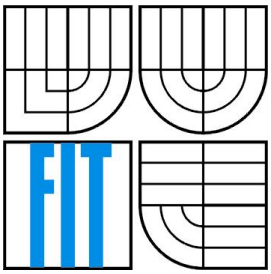
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Marián Krivda
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ KULEČNÍKU POMOCÍ DISTRIBUOVANÉHO SLEDOVÁNÍ PAPRSKU

RENDERING BILLIARD USING DISTRIBUTED RAYTRACING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Marián Krivda

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Adam Herout, Ph.D.

BRNO 2009

Abstrakt

Práce se zabýva studiem metody realistického zobrazení scény prostrednictvím distribuovaného sledování paprsku. Tato metoda simuluje různé vizuální efekty a tím generuje dvourozměrné obrázky s vysokou mírou realističnosti. Práce rozebírá problematiku a vysvětluje postupy řešení s touto technikou spojené. Součástí je i popis metody jednoduchého sledování paprsku, protože je základem distribuovaného sledování paprsku. Část práce se věnuje optimalizaci distribuovaného sledovaní paprsku.

Abstract

This thesis is concerned in the method of realistic rendering using a distributed raytracing.
This method simulates various visual effects and generates high realistic 2D images. The work analyses the problem and explains principles of solution related to this technique. There is also descriprion of the method of simple reytracing which provides a basis for the distributed raytracing. A part of work is specialized for optimalization of distributed raytracing.

Klíčová slova

distribuovaný raytracer, stochastický raytracer, metoda sledování paprsku, raytracing, mäkké stíny, hloubka ostrosti, rozmazání pohybem, refrakce, zrcadlové odrazy

Keywords

distributed raytraing, stochastic raytracing, raytracing, soft shadows, depth of field, motion blur, refraction, mirror reflection

Citace

Marián Krivda: Zobrazení kulečníku pomocí distribuovaného sledování paprsku, diplomová práce, Brno, FIT VUT v Brně, 2009

ZOBRAZENÍ KULEČNÍKU POMOCÍ DISTRIBUOVANÉHO SLEDOVÁNÍ PAPRSKU

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Marián Krivda
22.5.2009

Poděkování

Děkuji vedoucímu mé diplomové práce Ing. Adamu Heroutovi Ph.D. za jeho čas a cenné rady, které mi poskytl během řešení.

Obsah

1 Úvod

Metóda sledovania lúča tzv. raytracing je technika vizualizácie scény založená na šírení svetelných lúčov priestorom. Základom raytracingu je vrhanie lúčov z kamery do priestoru a neustále hľadanie priesečníkov s objektami v scéne. Algoritmy jednoduchého raytraceru sú postavené na základoch lineárnej algebry (vektory, matice) a fyzikálnej optiky (lom a odraz svetla). Ako každá iná metóda, aj metóda sledovania má svoje výhody a na druhú stranu aj limity. Obrázky generované takýmto jednoduchým raytracerom rešpektujú fyzikálne zákony, ale pritom vyzerajú trochu umelo (ostré hrany a tiene, bodové zdroje svetla, ...). Obmedzenia tohto jednoduchého raytraceru je možné odstrániť modifikáciou algoritmu sledovania lúča. Takýto raytracer potom namiesto jedného lúča vrhá zväzok vhodne rozložených lúčov, čím sa základný raytracer rozšíri na distribuovaný, posúvajúci hranice kvality zas kúsok ďalej. Prednosťou distribuovaného raytraceru je jednak odstránenie spomínaných negatív a simulovanie rôznych efektov, ktoré s jednoduchým raytracerom nie sú možné. S pomedzi možných efektov sú zaujímavé simulácie rozmazaných odrazov, antialiasing, mäkké tiene, hĺbka ostrosti („depth of field“) a rozmazanie pohybujúcich sa objektov („motion blur“). Pod názvom distribuovaný raytracer si väčšina ľudí predstavuje sieťový raytracer, bežiaci na viacerých počítačoch, ale v skutočnosti vôbec nejde o počet počítačov. Raytraceru bežiacemu na viacerých počítačoch prináleží názov paralelný raytracer. Pojem distribuovaný raytracer má svoje základy v matematike, s ktorou spája distribuovanie lúčov.

Cieľom tejto práce je popísať, navrhnúť a implementovať metódu distribuovaného sledovania lúča, a tiež navrhnúť a implementovať techniky urýchlenia tejto metódy. Výstupy z programu predviesť vo forme obrázkov, porovnať a zhodnotiť dosiahnuté výsledky voči klasickému raytracingu, ako aj zhodnotiť prínos navrhnutých urýchlení.

Nasledujúca kapitola popisuje princíp a fungovanie metódy sledovania lúča, ďalej tiež opisuje rozšírenie tejto metódy v podobne distribuovaného raytracingu. Po nej nasleduje popis urýchľovacích techník, a kapitola venujúca sa návrhu demonštračného programu. Na konci práce sú uvedené dosiahnuté výsledky, a nápady pre ďalšie možné rozšírenia.

2 Distribuovaný raytracing

V tejto kapitole nájdete informácie teoreticky vysvetľujúce základy metódy sledovania lúča, základy matematiky distribučných funkcií, integrálu Monte Carlo a záverečná podkapitola popisuje princípy distribuovaného raytraceru, nazývaného aj stochastický raytracer. Práva časť preberá jednoduchý raytracing a pokrýva jeho základy, nesnaží sa ísť príliš do hĺbky, pretože všetky informácie týkajúce sa základného raytracingu je jednak možné vyhľadať v množstve publikácií venujúcich sa tejto problematike a táto práca je zameraná na distribuovaný raytracing. Druhá časť pokrýva metódu integrácie Monte Carlo, a distribučné funkcie, zaisťujúce vhodný rozptyl lúčov. Tretia časť popisuje možné spôsoby distribúcie lúčov, a prezentuje dosiahnuteľné efekty ako mäkké tiene, realistickejšie odrazy, ...

2.1 Raytracing

Raytracing je elegantná technika realistického zobrazovania trojrozmernej grafiky s komplexnými svetelnými interakciami. Prakticky to znamená, že dovoľuje vytvárať pekné scény plné zrkadiel, priehľadných plôch, svetiel a tieňov. Algoritmus sledovania lúča vychádza z fyzikálneho šírenia svetelných lúčov postupujúcich priestorom. Všetky lúče vychádzajú zo zdrojov svetla (lampy, slnko, …), putujú priestorom, a pri dopade na povrch, sa od neho odrážajú (odrazy) a lámu (priehľadnosť). Časť lúčov nakoniec dopadne na sietnicu oka, kde vytvárajú farebný obraz. Pre napodobenie fyzikálneho šírenia svetla môžeme zo svetelných zdrojov vrhať lúče do scény až pokiaľ časť vrhnutých lúčov dopadne na celú plochu projekčnej roviny, aby pokryli každý jeden pixel obrazu (forward ray tracing). V praxi je takéto riešenie veľmi náročné a neefektívne, pretože generuje prebytočné lúče, ktoré nikdy nedopadnú na projekčnú rovinu. Alternatívou je otočenie smeru lúčov tak, že ich budeme vrhať z projekčnej roviny do scény a sledovať či narazia do svetelného zdroja alebo nie, túto metódu nazývame spätné sledovanie lúča (back raytracing). Pretože sa priamy (forward) raytracing skoro vôbec nepoužíva, označujeme slovom raytracing spätný raytracing.

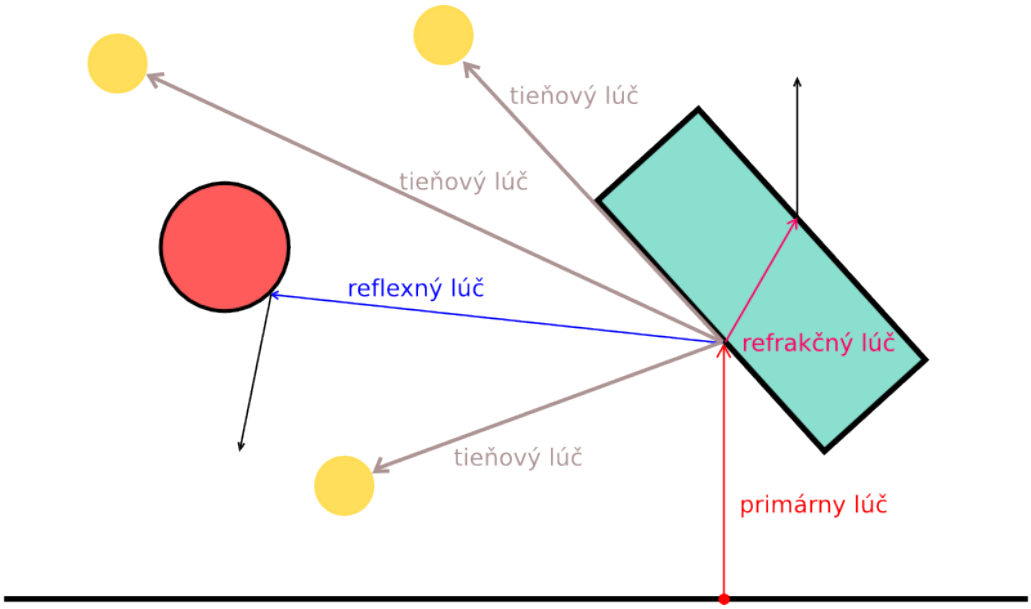
Cez každý pixel obrazu vyšleme lúč z kamery smerom do scény a hľadáme najbližší priesečník s objektami v scéne. V tomto bode potom vyhodnotíme osvetľovací model. Vyslaný lúč v skutočnosti nekončí pri prvom dopade na teleso, môže sa ďalej lámať alebo odrážať. Cestu lúča by sme mohli sledovať až dovtedy, kým neopustí scénu, ale prakticky sa lúč sleduje len do určitej hĺbky.

Pri spätnom sledovaní lúča sa používa viacero druhov lúčov, typy a význam jednotlivých lúčov prezentuje nasledujúca tabuľka.

Typ	Popis
Primárny	Je lúč vyslaný cez pixel obrazu do scény, celkový počet primárnych lúčov sa rovná počtu pixlov obrazu.
Sekundárny	Tento lúč sa vytvára pri dopade lúča na teleso, po dopade na teleso sa lúč môže odraziť, alebo ním prechádza a môže sa lámať (prípadne oboje naraz). Z jedného lúča môžu vzniknúť až dva lúče – odrazený lúč a lomený lúč.
Tieňový	Tento lúč sa vysiela po dopade primárneho alebo sekundárneho lúča z bodu dopadu k svetelnému zdroju. Ak lúč k svetelnému zdroju priamo dorazí, tak je daný bod týmto zdrojom osvetlený a zdroj sa zahrnie do osvetľovacieho modelu. Ak lúč narazí na nejaké teleso, potom je tento zdroj zatienený a daný bod leží v tieni.

Tabuľka 1: Typy lúčov v raytracingu

Jasnejšiu predstavu o použití jednotlivých druhov lúčov si môžete spraviť z nasledujúceho obrázka, zobrazujúceho všetky typy lúčov.



Obrázok 1: Raytracing: popis lúčov

Z projekčnej roviny vyšleme lúč, pre ktorý hľadáme najbližší priesečník s objektami v scéne, v tomto bode sa vypočíta osvetľovací model. Na výpočet farby daného bodu sa využije Phongov osvetľovací model rozšírený o reflexnú (reflection) a refrakčnú zložku (transmission).

ambientná zložka I_A – svetlo dopadajúce na objekt zo všetkých smerov

difúzna zložka I_D – svetlo, ktoré prichádza zo svetelného zdroja a na povrchu telesa je rozptýlené

spekulárna zložka I_S – svetlo, ktoré prichádza zo svetelného zdroja a v danom bode sa od povrchu zrkadlovo odrazí

odrazová zložka I_R – je to svetlo, ktoré prichádza do daného bodu zo smeru odrazu. Toto svetlo je prítomné, ak daný bod je osvetlený svetlom, ktoré nepochádza zo svetelného zdroja, ale vznikne odrazom od iného telesa. Pre biliard je efektné zosilniť odrazy pochádzajúce zo svetelných zdrojov.

lomová zložka I_T – svetlo, ktoré prichádza do daného bodu zo smeru lomu (priehľadnosť)

Celkové osvetlenie je dané súčtom zložiek:

$$I = I_A + I_D + I_S + I_R + I_T$$

Algoritmus spätného raytracingu môžeme zapísať nasledujúcim pseudokódom:

```
pre kazdy pixel obrazu
{
     vytvor luc s pociatkom v kamere, prechadzajuci pixelom obrazu
     farba = SledujLuc(luc)
     zapis farbu do vysledneho obrazu
}

SledujLuc (luc)
{
     nastav farbu na ciernu
     vypocitaj priesecniky so vsetkymi objektami v scene
     vyber najblizsi priesecnik
     ak luc nepretina ziaden objekt, vrat farbu ako ciernu

     ku kazdemu svetelnemu zdroju vyšli tienovy luc
     pre nekriziace sa tienove luce spocitaj osvetlovaci model

     ak je material reflexny, generuj odrazeny luc a vypocitaj
     jeho farbu : sleduj luc (reflected)

     ak je material priehladny, generuj lomeny luc a vypocitaj jeho
     farbu : sleduj luc (refracted)

     spocitaj sucet predchadzajucich vysledkov a vrat ako vvslednu
     farbu luca
}
```

Výhody raytracingu:

- rozšírená metóda

- jednoducho implementovateľná
- presné zobrazenie objektov (matematický popis)
- kvalitné výstupy, vhodné napr. pre technické modely

Nevýhody raytracingu:

- s rastúcou zložitosťou scény rapídne rastie náročnosť (možno optimalizovať - kd-tree, octree)
- ostré tiene (možno použiť mnoho bodových svetiel, čím ale narastie zložitosť)
- ostré odrazy
- dokonalý lom svetla
- odrazené svetlo neosvetľuje okolité objekty

2.2 Monte Carlo integrál, distribučná funkcia

2.2.1 Náhodná premenná, funkcia hustoty pravdepodobnosti, distribučná funkcia

Náhodná premenná je číslo, alebo vektor hodnôt nadobúdajúci „náhodne“ hodnoty. Chovanie náhodnej premennej charakterizujeme rozdelením hodnôt, ktoré môže nadobúdať. Rozdelenie jej hodnôt môžeme zapísať ako distribučnú funkciu náhodnej premennej (príp. funkciu hustoty pravdepodobnosti). Distribučná funkcia tak definuje obor hodnôt náhodnej premennej, ako aj pravdepodobnosť, s akou tieto hodnoty nadobúda.

Koncentrovaný popis rozdelenia náhodných premenných využíva číselné hodnoty, ktoré nazývame charakteristiky náhodných premenných. Najčastejšie používanými charakteristikami sú stredná hodnota EX (popisujúca polohu náhodnej premennej) a rozptyl DX (popisujúci variabilitu).

$$EX = \int_{-\infty}^{\infty} f(x)\,p(x)\,dx \tag{1}$$

$$DX = \int_{-\infty}^{\infty} (x - EX)^2 f(x)\,dx \tag{2}$$

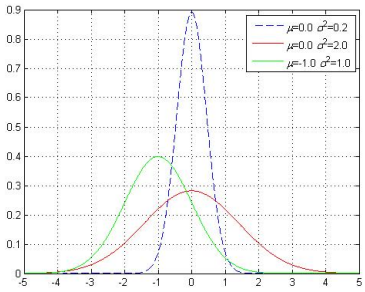
Distribučnú funkciu volíme podľa toho aký jav chceme simulovať, pre tento projekt sú vhodné nasledujúce rozdelenia pravdepodobnosti:

- **rovnomerné rozdelenie**, pri tomto rozdelení má každá hodnota rovnakú pravdepodobnosť že nastane (napríklad pri hode kockou, ktorá má šesť strán, má každá strana rovnakú pravdepodobnosť). V distribuovanom raytracingu je vhodné pre distribúciu lúčov na šošovke pri simulácii hĺbky obrazu, alebo pre rozdelenie lúčov v čase, pri simulácii efektu „motion blur“

- **normálne** (Gaussove), je dané gaussovou krivkou, ktorej obsah pod krivkou je rovný jednej. Je vhodné pre simulovanie zložitých náhodných systémov, kde nieje presne zrejmé, ktorá veličina má vplyv na daný stav. Pre raytracing ho je možné použiť pre simuláciu drsného povrchu pri matných odrazoch, aj rozmazanej refrakcii.
- **trojuholníkové**, je dané troma bodmi *a*, *b*, *c* . Jeho hustota pravdepodobnosti sa podobá trojuholníku s vrcholom v bode *b* (miesto s najvyššou pravdepodobnosťou), body *a* a *c* určujú nulovú pravdepodobnosť. Tiež je vhodné pre rozptyl lúčov v čase, pričom produkuje rozdielny výsledok ako keby sme použili rovnomerné rozdelenie.

Normálne rozdelenie môžeme vyjadriť funkciou hustoty pravdepodobnosti (3) a je dané dvoma parametrami: strednou hodnotou μ a rozptylom σ^2. Nasledujúci obrázok ukazuje aký vplyv majú tieto hodnoty na výsledné rozdelenie.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{3}$$

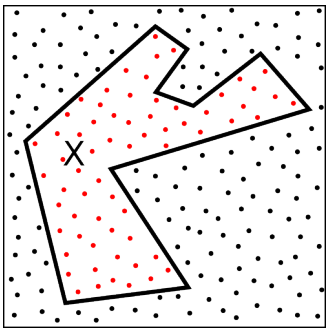


Obrázok 1: Hustota normálneho rozdelenia s rôznymi parametrami

2.2.2 Integračná metóda Monte Carlo

Obecne je Monte Carlo stochastická metóda využívajúca náhodné alebo pseudonáhodné čísla pre výpočet integrálov, väčšinou viacrozmerných, pri ktorých sú ostatné metódy výpočtov neefektívne. Riešenie spočíva vo vykonaní mnohých pokusoch (meraniach), z ktorých výsledky štatisticky spracujeme. Postup je taký, že pôvodný model úlohy prevedieme na pravdepodobnostný, vykonáme dostatočný počet pokusov, ktoré nám po spracovaní dajú výsledok, Kedže nepočítame s pôvodným modelom, ale s pravdepodobnostným, výsledok má pravdepodobnostný charakter, a jeho presnosť závisí od počtu pokusov.

Základnú myšlienku môžeme ukázať na príklade geometrickej úlohy určenia obsahu plochy útvaru X ležiaceho v jednotkovom štvorci. Útvar X môže mať ľubovolný tvar, prípadne môže byť zložený z viacerých segmentov. Úlohu znázorňuje obrázok:



Obrázok 2: Monte Carlo - určenie plochy náhodnými pokusmi

Postup výpočtu obsahu útvaru spočíva, že v ohraničujúcom štvorci si zvolíme N náhodných bodov rovnomerne rozložených a zisťujeme koľko z nich padlo do útvaru X . Tento počet označme ako m . Obsah plochy nášho útvaru je potom rovný približne pomeru m/N . Presnosť vypočítaného výsledku veľmi závisí na rovnomernom rozložení bodov v štvorci, a tiež čím viac bodov použijeme, tým je výsledok presnejší.

Obecný postup výpočtu je nasledovný:

- generovanie náhodných rovnomerne rozložených čísel x_i
- pre body x_i spočítame odhady charakteristík
- výsledky štatisticky spracujeme

Integrál môžeme touto metódou vypočítať tak, že určíme hodnoty funkcie $f(x)$ v náhodných bodoch, ležiacich v integračnej oblasti V , potom platí vzťah:

$$\int_V f(x)\,dV \approx V f(x) \pm \frac{\sigma V}{\sqrt{n}} \tag{4}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (\bar{y} - y_i)^2}{n}} \tag{5}$$

kde x označuje náhodný bod v oblasti V , $\bar{y}$ označuje priemer funkčných hodnôt a y_i jednotlivé výsledky. Ďalej oblasť V uzatvoríme do čo najmenšej oblasti V' , zavedieme si funkciu $\bar{f}$ takto:

$$\bar{f}(x) = \begin{Bmatrix} 0 & x \notin V \\ f(x) & x \in V \end{Bmatrix} \tag{6}$$

Potom vygenerujeme N náhodných bodov ležiacich v oblasti $\bar{V}$, približnú hodnotu integrálu určíme nasledovne:

$$I \approx \frac{\bar{V}}{N} \sum_{i=1}^{N} \bar{f}(x) \tag{7}$$

K dosiahnutiu prijateľnej presnosti je potrebné mnohonásobne opakovanie pokusov (meraní), Súčasne s odhadom neznámej hodnoty je tiež dôležité určenie presnosti odhadu. K odhadu chyby výsledku sa väčšinou používa stredná kvadratická chyba aritmetického priemeru. Chyba výsledku získaného z N pokusov je úmerná $1/\sqrt{N}$, takže aby sme zlepšili presnosť výsledku o jeden rád, budeme musieť zvýšiť počet pokusov aspoň o dva rády.

2.2.3 Monte Carlo v počítačovej grafike

V počítačovej grafike často krát zanedbávame integrálne výpočty a nahrádzame ich jednoduchými aproximáciami, ako napríklad v metóde sledovania lúča, kde farbu jedného pixlu obrazu určuje priesečník jedného nekonečne tenkého lúča s jedným objektom scény.

Problém je v tom, že pixel má určitú plochu (nieje bodom), a preto správnu hodnotu jeho intenzity (analogicky farbu) určuje súčet intenzít $I(x, y)$ všetkých svetelných lúčov dopadajúcúcich na plochu pixlu (pre čiernobiely obraz, pre farebný obraz by sme tento model rozšírili z jednej na tri farebné zložky RGB). Matematický zápis je nasledovný, kde intenzita svetla dopadajúceho na obdĺžnikovú plochu pixlu je :

$$intenzita(x, y) = \int_{x-0.5}^{x+0.5} \int_{y-0.5}^{y+0.5} I(x, y)\, dy\, dx \tag{8}$$

Pričom funkcia $I(x, y)$ vyjadruje intenzitu dopadajúcu do bodu (x, y) . Tento výpočet môžeme aproximovať metódou Monte Carlo:

$$intenzita(x, y) \approx \frac{1}{N} \sum_{i}^{N} I(x_i, y_i) \tag{9}$$

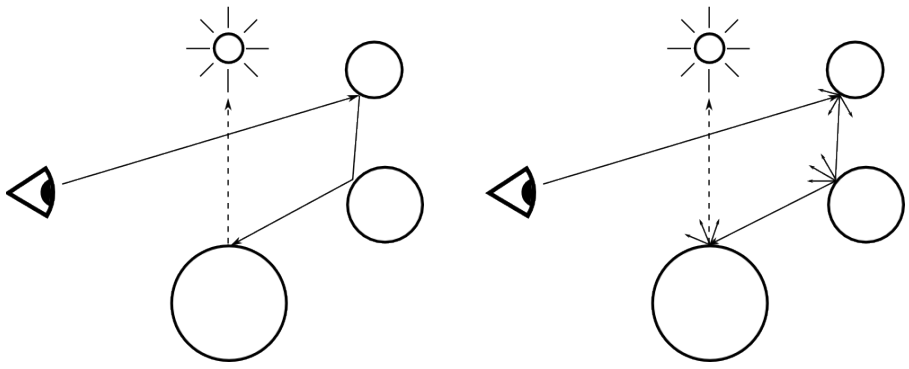
Hodnoty x_i a y_i predpokladajú štvorcovú plochu pixlu, sú z rozsahu $x \in (x-0.5, x+0.5)$, resp. $y \in (y-0.5, y+0.5)$, a musia byť rovnomerne rozložené po ploche pixlu. Chyba výpočtu pri tejto aproximácii klesá s rastúcim množstvom pokusov (lúčov). Táto aproximácia odstraňuje

nevzhľadné artefakty známe ako aliasing a tým skvalitňuje výsledný obraz. Rovnakým spôsobom môžeme postupovať pri aproximovaní reflexných, tieňových a refrakčných lúčov. Túto myšlienku nahradenia jedného lúča zväzkom (aproximáciu integrálu) lúčov realizuje distribuovaný raytracer.

2.3 Distribuovaný raytracing

Distribuovaný, nazývaný tiež stochastický je vylepšený raytracing, umožňujúci zobrazenie javov, nemysliteľných pre konvenčný raytracing. Tradičný raytracing je obmedzený použitím, len jediného lúča pre výpočet rôznych javov. Napríklad keď počíta farba bodu na povrchu telesa, raytracing vyšle z tohto bodu lúče k svetelným zdrojom, jeden ku každému svetlu, čím zisťuje či je bod osvetlený, alebo nieje. Tento princíp vedie k nerealisticky ostrým tieňom, a nepripúšťa možnosť hladkých tieňov, keď objekty len čiastočne prekrývajú svetelné zdroje (inak povedané, svetelné zdroje majú nulovú plochu). Ďalším takýmto nedostatkom sú napríklad ostré hrany objektov.

Distribuovaný raytracing prekonáva tieto obmedzenia raytracingu použitím zväzku rozptýlených lúčov (jeden lúč sa nahradí zväzkom lúčov). Výmenou jedného tieňového lúča za zväzok lúčov, dokážeme produkovať pekné mäkké tiene, čo predtým nešlo. Obdobne skupina reflexných lúčov produkuje rozmazané odrazy. Pridanie tzv. mäkkých tieňov, rozmazaných odrazov do metódy sledovania lúča výrazne zlepšuje reálnosť produkovaných obrázkov, pretože ostré tiene v reálnom svete takmer nevidieť. Distribuovaný raytracing pritom do algoritmu raytraceru nepridáva žiadny nový typ lúča.



Obrázok 2: Konvenčný raytracing (vľavo) a distribuovaný (vpravo)

Okrem spomínaných skrášlení odrazov a tieňov dokáže distribúcia lúčov aj pokročilejšie efekty. Distribúcia lúčov do časovej domény produkuje v animovanej scéne efekt rozmazaného pohybu tzv. „motion blur“, distribúcia lúčov v projekčnej rovine dokáže simulovať efekt známy

ako hĺbka ostrosti tzv. „depth of field“. Cenou za vyššiu kvalitu výsledku je znásobenie celkového počtu lúčov a teda aj času výpočtu v závislosti na počte použitých efektov a množstve distribuovaných lúčov.

Maximálnu teoretickú náročnosť metódy sledovania lúča môžeme zapísať vzorcom:

$$S_c = \sum_{i=0}^{R} [S_p 2^R (1 + L)] \tag{10}$$

S_p - počet primárnych lúčov $S_p = W \times H$

S_c - celkový počet lúčov

W - šírka obrazu

H - výška obrazu

L - počet svetelných zdrojov

R - hĺbka zanorenia, 0 znamená použitie iba primárnych lúčov

Pre distribuovaný raytracing musíme vzorec č.10 rozšíriť o distribúciu lúčov:

$$S_D = \sum_{i=0}^{R} [S_{time} S_{dof} S_{aa} S_p (N_{reflected} + N_{refracted})^R (1 + N_{shadow} L)] \tag{11}$$

S_D - celkový počet lúčov

N_{aa} - počet lúčov pre antialiasing

N_{time} - počet lúčov pre „motion blur“ , pričom musí byť aspoň 1

N_{dof} - počet lúčov pre „depth of field“, pričom musí byť aspoň 1

$N_{reflected}$ - počet reflexných lúčov

$N_{refracted}$ - počet refrakčných lúčov

N_{shadow} - počet tieňových lúčov, pričom musí byť aspoň 1

Ak by sme do rovnice č.11 dosadili minimálne hodnoty za distribučné konštanty, získame rovnicu raytracingu č.10. Z porovnania predchádzajúcich rovníc je jasne vidieť veľkú cenu distribuovaných efektov, lepšiu predstavu poskytuje nasledujúca tabuľka.

	iba raytracing	motion blur	mäkké tiene	mäkké tiene + hĺbka poľa	mäkké tiene + hĺbka poľa + antialiasing
N_{TIME}	1	1	32	1	1
N_{DOF}	1	1	1	16	16
$N_{REFLECTED}$	1	1	1	1	1
$N_{REFRACTED}$	1	1	1	1	1
N_{SHADOW}	1	12	2	12	12
N_{AA}	1	1	1	1	8
počet lúčov	$13{,}8 \cdot 10^6$	$11{,}5 \cdot 10^7$	$73{,}7 \cdot 10^7$	$184{,}3 \cdot 10^7$	$14{,}7 \cdot 10^9$
nárast lúčov	0x	8x	53x	133x	1065x

Výsledky boli počítané pre rozlíšenie 640x480, scénu s dvoma svetlami a hĺbkou rekurzie 3. Ako vidieť distribuovaný raytracing je veľmi náročný na výpočetný výkon, napr. pri použití mäkkých tieňov s počtom lúčov 12 je približne 8 krát pomalší ako klasický raytracing. Hodnoty v tabuľke slúžia len na vytvorenie predstavy o náročnosti distribuovaného raytracingu, pretože modelujú teoretický počet lúčov pre scénu, v ktorej je každý objekt reflexný, a priehľadný. V praxi je vhodné sa takýmto scénam vyhýbať a používať striedmo tieto pokročilé efekty.

Výhody distribuovaného raytracingu proti klasickému:

- antialiasing
- mäkké tiene
- rozmazané odrazy
- „motion blur“
- reálna priehľadnosť

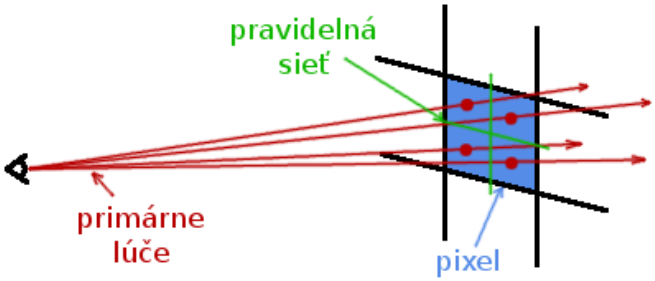
Nevýhody distribuovaného raytracingu proti klasickému:

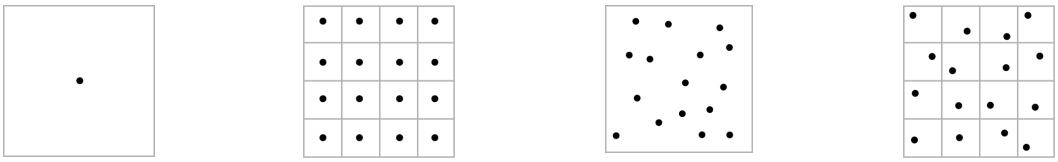
- omnoho väčšia náročnosť na výpočetný výkon

2.3.1 Antialiasing

Konvenčný raytracing z kamery vysiela do scény jeden lúč na jeden pixel obrazu, riešením rovnice tohto jediného lúča je farba pixlu a tá je daná osvetľovacím modelom najbližšieho priesečníka lúča s objektami v scéne. Tieto primárne lúče sú nekonečne tenké a vysielané cez stredy obrazových pixlov. Výsledná farba každého pixlu je daná nekonečne malým priesečníkom primárneho lúča s objektami v scéne. Nevýhodou tohto prístupu je, že plocha pixlu je veľká vzhľadom na veľkosť lúča, čo sa vo výsledku prejaví ako nežiadúci jav nazývaný alias (napr. zubaté hrany objektov) . Dalo by sa povedať, že scéna je vzorkovaná s nízkou frekvenciou. Distribuovaním primárnych lúčov cez plochu pixlu je možné potlačiť vplyv nežiadúceho aliasu.

Distribúcia zvyšuje množstvo vysielaných primárnych lúčov, z každého pixlu posiela zväzok lúčov, a farba pixlu je potom priemerom tohto zväzku. Rozdelenie lúčov v zväzku môže byť náhodné, pravidelnou sieťou, náhodné v pravidelnej sieti, ...





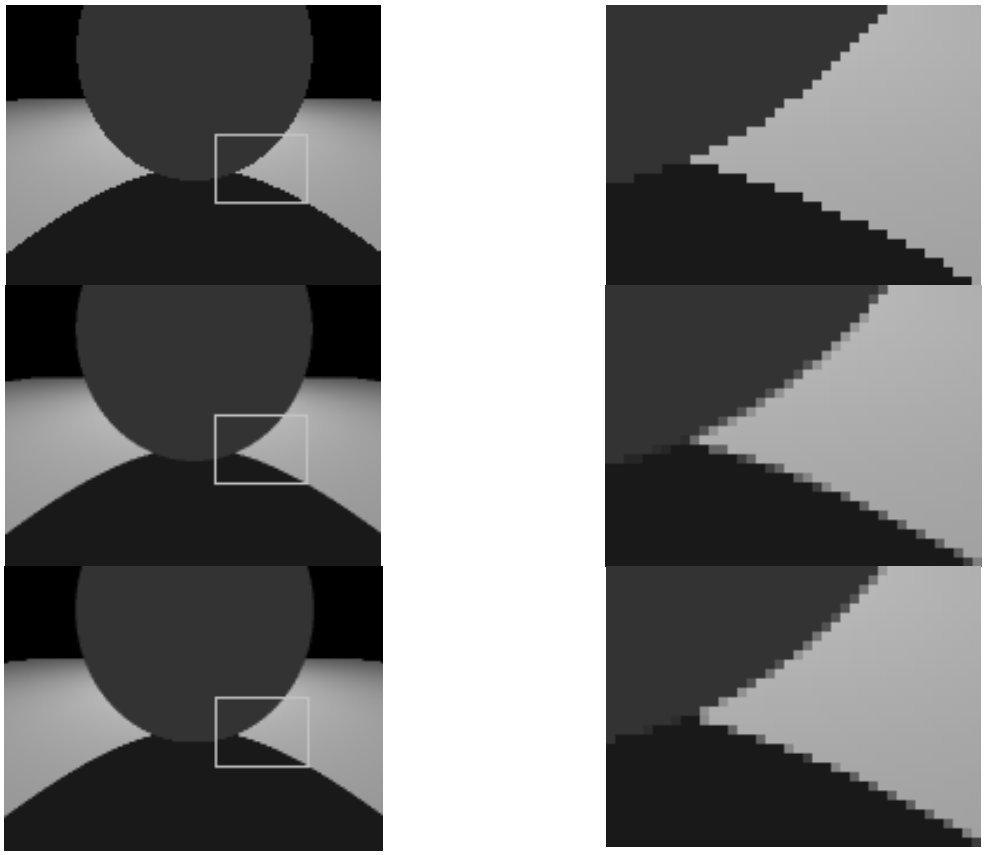
Obrázok 3: Rôzne stratégie vzorkovania (z ľava): raytracing, pravideľná mrižka, rovnomerne rozptýlené, náhodné v rámci pravidelnej siete

Zvýšenie počtu primárnych lúčov zo sebou prináša značné spomalenie, čím viac lúčov použijeme, tým viac času spotrebuje výpočet. Dobrou voľbou sú siete rozmerov 3x3, príp. 4x4, ktoré spomalia výpočet približne 9, resp. 16krát.

a)

b)

c)



Obrázok 4: Vplyv počtu lúčov na kvalitu: a) raytracing 1 lúč na pixel b) pravidelná mriežka 2x2 c) pravidelná mriežka 4x4

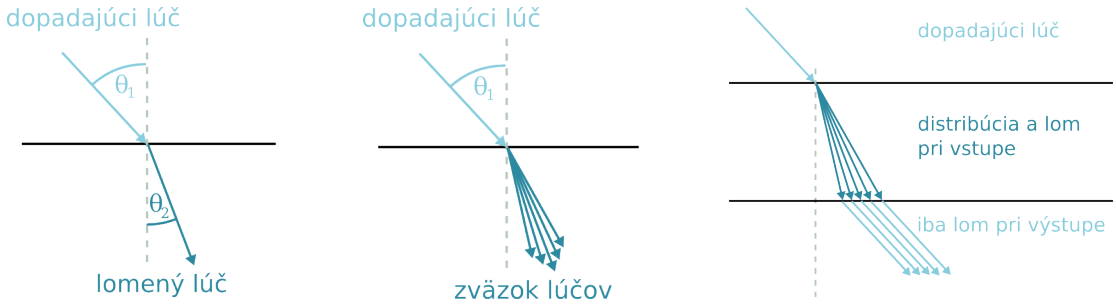
Alias je najviditeľnejší na hranách kontrastných objektov, pre zachovanie dobrého výkonu a prijateľnej kvality je možné obmedziť distribúciu lúčov len do týchto „kritických“ miest. Táto technika by mohla byť implementovaná prostredníctvom dodatočnej pamäte udržujúcej pre každý pixel ukazovateľ na objekt v scéne. Algoritmus by potom porovnal hodnotu ukazovateľa aktuálneho pixlu so svojím okolím a rozhodol by či použiť distribúciu alebo nie. Tento algoritmus je vhodný pre scény bez reflexných povrchov, pretože tie nevyhladí, prípadne je nutné ho rozšíriť o sledovanie reflexných lúčov.

2.3.2 Matná priehľadnosť

Refrakčné lúče v raytracingu sa počítajú zákonom lomu, ktorý hovorí: ak lúč dopadá z prostredia s indexom lomu n_1 pod uhlom θ_1 na povrch materiálu s indexom lomu n_2 , láme sa pod uhlom θ_2 (rovnica č.12).

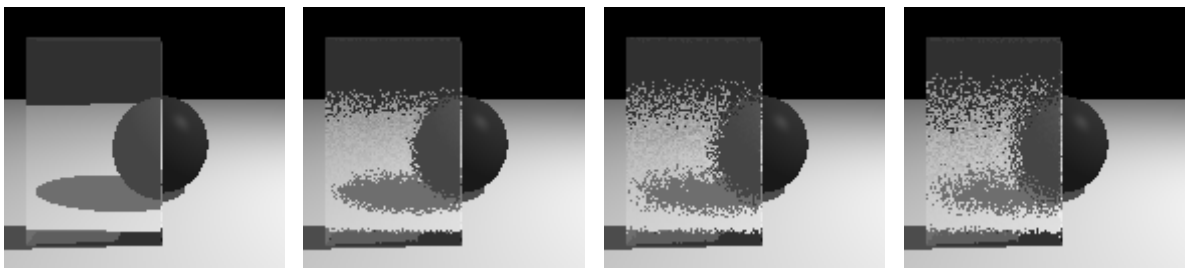
$$\sin(\theta_2)=\sin(\theta_1)\frac{n_1}{n_2} \tag{12}$$

Jeden lúč dopadajúci na priehľadný materiál generuje jeden refrakčný lúč podľa rovnice č.12, takýmto prístupom získame dokonale ostrú priehľadnosť. Naproti tomu ak chceme simulovať matne priehľadný materiál môžeme namiesto jedného refrakčného lúča generovať zväzok refrakčných lúčov. Rozptýlenie prevádzame pri vstupe do telesa, vystupujúci lúč nieje distribuovaný, pričom rozptýlenie lúčov je rovnomerné do blízkeho okolia osi hlavného lúča lomu (distribuované vhodnou distribučnou funkciou).



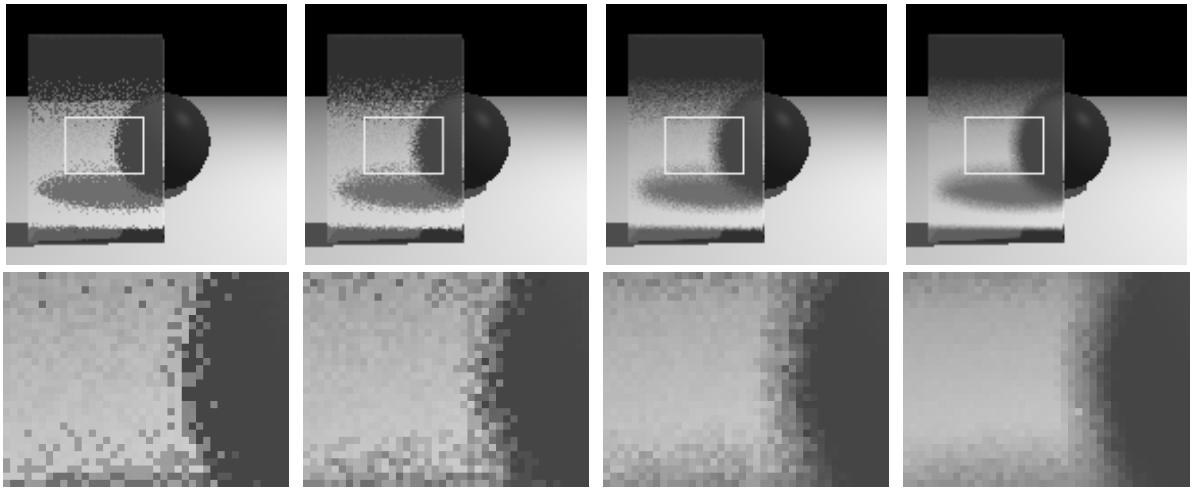
Obrázok 5: Lom lúča a jeho distribúcia, vľavo raytracing, v strede a napravo distribuovaný raytracing

Matnosť materiálu získaného distribúciou lúčov závisí od nastavenia parametrov veľkosti rozptylu („dispersion“) a počtu distribuovaných lúčov („samples“), čo zobrazuje nasledujúca séria obrázkov.



a) raytracing *b) dispersion 0.1* *c) dispersion 0.15* *d) dispersion 0.2*

Obrázok 6: Rôzny rozptyl refrakčných lúčov



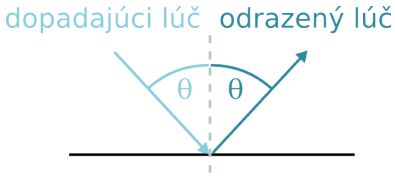
a) 2 lúče *b) 4 lúče* *c) 16 lúčov* d) 64 lúčov

Obrázok 7: Vplyv počtu lúčov na kvalitu (dispersion 0.15)

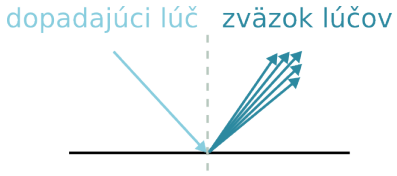
Stupeň matnosti materiálu sa reguluje rozptýlením lúčov, od skoro veľmi nepatrnej matnosti až po husto matne priehľadný materiál. Jemnosť (zrnitosť) je daná počtom rozptýlených lúčov, s narastajúcim množstvom je výsledok kvalitnejší, ale pomalší.

2.3.3 Rozostrené odrazy

Odrazy v raytracingu vytvárajú reflexné lúče, ich uhol je daný zákonom odrazu, podľa ktorého sa uhol odrazu rovná uhlu dopadu.

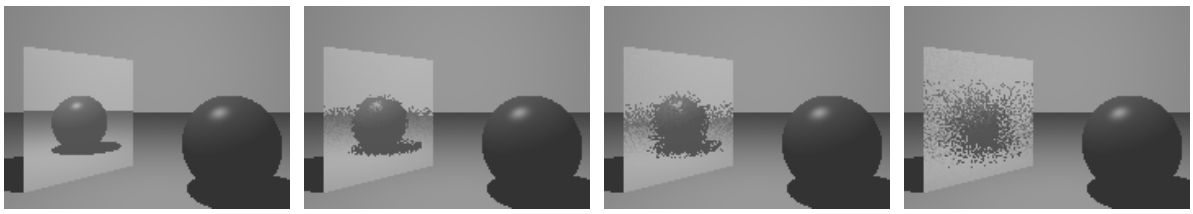


Obrázok 8: Reflexia v raytracingu



Obrázok 9: Distribuovaná reflexia

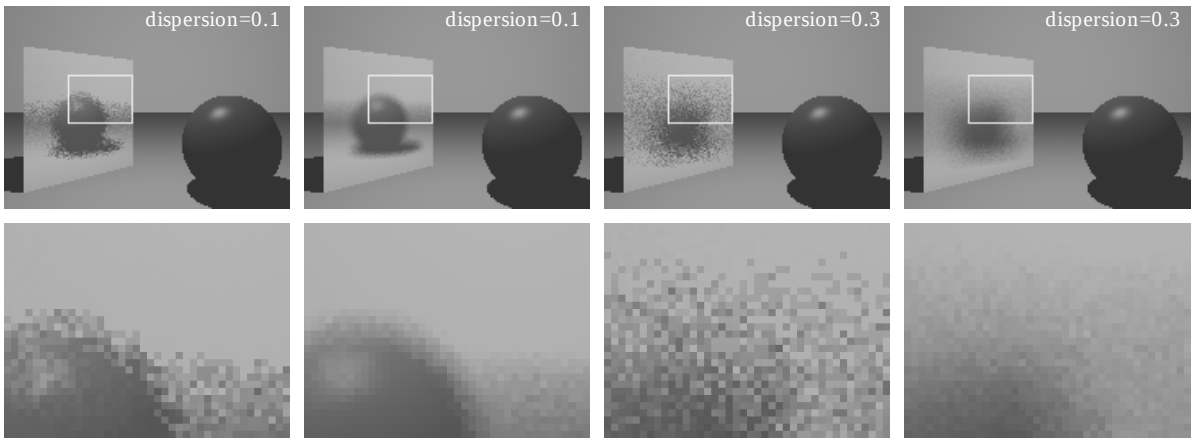
Nahradenie jedného odrazového lúča väčším množstvom lúčov dosiahneme jemne rozmazané odrazy, ktoré vyzerajú vizuálne realistickejšie oproti klasickým matematicky presným odrazom. Minimálny počet odrazených lúčov by mal byť aspoň 8, aby ich priemer dával primerané výsledky, pričom by mali smerovať do blízkeho okolia pôvodného lúča. Čím väčší rozptyl zvolíme, tým by mal byť aj počet lúčov väčší, pretože vzorkujeme väčšiu oblasť a z nej počítame priemernú hodnotu.



a) dispersion 0 *b) dispersion 0.05* *c) dispersion 0.1* *d) dispersion 0.3*

Obrázok 10: Rôzny rozptyl lúčov

Predchádzajúce obrázky prezentujú rôznu veľkosť rozmazaného odrazu, ktorá sa nastavuje parametrom „dispersion“, s jeho rastúcou hodnotou rastie rozptyl lúčov. Pre dosiahnutie kvalitných odrazov je nutný aj dostatočný počet použitých lúčov (Obrázok 11).

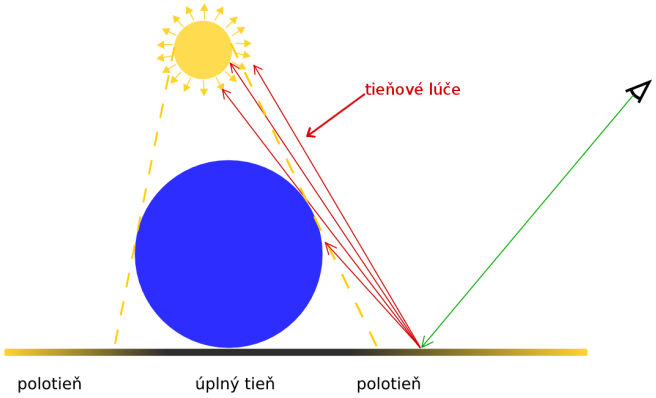


a) 4 lúče b) 64 lúčov c) 4 lúče d) 64 lúčov

Obrázok 11: Vplyv počtu lúčov na kvalitu lesklého materiálu

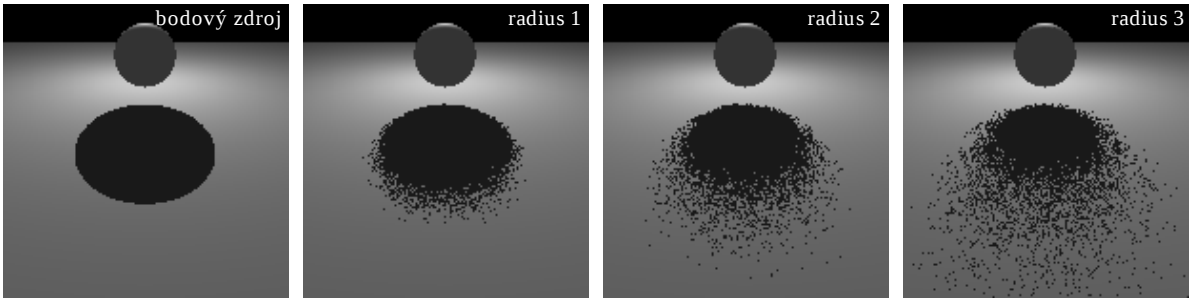
2.3.4 Mäkké tiene

Ako bolo v úvode naznačené, možnosti distribuovania lúčov sú aplikovateľné v mnohých situáciách. Ďalšou aplikáciou je výpočet tieňových lúčov, ktoré počas cesty z povrchu objektu ku svetelným zdrojom hľadajú prekážky, objekt je osvetlený svetlom len v tedy ak medzi ním a svetlom nieje prekážka. Jeden tieňový lúč určuje či je bod na povrchu telesa osvetlený, alebo nesvetlený z daným svetelným zdrojom, vo výsledku takýto prístup vytvára ostré tiene. V skutočnosti má väčšina tieňov mäkké okraje a pretože zdroje svetla majú plošný charakter (nie bodový). Simulovanie plošných zdrojov svetla a dosiahnutie mäkkých tieňov dosiahneme nahradením bodových zdrojov guľou (príp. plochou, kvádrom, ...) a tieňového lúča zväzkom lúčov. Nevýhodou je opäť vyššia náročnosť na výpočetný výkon, preto treba zvážiť počet lúčov. Z pokusov sa overila hodnota minimálne 9, pričom ak použijeme techniku na vyhladenie obrazu (antialiasing) je možné toto číslo znížiť.

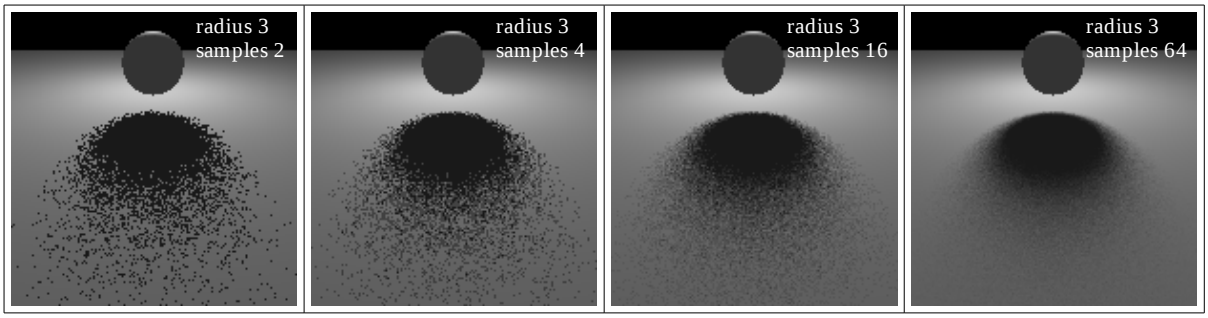


Obrázok 12: Mäkké tiene

Veľkosť polotieňa určuje svetelný zdroj svojou veľkosťou. Okrem veľkosti polotieňa je dôležité nastaviť počet tieňových lúčov na jeden zdroj svetla („sampless“). Rôzne hodnoty týchto parametrov ukazuje nasledujúca séria obrázkov.



Obrázok 13: Vplyv veľkosti zdroja (guľový zdroj) na veľkosť polotieňa

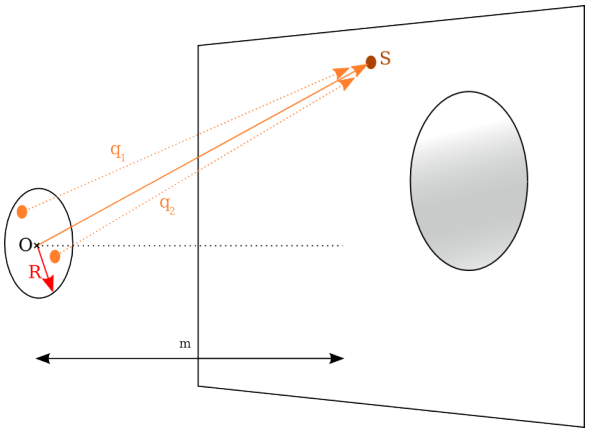


Obrázok 14: Vplyv počtu lúčov na kvalitu (gulový zdroj svetla)

Kvalitné výsledky zabezpečí veľké množstvo tieňových lúčov vyslaných k zdrojom svetla, nevýhodou je že čím viac lúčov použijeme, tým je výpočet náročnejší, približne platí ak použijeme *n* lúčov namiesto jedného, výpočet bude *n*-krát pomalší.

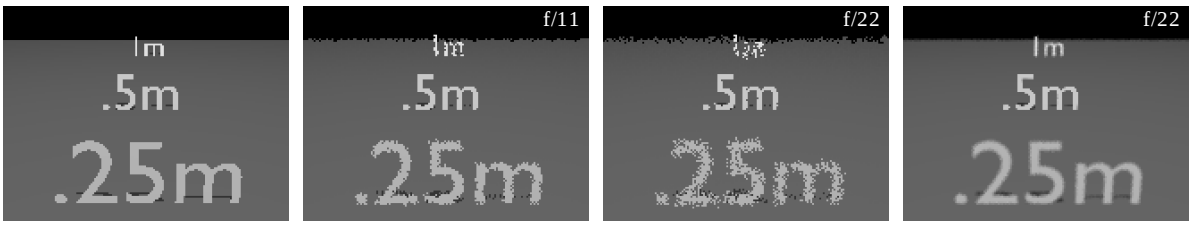
2.3.5 Hĺbka ostrosti „depth of field“

Ďalším z rady efektov je efekt hĺbky ostrosti, kde na výslednom obraze sú niektoré objekty ostré, a niektoré rozmazané, veľkosť rozmazania závisí na vzdialenosti objektu voči kamere. Hĺbka ostrosti je priestor pred a za rovinou na ktorú je zaostrené a na ktorej sú všetky predmety zobrazené ostré. V raytracingu používame jednoduchý model kamery, kde všetky lúče vychádzajú z rovnakého počiatku („pinhole camera“), tento model vytvára obrázky kde všetky objekty v scéne sú ostré. Na simuláciu efektu hĺbky ostrosti je nutné generovať lúče napodobňujúc prechod šošovkou objektívu, čo znamená generovať lúče s rôznym počiatkom.



Obrázok 15: Generovanie lúčov hĺbky ostrosti

Primárne lúče sú generované z bodu O, pre dosiahnutia hĺbky ostrosti budeme musieť pre jeden pixel obrazu generovať zväzok lúčov (q_1, q_2, ... , q_n) s náhodným počiatkom v kruhu rovnobežnom s projekčnou rovinou a stredom v bode O. Polomer tohto kruhu a tým aj veľkosť clony) udáva parameter R. V praxi je zavedené používať označenie prostredníctvom clonového čísla $N = f/(2R)$, zapisovaného ako *f/11*, *f/22*, ... Pre veľkosť clonového čísla platí, že jeho zníženie (otvorenie clony) znižuje hĺbku ostrosti, a naopak. Ďalším parametrom je nastavenie vzdialenosti m, v ktorej budú všetky vykresľované objekty ostré.

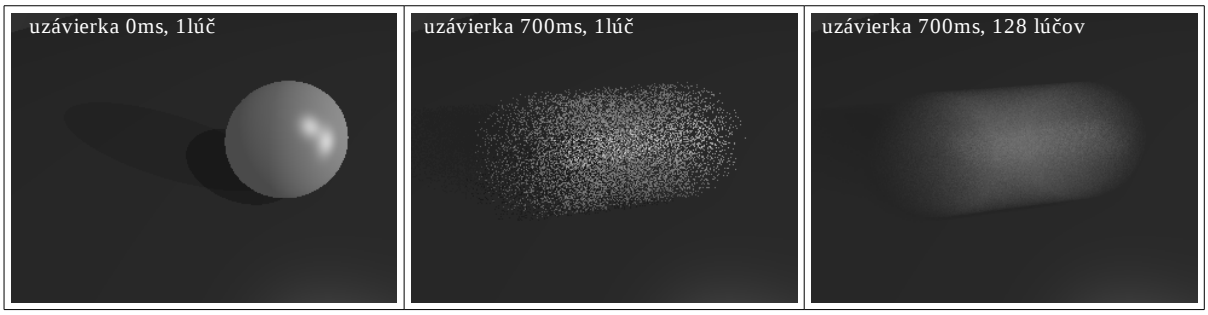


Obrázok 16: Vplyv veľkosti clony na hĺbku ostrosti, kamera je zaostrená na vzdialenosť 0.5m (prvé tri obrázky sú renderované s jedným lúčom na pixel, posledný obrázok s 32 lúčmi)

2.3.6 Rozmazanie pohybom „motion blur“

Posledným efektom je efekt spôsobujúci rozmazanie pohybujúcich sa objektov. Tento efekt vychádza zo skutočných vlastností kamery a fotoaparátu. Základom fotoaparátu je uzavretá svetlotesná komora s otvorom vybaveným šošovkami (objektívom). V momente, keď je stlačená spúšť, uzávierka sa na určitý čas otvorí a umožní svetlu vniknúť do vnútra komory. V jej vnútri sa nachádza svetlocitlivá vrstva (film, CCD, ...), na ktorú svetlo kreslí obraz. Pokiaľ sa stane, že sa počas otvorenej uzávierky pohne nejaký objekt, tento pohyb sa zaznamená do obrazu v podobne rozmazania.

Na získanie podobného efektu je potrebná distribúcia primárnych lúčov do časového intervalu (časového okna), počas ktorého je otvorená uzávierka fotoaparátu. Takže jeden primárny lúč nahradíme viacerými, a tie budeme distribuovať v čase. Z hodnôt všetkých lúčov potom vypočítame aritmetický priemer (samotnému aritmetickému priemeru môže predchádzať váhovanie hodnôt, napríklad pre simulovanie množstva dopadajúceho svetla na svetlocitlivú vrstvu).



Obrázok 17: Vplyv počtu lúčov a dĺžky uzávierky na kvalitu

Pre distribúciu lúčov v čase je možné použiť rôzne techniky. Napríklad budeme do časového okna vysielať lúče rovnomerne rozložené v čase a ich výsledky váhovať podľa veľkosti otvorenia clony. Druhým prístupom je vytvorenie distribučnej funkcie simulujúcej chovanie reálnej clony, touto funkciu bude potom dané rozloženie lúčov v čase. Nakoniec sa zo všetkých lúčov vypočíta priemer. Táto technika je implementovaná v demonštračnom programe.

3 Rýchlosť a kvalita

Distribuovaný raytracing je často používaná metóda vizualizácie trojrozmerných modelov, pretože poskytuje kvalitné výsledky. Jej slabým miestom je náročnosť výpočtov, ktorá vychádza z množstva použitých lúčov a množstva počítaných priesečníkov. Toto sú jej kritické miesta, a pri urýchľovaní renderovania by sme sa mali na ne zamerať. Počet priesečníkov môžme efektívne redukovať použitím techník inteligentne deliacich priestor scény na menšie časti. Rozumným riešením je použitie kd-stromu, ktorému sa hlavne venuje úvodná podkapitola. Ďalšie časti tejto kapitoly popisujú metódy znižovania počtu lúčov prostredníctvom heuristických techník.

3.1 Distribúcia lúčov

Táto podkapitola popisuje distribúciu lúčov v distribuovanom raytraceri, presne tak ako je implementovaná v demonštračnej aplikácii.

3.1.1 Distribúcia tieňových lúčov

Tieňové lúče sú distribuované rovnomerne po celej ploche plošného zdroja svetla. Demonštračný raytracer implementuje guľový zdroj svetla, takže je potrebné generovanie bodov rovnomerne rozdelených na povrchu gule. Toto je možné zjednodušiť generovaním bodov v kruhu so stredom v strede zdroja svetla. Postup generovania jedného tieňového lúča T je nasledovný:

- výpočet vektoru $\vec{P}$, smerujúceho z bodu O na povrchu telesa do stredu C guľového zdroja svetla
- výpočet lokálnych ôs $\vec{L_x}$ a $\vec{L_y}$ kolmých na $\vec{P}$ a na seba navzájom
- normalizácia lokálnych ôs
- adaptácia lokálnych osí na polomer kružnice
- generovanie náhodných hodnôt $\alpha \in \langle 0, 360 \rangle$ a $r \in \langle 0,1 \rangle$, $d = \sqrt{r}$
- výpočet $u = d\cos(\alpha)$ a $v = d\sin(\alpha)$
- výpočet smeru tieňového lúča $\vec{T_{dir}} = \vec{P} + u\vec{L_x} + v\vec{L_y}$
- vektor $\vec{T_{dir}}$ je vhodné normalizovať
- výsledný tieňový lúč $T = (O, \vec{T_{dir}})$

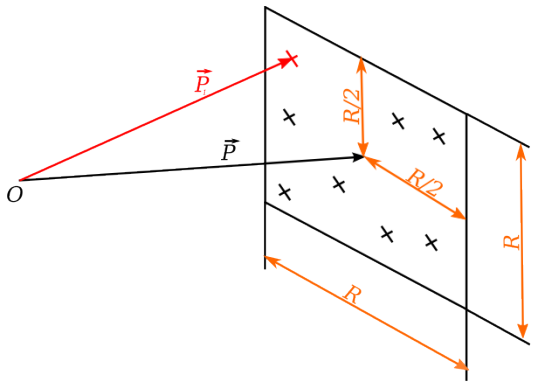
Takto generované tieňové lúče majú rovnomerné rozdelenie v kruhovej ploche nahrádzajúcej pôvodný guľový zdroj svetla.

Generovanie tieňových lúčov pre obdĺžnikový zdroj je podobné, obdĺžnik je zadaný vrcholmi A, B, C, D . Postup generovania je nasledovný:

- výpočet lokálnych osí $\vec{L_x} = B - A$ a $\vec{L_y} = D - A$
- generovanie náhodných hodnôt $\alpha \in \langle 0,1 \rangle$ a $\beta \in \langle 0,1 \rangle$
- výpočet smeru tieňového lúča $\vec{T_{dir}} = A - O + \alpha \vec{L_x} + \beta \vec{L_y}$
- vektor $\vec{T_{dir}}$ je vhodné normalizovať
- výsledný tieňový lúč $T = (O, \vec{T_{dir}})$

3.1.2 Distribúcia reflexných a refrakčných lúčov

Distribuované reflexné a refrakčné lúče používajú rovnakú techniku distribúcie. Pri distribuovaní týchto lúčov chceme simulovať nerovný povrch, preto musíme lúče rozptýliť do okolia pôvodného smeru lúča. Veľkosť rozptylu lúčov reguluje veľkosť nerovností na povrchu materiálu. Tento rozptyl môžeme aproximovať rovnomerným rozptylom lúčov do štvorcovej plochy, kolmej na pôvodný lúč. Veľkosť tejto plochy stanovuje veľkosť nerovností povrchu.



Obrázok 18: Generovanie refrakčných lúčov

O je počiatok refrakčného príp. reflexného lúča, $\vec{P}$ je normalizovaný vektor smeru pôvodného lúča, P_1 je náhodne generovaný lúč, R určuje veľkosť rozptylu. Postup generovania je nasledovný:

– určenie lokálnych osí L_x a L_y , kolmých na vektor $\vec{P}$ a na seba navzájom:

$min(\vec{P_x}, \vec{P_y}, \vec{P_z}) \equiv P_x \quad \vec{N_d} = (1,0,0)$

$min(\vec{P_x}, \vec{P_y}, \vec{P_z}) \equiv P_x \quad \vec{N_d} = (1,0,0)$

$min(\vec{P_x}, \vec{P_y}, \vec{P_z}) \equiv P_x \quad \vec{N_d} = (1,0,0)$

$\vec{L_x} = R(\vec{P} \times \vec{N_d})$

$\vec{L_y} = R(\vec{L_x} \times \vec{N_d})$

– generovanie dvoch náhodných hodnôt $x, y \in \langle -0.5, 0.5 \rangle$

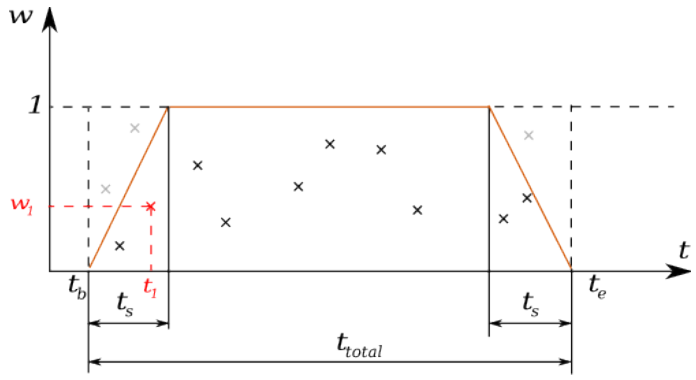
– výpočet smeru lúča $\vec{P_1} = \vec{P} + x L_x + y L_y$

– nový lúč je potom určený počiatkom O a normalizovaným vektorom $\vec{P_1}$

Takto vytvorený rozptyl lúčov je rovnomerne rozptýlený do štvorcovej oblasti aproximujúci rozptyl lúčov na nerovnom povrchu. Existujú aj iné riešenia, napríklad aproximácia kruhovou plochou, namiesto štvorcovej, alebo aj váhovanie podľa uhla odklonu od pôvodného smeru.

3.1.3 Distribúcia lúčov pre „motion blur“

Pri tomto efekte namiesto generovania hodnôt v priestore potrebujeme generovať časy, v ktorých budeme snímať scénu. Pre snímanie scény si vyrobíme závierku v tvare štvoruholníka, ktorá bude simulovať chovanie skutočnej závierky. Pravdepodobnosť, že scéna bude vykreslená pri otváraní alebo zatváraní závierky je menšia ako pravdepodobnosť, že scéna bude vykreslená pri plne otvorenej závierke. Pri výpočte sa rovnomerne náhodne zvolí bod z obdĺžniku t_b a t_e a zisťuje sa či leží pod krivkou závierky, ak áno prevedie sa výpočet v tomto čase. Ak neleží, potom musíme zvoliť nový bod a výpočet opakovať.



Obrázok 19: Distribučná funkcia pre "motion blur"

Horizontálna os t reprezentuje plynúci čas, vertikálna os w určuje váhu aktuálneho výsledku na celkový výsledok, t_{total} je čas závierky, t_s určuje čas otvárania aj čas uzatvárania závierky, t_b a t_e ohraničujú interval počas snímania scény, w_1 je náhodné číslo z intervalu $\langle 0,1 \rangle$. Celý výpočet pomocou závierky je nasledovný:

– vygenerujeme náhodné číslo t_1 z intervalu $\langle t_b, t_e \rangle$

– ak $t_1 \in \langle t_b + t_s, t_e - t_s \rangle$ vypočítame lúč v čase t_1 a jeho farbu pripočítame k celkovej farbe bodu, ďalej pokračujeme novým výpočtom

- inak vygenerujeme náhodné číslo $w_1 \in \langle 0,1 \rangle$
- ak w_1 je pod krivkou závierky, vypočítame farbu pixlu v čase t_1 , a pripočítame ju k farbe pixlu v danom bode
- ak w_1 leží nad krivkou závierku pokračujeme novým výpočtom

3.1.4 Distribúcia lúčov pre hĺbku ostrosti

Pre tento efekt potrebujeme generovať počiatky distribuovaných lúčov v kruhu s polomerom R . Tento kruh reprezentuje šošovku kamery. Algoritmus vrhania primárnych lúčov generuje lúče vychádzajúce z bodu $(0,0,0)$ do projekčnej roviny obrazu a potom tieto lúče transformuje transformačnou maticou kamery. Tento postup zjednodušuje aritmetiku generovania primárnych lúčov.

Šošovku nahradíme kruhom so stredom $(0,0,0)$ ležiacim v rovine XY , čiže potrebujeme vypočítať náhodné hodnoty x a y rovnomerne rozdelené v kruhu s polomerom R . Postup je nasledovný:

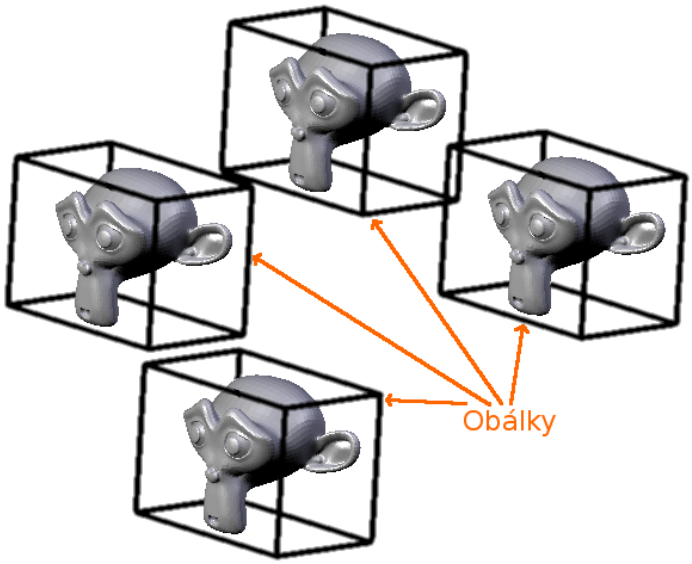
- generovanie náhodných hodnôt $\alpha \in \langle 0,360 \rangle$ a $r \in \langle 0,1 \rangle$, $d = \sqrt{r}$
- výpočet bodov ležiacich na šošovke $x = d \sin(\alpha)$ a $y = d \cos(\alpha)$

3.2 Hierarchické delenie priestoru, kd-strom

Pre vykreslenie celej scény je potrebné vyslať do scény mnoho lúčov, pre ktoré potom hľadáme najbližšie priesečníky. Nájdenie najbližšieho priesečníka lúča vyslaného do scény je časovo kritická operácia. Najjednoduchším riešením je vypočítať priesečníky lúča so všetkými objektami scény a z nich vybrať najbližší. Tento algoritmus je jednoduchý, nepotrebuje žiaden prípravný krok (predpočítanie scény), a celá implementácia je na pár riadkov. No hneď pri jeho prvom použití narazíme na jeho časovú náročnosť, pretože prechádza všetkými objektami v scéne. Jeho zložitosť lineárne rastie so zvyšujúcim sa počtom objektov scény. Ak má scéna napríklad štyri objekty, tak pre každé hľadanie najbližšieho objektu, sa zbytočne počítajú najmenej tri priesečníky. Celkovo je v takomto prípade zbytočne počítaných minimálne 75% priesečníkov. Ak by sme do scény pridali ďalších 996 objektov, tak toto číslo stúpne na 99.9%, pretože na jeden použitý priesečník sa počíta 999 prebytočných priesečníkov. Preto je potrebné redukovať veľký počet počítaných priesečníkov na čo najnižšiu hodnotu.

Pre redukciu nadbytočného počtu priesečníkov a tým pádom zrýchlenie celého raytraceru existuje viacero druhov prístupov. Najznámejšie z nich sú techniky používania hraničných obálok, a hierarchické delenie scény.

Hraničné obálky sú jednoduchým a pritom účinným nástrojom ako redukovať počet a hlavne čas potrebný pre výpočet priesečníka. Obálka má tvar matematicky jednoducho opísateľného telesa, najčastejšie kvádra, alebo gule, do ktorého sa obalí väčšie množstvo objektov. Postupne sa obálkami obalia všetky objekty v scéne. Obálky môžu byť v sebe vnorené, čo má význam pre zložitejšie scény. Pri hľadaní priesečníka v scéne, sa najprv otestuje križovanie lúča s obálkou. Ak lúč pretína obálku, potom nasleduje zanorenie do obálky a hľadanie priesečníka pokračuje v nej. V druhom prípade lúč obálku nepreťal, takže nepretína ani jeden v nej obsiahnutý objekt. Pre nájdenie najbližšieho priesečníka otestujeme všetky obálky. Druhou variantou použitia obálok sú objekty s náročným výpočtom priesečníka, tu je vhodné objekt obaliť čo najtesnejšou obálkou, a priesečník počítať až potom ako lúč pretne túto obálku. Keďže náš raytracer používa prevažne gule a trojuholníky, je lepšie zamerať sa na efektívnejšiu metódu.



Obrázok 20: Obálky obalujúce mnohopolygónové objekty

Hierarchicky rekurzívne delenie priestoru je technika delenia trojrozmerného priestoru na menšie podpriestory. Delenie priestoru na menšie časti rapídne urýchľuje prechod lúča scénou. Jednou z tohto druhu metód a veľmi efektívnou je BSP (binary space partitioning) a jeho najčastejšie používaná varianta kd-strom.

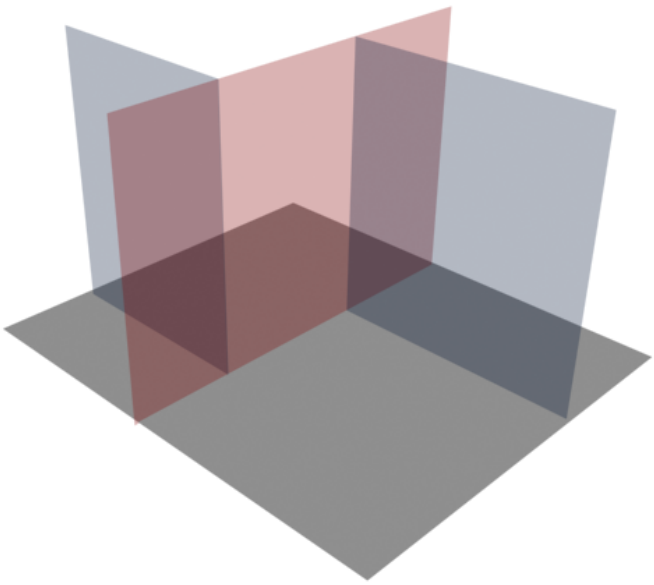
BSP strom je metóda delenia priestoru použiteľná pre množstvo geometrických úloh (problémov). Pôvodne bola vyvinutá pre riešenie viditeľnosti objektov v trojrozmernom priestore, a jej hlavnú štruktúru tvorí viacrozmerný binárny stromom, v ktorom je uložená celá scéna.
V počítačovej grafike sa používajú dve varianty BSP stromov, jedna varianta delí priestor plochami prechádzajúcimi polygónmi scény, druhá používa deliace plochy rovnobežné so súradnými osami.

V prvej variante BSP sa deliaca rovina určuje z rovín prechádzajúcich plochami polygónov v scéne. Väčšinou vyžaduje aby scéna bola zložená len s polygónov, čo je pre raytracing omädzujúce hľadisko a touto metódou sa ďalej nebudeme zaoberať.

Druhá varianta je kd-strom, ktorý je špeciálnym prípadom obecného BSP stromu. Podmienkou kd-stromu je, aby každá deliaca rovina bola rovnobežná s jednou z hlavných súradných osí (X, Y, Z). Detailnému popisu kd-stromu sa venuje nasledujúci text.

3.2.1 Kd-strom

Definícia kd-stromu je nasledovná: všetky uzly majú svoju hraničnú schránku (kváder) rovnobežnú so súradnými osami, túto schránku nazveme bunka. Bunka koreňového uzlu celého kd-stromu je osovo rovnobežný kváder, obsahujúci všetky objekty stromu. Každému internému uzlu je priradená deliaca rovina, rozdeľujúca bunku na dve menšie bunky. Každý listový uzol obsahuje zoznam objektov zasahujúcich do jeho bunky, pričom nemusí obsahovať ani jeden objekt, vtedy je bunka prázdna.



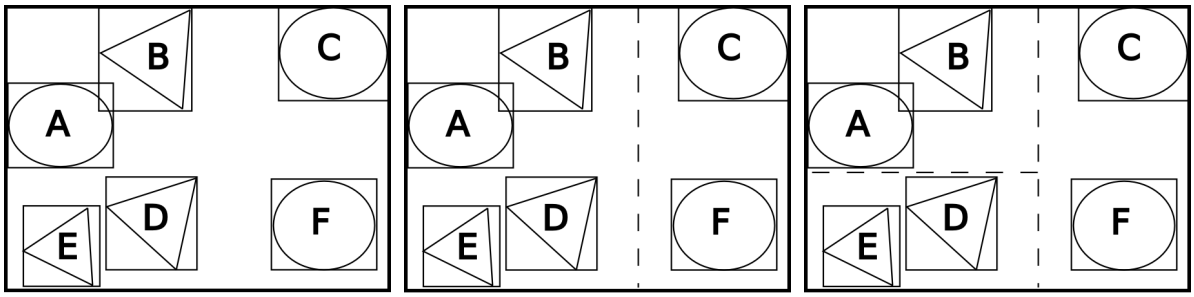
Obrázok 21: Kd-strom

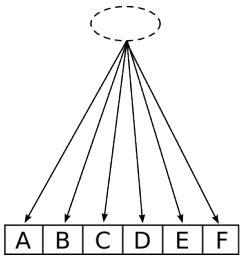
Rovnobežnosť deliacich rovín so súradnými osami zjednodušuje výpočet priesečníka s deliacou rovinou. V trojrozmernom priestore pri výpočte priesečníka lúča s rovinou sa počítajú tri rovnice, toto však neplatí pre rovinu rovnobežnú so súradnicovou osou, kde si vystačíme s jednou. Použitie kd-stromu vyžaduje počítať s prípravnou operáciou na zostavenie samotného stromu. Čas prípravy býva väčšinou zanedbateľný v porovnaní s časom stráveným renderovaním scény, a pár sekundová (minútová) príprava vedie k rapídnemu zníženiu celkového času renderingu (z hodín

na minúty). Základné kd-stromy sú vhodné predovšetkým pre statické scény, pretože každá zmena si vyžaduje opätovné prepočítanie štruktúry deliacich rovín a zostavenie nového stromu.

3.2.2 Zostavenie kd-stromu

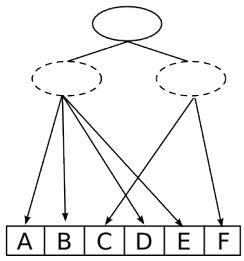
Kd-stromy sa zvyčajne stavajú z hora nadol, postupným delením priestoru na menšie bunky. Zostavenie stromu najčastejšie prebieha rekurzívnym spôsobom, a to tak, že na počiatku vytvoríme listový uzol, obsahujúci všetky objekty. Tento uzol bude koreňom nášho stromu. Po jeho vytvorení sa zavolá rekurzívny algoritmus delenia, ktorý nájde vhodnú deliacu rovinu, aby bolo delenie efektívne. Deliacou rovinou sa rozdelia objekty do nových listových uzlov, a aktuálny uzol sa zmení z listového na interný s dvoma synmi. Pri delení je bežné ak niektoré objekty budú patriť do oboch nových uzlov súčasne, týka sa to objektov, ktorých sa dotýka (resp. pretína) deliaca rovina. Delenie pokračuje rekurzívnym delením oboch synov, a končí pri dosiahnutí stanovenej hĺbky zanorenia alebo ak počet objektov klesne pod stanovenú minimálnu hranicu (prípadne pokrýva prázdny priestor, čo zavisí od použitého algoritmu určujúceho deliacu rovinu, prázdne bunky môžu mať pozitívny dopad na rýchlosť priechodu stromom). Obrázky 22 až 24 ilustrujú prvé kroky zostavenia kd-stromu.





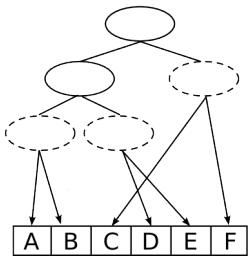
Obrázok 22: Krok č.1

vytvorenie koreňového uzlu



Obrázok 23: Krok č.2

delenie koreňového uzlu



Obrázok 24: Krok č.3

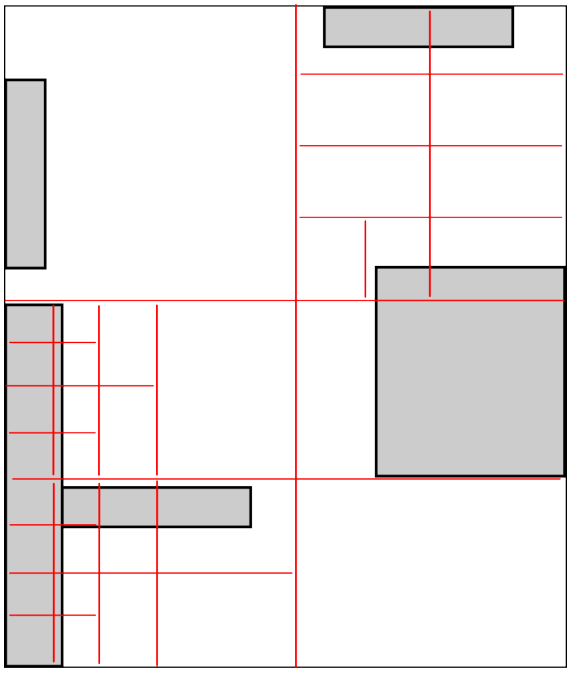
rekurzívne delenie nových uzlov

Ako vidieť na predchádzajúcich obrázkov, deliaca rovina rozdeľuje priestor podľa potreby. Výhodou plávajúcej deliacej roviny je lepšia schopnosť adaptácie na geometriu scény, a dosiahnutie lepších výsledkov oproti ostatným technikám. Rozsiahlu štúdiu rôznych metód napísal Vlastimil Havran [2], kde kd-stromy porazili všetky ostatné techniky.

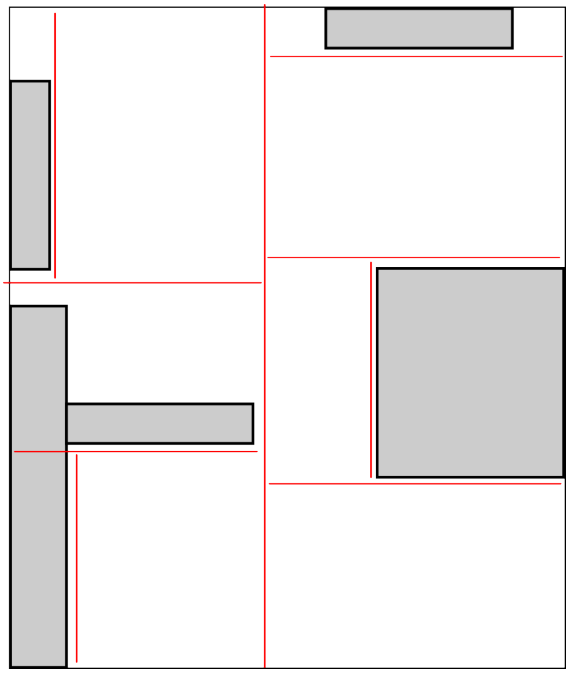
Rekurzívne zostavenie kd-stromu vykonáva nasledujúci algoritmus:

```
ZostavStrom(zoznam objektov)
{
      koren = vytvor novy listovy uzol
      vsetky objekty zo zoznamu prirad do korenového uzlu
      Rozdel(koren, 1)
}

Rozdel(uzol, hlbka)
{
      ak je hlbka vacsia alebo rovna maximalnej hlbke, skonci
      ak je pocet objektov mensi nez minimalny povoleny skonci
      VypocitajDeliacuRovinu(split)
      RozdelUzolPodlaOsi(uzol, split)
      Rozdel(uzol->left)
      Rozdel(uzol->right)
}

RozdelUzolPodlaOsi(uzol, split)
{
      vytvor dva nove listove uzly left a right
      do zoznamu left prirad objekty zasahujuce priestor v lavo
      do zoznamu right prirad objekty zasahujuce priestor v pravo
      zmen typ uzlu z listoveho na interny
      nastav uzlu deliacu rovinu na split
      uzlu prirad synov s objektami left a right
}
```

Pre dosiahnutie kvalitných a efektívnych stromov je dôležité aby prechádzajúci lúč križoval čo najmenej buniek a čím menší počet objektov búdu obsahovať, tým bude prechod rýchlejší. Samotné zostavenie kd-stromu vyžaduje určité výpočtové prostriedky, ktoré ale sú v porovnaní s ušetreným časom proti ne-optimalizovanému riešeniu zanedbateľné, a tak je výhodnejšie stráviť určitú dobu prípravou kvalitného stromu, ktorý potom ušetrí viac času pri renderovaní. Obecne platí že príprava kd-stromu zaberie zlomok z celkového času. Kvalitu výsledného stromu určuje samotné rozdelenie priestoru do buniek deliacimi rovinami. Výpočet deliacich rovin vykonáva deliaci algoritmus, a preto by sa mu mala venovať zvýšená pozornosť, pretože zlý algoritmus dokáže radikálne znížiť rýchlosť celého raytraceru. Obrázok 25 a Obrázok 26 ukazuje príklad použitia dobrého a zlého algoritmu na jednoduchú scénu. V súčastnosti sa javí ako najlepšia metóda SAH (surface area heuristic), využívajúc kalkuláciu ceny pri určovaní deliacej osi.



Obrázok 25: Kd-strom - triviálny deliaci algoritmus vytvára nie moc kvalitný strom



Obrázok 26: Kd-strom - inteligentný algoritmus pracuje lepšie s priestorom

3.2.3 Princípy deliacich algoritmov

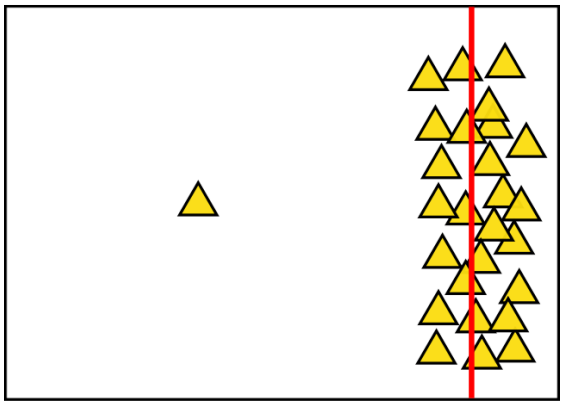
Pri konštruovaní kd-stromu z hora na dol sa stretávame s problémom určenia osi a pozície deliacej roviny. Pre riešenie tohto problému existuje viacero známych techník.

3.2.3.1 Delenie podľa priestorového mediánu

Bunka sa delí vždy v svojom strede, a tým sa vytvárajú dve rovnako veľké bunky, pričom deliace osi sa cyklicky striedajú. Keďže vytvorené bunky majú polovičnú veľkosť, tak sa s pribúdajúcim delením zmenšujú. Tento princíp vytvára priestorovo vyvážené stromy, jeho kvalita je závislá od rovnomerného rozloženia objektov v scéne.

3.2.3.2 Delenie geometrickým stredom

Je jednoduchou metódou pre stanovenie polohy deliacej roviny počítaním geometrického stredu všetkých objektov v bunke. Tento algoritmus je ľahké implementovať, avšak rovnako ako predchádzajúci, jeho výsledok závisí od vhodnosti použitej scény (viz. Obrázok 27).



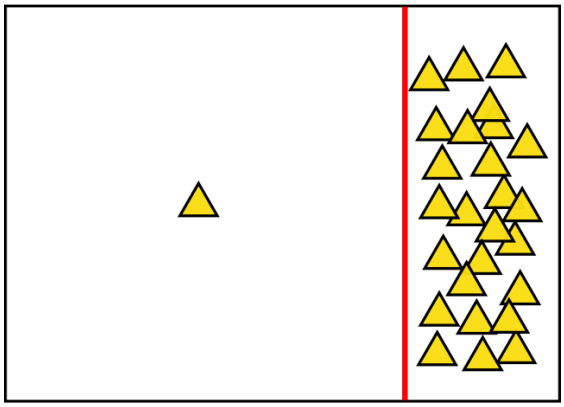
Obrázok 27: Delenie geometrickým stredom

3.2.3.3 Delenie objektovým mediánom

Ďalšou možnosťou je umiestniť deliacu rovinu tak, aby na oboch jej stranách ostal rovnaký (približne) počet objektov. Táto metóda sa javí ako výhodná, pretože získame na oboch stranách vyvážený strom. V skutočnosti táto voľba nieje ideálna, vyváženosť stromu je dôležitá pre binárne vyhľadávacie stromy, kde všetky listy binárneho vyhľadávacieho stromu majú rovnakú pravdepodobnosť úspechu. Naproti tomu bunky kd-stromu majú rôznu veľkosť, a tým rôznu pravdepodobnosť stretu s lúčom (väčšia bunka má väčšiu pravdepodobnosť zásahu lúčom, pretože zaberá viac miesta).

3.2.3.4 Algoritmus „SAH“

Táto technika stanovuje cenu pre jednotlivé pozície deliacej roviny a z nich vyberie delenie s najlepšou cenou. Implementácia je zložitejšia oproti predchádzajúcim metódam, ale zároveň produkuje najlepšie výsledky (podľa [2] a [3]). Zo všetkých spomínaných techník je pre nás najzaujímavejšia a jej popisu sa venuje nasledujúci text.



Obrázok 28: Delenie s najlepšou cenou

3.2.4 Algoritmus „SAH“

Surface area heuristic je metóda založená na teoretickom modeli odhadujúceho cenu putovania lúča kd-stromom za určitých zjednodušujúcich predpokladov:

- všetky lúče križujú scénu
- rovnomerné rozloženie lúčov
- lúče nepretínajú objekty

Tieto predpoklady síce sú nerealistické, ale na druhú stranu nám umožňujú vyjadriť pravdepodobnosť s akou lúč križuje bunku:

$$P(V_{koreň}) = 1 \tag{13}$$

$$P(V_L|V) = \frac{SA(V_L)}{SA(V)} \tag{14}$$

$$P(V_R|V) = \frac{SA(V_R)}{SA(V)} \tag{15}$$

$P(V)$ - geometrická pravdepodobnosť zásahu bunky V lúčom

$V_{koreň}$ - koreňový uzol

V_L - ľavý podstrom (bunka) uzlu V

V_R - pravý podstrom (bunka) uzlu V

$SA(V)$ - obsah plochy bunky $SA(V) = 2(výška \times šírka + výška \times dĺžka + šírka \times dĺžka)$

Potom pri rozhodovaní o delení bunky ohodnotíme každú možnú variantu cenou:

$$C_{delenia}(V_L, N_L, V_R, N_R) = C_{prechodu} + C_{priesečníka}[P(V_L|V)N_L + P(V_R|V)N_R] \tag{16}$$

$C_{delenia}$ - cena uzlu po rozdelení

$C_{prechodu}$ - cena prechodu deliacou rovinou (zanorenia)

$C_{priesečníka}$ - cena výpočtu priesečníka lúča s objektom v scéne

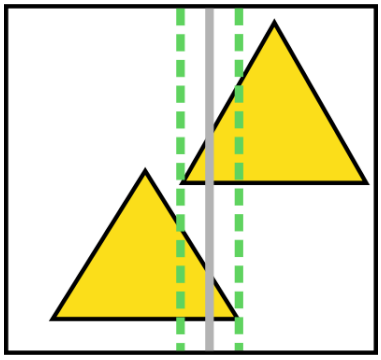
N_L - počet objektov v ľavej bunke

N_R - počet objektov v pravej bunke

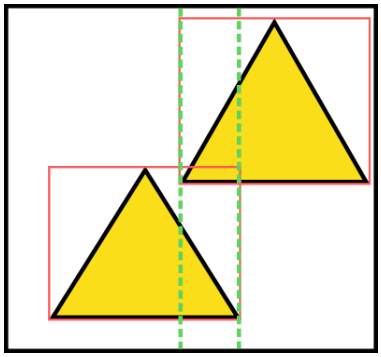
Odhad ceny zahŕňa jednak cenu prechádzania štruktúrou (prechod deliacimi rovinami) stromu a cenu prehľadávania geometrie listu (geometrických objektov v liste stromu). Kvalita kd-stromu

je daná dĺžkou cesty, ktorú musí lúč prejsť kým nájde hľadanú bunku, a tiež počtom objektov v bunke. Preto býva výhodné ak kd strom obsahuje veľké prázdne tzv. výplňové bunky, pomocou ktorých lúč prejde veľkú vzdialenosť s minimálnou cenou.

Funkcia $C_{delenia}$ má spojitý charakter, s výnimkou bodov, v ktorých sa menia hodnoty N_L a N_R, čo sú miesta hraníc geometrických objektov (Obrázok 29), a tieto hranice sú ideálne miesta pre delenie bunky. Počet objektov v bunke je konečný, a tak je aj počet ideálnych deliacich rovín konečný. Analytické počítanie hraníc objektov je náročné, pričom efektívnejšie a s rovnakým výsledkom môžeme zabaliť objekty do hraničných kvádrov rovnobežných zo súradnicovými osami (axis aligned bounding box AABB) a namiesto hrán objektov, použiť hrany obálok (Obrázok 30).



Obrázok 29: Vhodné a nevhodné miesta pre delenie



Obrázok 30: Obalenie a urýchlenie AABB obálkami

Potom ako ohodnotíme všetky deliace roviny, vyberieme z nich rovinu s minimálnou cenou C_{min}, v tomto mieste sa nachádza najvhodnejšie miesto pre rozdelenie bunky. I keď deliaca rovina leží v bode s minimálnou cenou, ešte jej cenu musíme porovnať s cenou nerozdeleného uzlu, ktorá sa rovná výpočtu priesečníkov so všetkými objektami bunky:

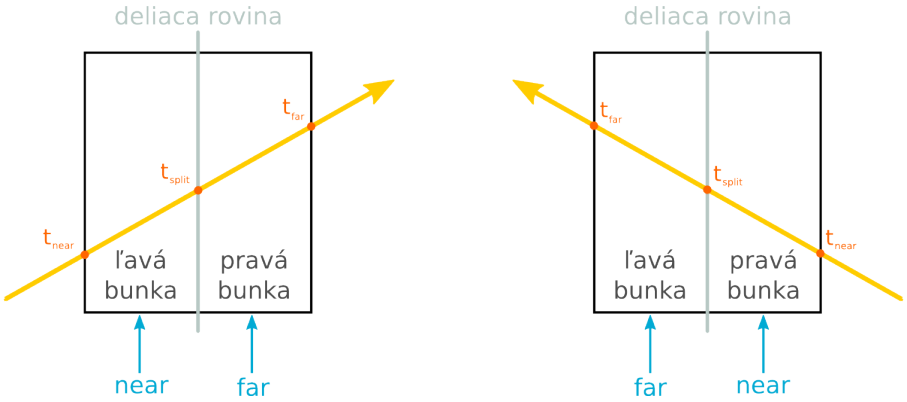
$$C_{nerozdelený}(V) = C_{priesečnika} \times N_V \qquad (17)$$

N_V - počet geometrických objektov patriacich uzlu V

Bunku rozdelíme len vtedy ak je cena C_{min} nižšia než cena nerozdeleného uzlu $C_{nerozdelený}$. Táto podmienka predchádza neefektívnemu deleniu uzlov (napríklad malé bunky s nízkou geometrickou pravdepodobnosťou, ...). Detailnejšiu analýzu techniky SAH nájdete v [9].

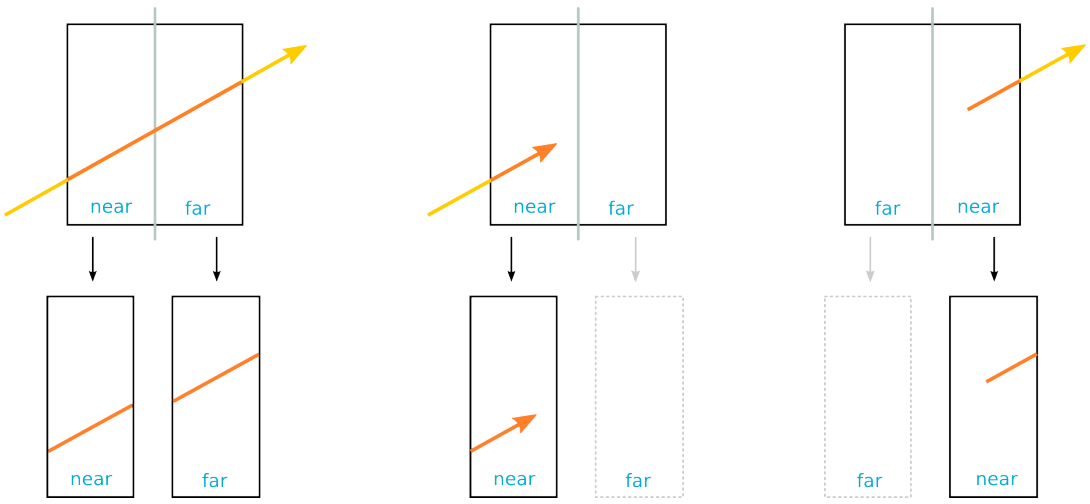
3.2.5 Prechod kd-stromom

Algoritmus prechodu kd-stromu pracuje v dvoch fázach, prvou je prechod internými uzlami (deliacimi rovinami), druhou je hľadanie priesečníka v objektoch listového uzlu. Počas prechodu sú postupne prehľadávané potenciálne bunky (ktoré pretína lúč). Počiatok hľadania začína koreňovým uzlom, koniec nastáva pri prvom nájdení bunky, ktorej objekty pretínajú lúč. Hľadanie tiež môže skončiť neúspechom ak lúč nepretína žiaden objekt. Pri putovaní interným uzlom musíme určiť správne poradie prechodu jeho buniek (určiť `near` a `far` `bunky` viz. Obrázok 31).



Obrázok 31: Správny prechod interným uzlom - určenie `near` a `far`

Spracovanie interného uzlu má vždy jednu z nasledujúcich podôb podľa relatívnej polohy lúča voči bunke:



a) spracuj obe bunky *b) spracuj `near`* *c) spracuj `near`*

Variant a) najprv spracuje bunku `near`, skončí ak v nej nájde priesečník, inak nasleduje spracovanie bunky `far`. Varianty b) a c) vyraďujú zo spracovania vzdialenejšiu bunku, pretože do nej lúč

nezasahuje. Možnosti b) a c) spracovávajú iba bližšiu bunku, tieto varianty nastávajú pri splnení podmienky $t_{near} >= t_{split} || t_{far} <= t_{split}$, pričom t_{split} je priesečník lúča s deliacou rovinou, t_{near} a t_{far} ohraničujú segment lúča zasahujúci bunku. Spracovanie listového uzla pozostáva z výpočtu priesečníka lúča so všetkými objektami uzlu, z nich sa potom vyberie najbližší. Jednoduchá priamočiara implementácia využívajúca rekurziu je vyjadrená nasledovným algoritmom:

```
NajdiNajblizsi (luc, dlzka)
{
      najdi vzdialenosti t_near a t_far v ktorých luc pretina hranicny
      kvader sceny

      ak luc nepretina scenu skonci
      ak t_near < 0 => t_near = 0
      ak t_far > dĺžka => t_far = dlžka
      zavolaj PrejdiStrom(luc, t_near, t_far, koren) a vrat jeho vysledok
}

PrejdiStrom(luc, t_near, t_far, uzol)
{
      ak je uzol listovym uzlom
      {
            pokus sa najst najblizsi priesecnik s objektami
            ak najdes   => vrat ho ako hladany objekt
            ak nenajdes => vrat neznamy objekt
      }

      urci near a far bunky
      urci priesecnik t_split luca s deliacou rovinou

      ak t_near >= t_split alebo t_far <= t_split
      {
            zavolaj PrejdiStrom(luc, t_near,  t_far, near) a vrat jeho
            vysledok
      }
      inak
      {
            zalovaj PrejdiStrom(luc, t_near, t_split, near), ak najde
            pretinajuci objekt, vrat jeho vysledok, inak pokracuj

            zavolaj PrejdiStrom(luc, t_near, t_split, far) a vrat jeho
            vysledok
      }
}
```

3.2.6 Iteratívny prechod kd-stromom

Keďže prechod kd-stromom je jedna z najčastejšie volaných funkcií raytraceru, je dobré aby jej implementácia bola čo najrýchlejšia. Základný algoritmus využíva rekurziu pre pohyb kd-stromom, každé zanorenie do uzlu vyžaduje nové volanie funkcie. Tieto rekurzívne zanorenia môžeme odstrániť prepísaním algoritmu na iteratívny tvar s využitím zásobníku. Upravený iteratívny algoritmus demonštruje nasledujúci pseudokód.

```
NajdiNajblizsi (luc, dlzka)
{
      inicializuj zasobnik
      najdi vzdialenosti t_near a t_far v ktorých luc pretina hranicny
      kvader sceny

      ak luc nepretina scenu => skonci
      ak t_near < 0, nastav t_near = 0
      ak t_far > dĺžka, nastav t_far = dĺžka
      zavolaj PrejdiStrom(luc, t_near, t_far, koreň) a vrat jeho vysledok
      uzol = koren

      nekonecny  cyklus
      {
            pokial uzol je internym ozlom
            {
                  urci far a near
                  urci t_split

                  ak t_split <= t_near alebo t_split >= t_far =>   uzol = near
                  inak
                  {
                        na zasobnik uloz (far, t_split, t_far)
                        uzol = near
                        tfar = t_split
                  }
            }
            ak uzol je listovym uzlom
            {
                  pokus sa najst najblizsi priesecnik s objektami
                  ak najdes   => vrat ho ako hladany objekt
                  ak nenajdes => pokracuj v cykle
            }
            ak je zasobnik prazdny => vrat neznamy objekt
            vezmi prvok z vrchu zasobnika (uzol, t_near, t_far)
      }
}
```

3.3 Výpočet priesečníka lúča s trojuholníkom

Pri počítaní priesečníka lúča s trojuholníkom je možné využiť viacero techník, pričom v súčastnosti sú najefektívnejšie techniky využívajúce barycentrické súradnice. Algoritmus najprv bežným spôsobom vypočíta priesečník lúča s rovinou trojuholníka, potom testuje vypočítaný priesečník či leží v oblasti trojuholníka.

Priesečník lúča P_P (s počiatkom O a smerom $\vec{D}$) s rovinou trojuholníka (danou normálou $\vec{N}$) vypočítame:

$$P_p = O + t\vec{D} \tag{18}$$

$$t = -\frac{(O-A)\cdot\vec{N}}{\vec{D}\cdot\vec{N}} \tag{19}$$

Všetky body trojuholníka môžeme popísať rovnicou:

$$P = \alpha A + \beta B + \gamma C \tag{20}$$

Pričom P je bod ležiaci v trojuholníku, body A, B, C sú vrcholmi trojuholníka. Bod ležiaci v trojuholníku musí spĺňať podmienku:

$$\alpha + \beta + \gamma = 1 \tag{21}$$

Rovnicu (20) potom môžeme prepísať do tvaru:

$$\beta(B-A) + \gamma(C-A) = P - A \tag{22}$$

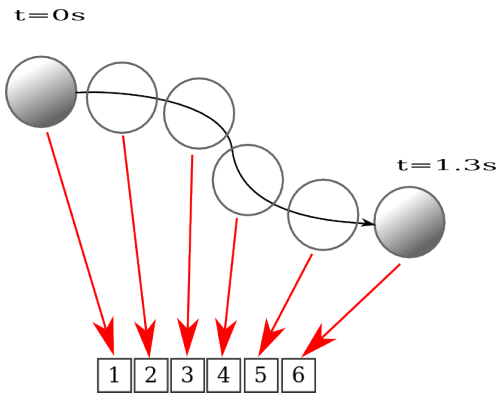
Túto rovnicu môžeme riešiť v trojrozmernom priestore, efektívnejšie je ale jej riešenie v dvojrozmernom, preto trojuholník premietneme na jednu z rovín XY, XZ, YZ. Najvhodnejšiu rovinu vyberieme podľa dominantnej osi normály trojuholníka, ak je dominantnou osou Z, trojuholník premietneme na XY (pre Y => XZ, Z=>XY). Koeficienty z rovnice (22) potom vypočítame:

$$\begin{gathered}\beta = (u_1 h_2 - u_2 h_1)/d \\ \gamma = (h_1 v_2 - h_2 v_1)/d \\ d = (u_1 v_2 - u_2 v_1) \\ u = C - A \\ v = B - A \\ h = P_p - A\end{gathered} \tag{23}$$

Kde u_1, v_1, h_1 značí prvú premietnutú súradnicu, a u_2, v_2, h_2 druhú. Pre urýchlenie výpočtu môžeme delenie v rovnici (23) nahradiť násobením $1/d$ a predpočítať konštantné hodnoty $\vec{N}, u, v, d$ (resp. $1/d$). Celý tento algoritmus aj s predpočítaním hodnôt je implementovaný v demonštračnej aplikácii.

3.4 Dynamické objekty

Keďže distribuovaný raytracing so sebou prináša možnosť renderovania dynamických scén, je nutné riešiť architektúru raytraceru s podporou pohybujúcich sa objektov. Pohyb objektov má spojitý priebeh, je ho možné simulovať na počítači. Pre rýchle spracovanie v raytraceri je vhodné pohyb každého objektu zachytiť v určitých intervaloch a uchovať ako transformačnú maticu s pozíciou a rotáciou objektu. Takže každý objekt bude obsahovať pole matíc uchovávajúcich jeho dynamiku v scéne. Statické objekty obsahujú len jednu transformačnú maticu, ktorá určuje ich východziu polohu. Takýto prístup je vhodný pre rendering krátkych scén, akou je napríklad jeden snímok renderovaný distribuovaným raytracerom. Jej výhodou je rýchlosť a priamočiarosť implementácie.

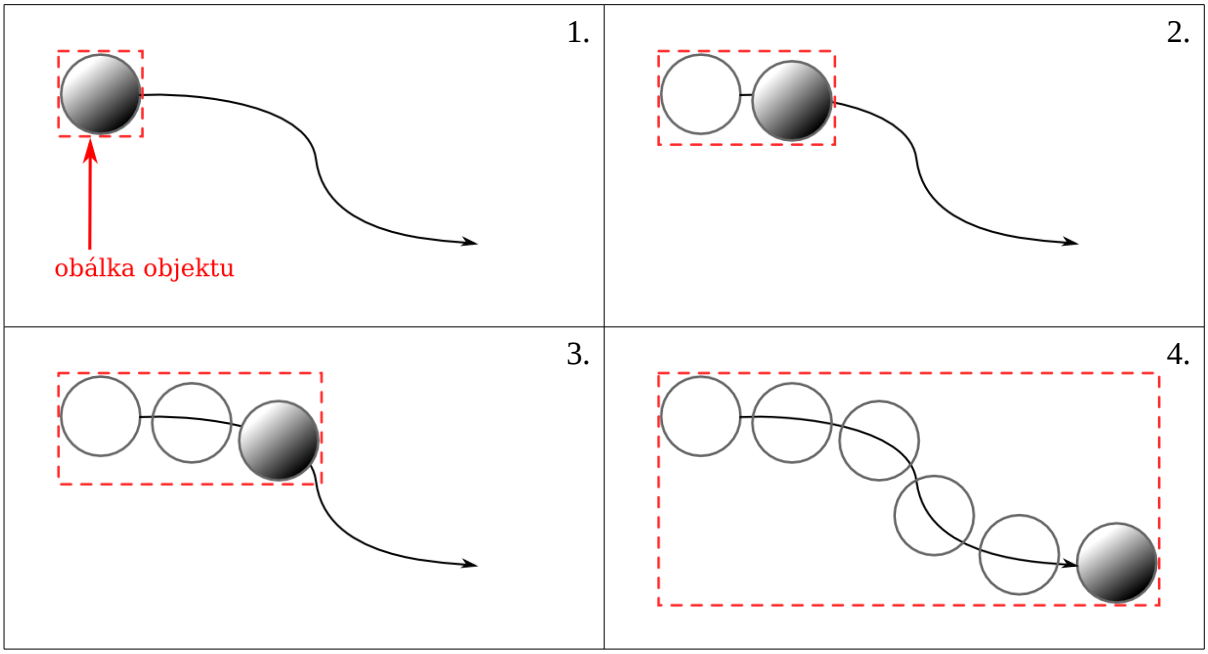


Pri výpočte priesečníka lúča s telesom v danom čase *t* sa z tohto času vypočíta index udávajúci pozíciu v poli transformačných matíc objektu. Pomocou tejto matice transformujeme objekt na jeho polohu v čase *t*, a zavoláme metódu pre výpočet priesečníka lúča s objektom.

S pohybom objektov objektov je nutné počítať aj pri vytváraní kd-stromu. Jednoduchou variantou je pre každý jeden snímok scény vytvoriť vlastný strom, ale takéto riešenie spotrebuje veľa pamäte a tiež je nutné každý kd-strom vypočítať. Dobrou variantou je použitie hraničných obálok.

Každý objekt bude mať vlastnú obálkou, tvar obálky je kváder, ktorého strany sú rovnobežné so súradnými osami (AABB). Rozmery hraničnej obálky sú dané rozmermi objektu a pohybom objektu v scéne. Takže v počiatočnej pozícii obálka tesne obopína objekt, a postupne rastie s pohybom objektu. Postup vytvorenia obálky pre statický objekt je jednoduchší, pretože obálka obalí

objekt v jeho východzej polohe a ďalej nerastie. Použitie obálok je limitované na malý časový interval, pri väčšom intervale by pohybujúce objekty mohli vytvoriť veľké, prekrývajúce sa obálky.

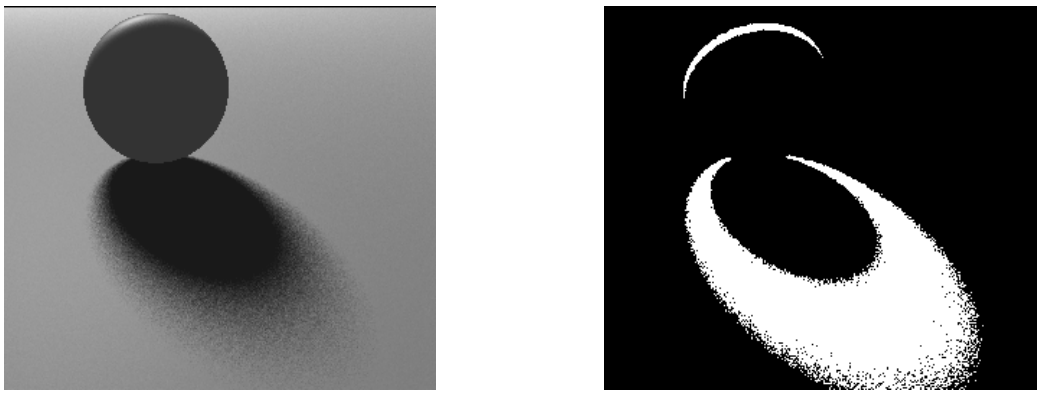


Obrázok 32: Postup vytvorenia obálky pohybujúceho sa objektu

Použitie hraničných obálok zabezpečí elegantné riešenie pohybu tam, kde sa uvažuje s malým časovým okamžikom, a preto je vhodné pre distribuovaný raytracer.

3.5 Mäkké tiene s predpoveďou

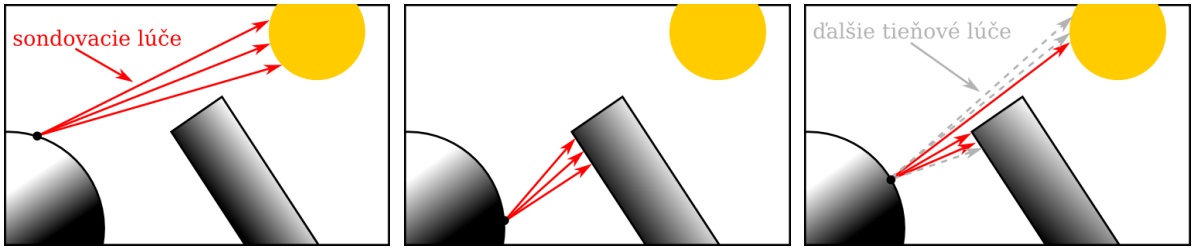
Použitie mäkkých tieňov niekoľkonásobne predlžuje čas renderovania scény, veľkosť spomalenia závisí od nastavenia počtu tieňových lúčov. Malý počet tieňových lúčov je nedostatočný pre vytvorenie jemného polotieňa, vytvorený polotieň obsahuje priveľké množstvo šumu (Obrázok 14). Veľký počet tieňových lúčov vytvára realisticky pôsobiaci polotieň (bez šumu), ale za cenu dlhšieho času renderovania. V bežných scénach býva veľká časť povrchov telies buď kompletne osvetlená, alebo úplne zatienená (vzhľadom na jeden svetelný zdroj). Toto sú miesta, na ktorých nevzniká polotieň (Obrázok 33). Ak z takéhoto miesta vyšleme zväzok tieňových lúčov, tak buď celý dopadne na povrch svetelného zdroja, alebo celý narazí na prekážku. Inak povedané, všetky tieňové lúče vrátia rovnaký výsledok a ich počet môžeme redukovať jednoduchou heuristikou (predpoveďou).



Obrázok 33: Polotienové oblasti (napravo), obrázok scény (naľavo)

Základom tejto techniky je zväzok tieňových lúčov, z ktorých časť slúži ako sondovacie lúče. Prostredníctvom nich zistíme či je daný bod pravdepodobne v polotieni, alebo nieje. Postup pri použití predpovede je nasledovný:

– vyslanie *n* sondovacích tieňových lúčov k svetelnému zdroju
– ak všetkých *n* lúčov dopadne na svetelný zdroj, skonči výpočet => bod je plne osvetlený (Obrázok 34a)
– ak všetkých *n* lúčov narazí na prekážku, skonči výpočet => bod je úplne zatienený (Obrázok 34b)
– inak je bod v polotieni, vyšli zvyšné lúče a vypočítaj mieru zatienenia (Obrázok 34c)

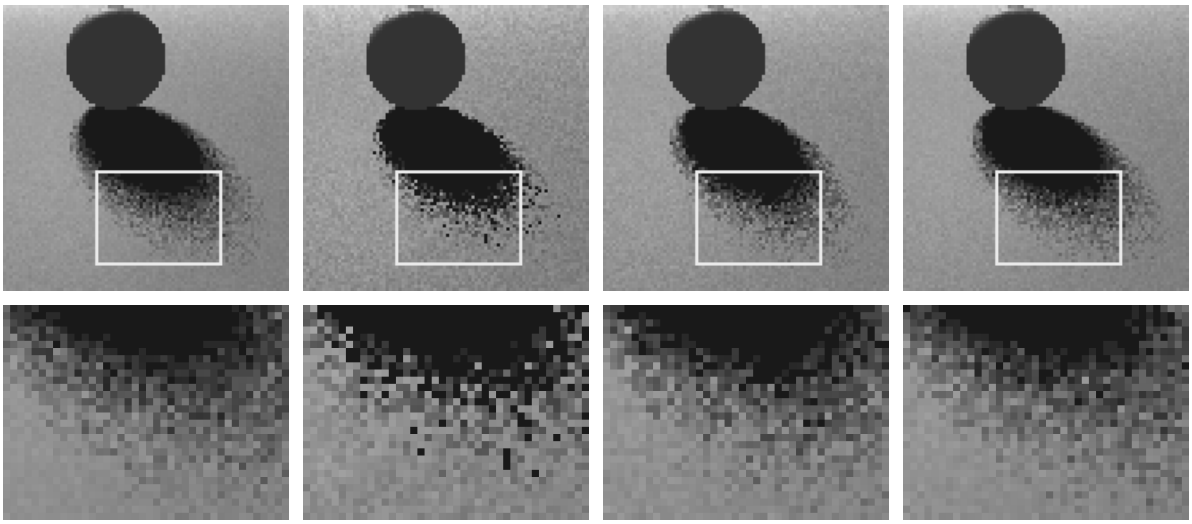


a) plne osvetlený bod *b) úplný tieň* *c) polotieň*

Obrázok 34: Použitie sondovacích lúčov

Viditeľným efektom tejto heuristiky je zmenšenie polotieňových oblastí, veľkosť tohto zmenšenia reguluje počet použitých sondovacích lúčov. Čím viac lúčov použijeme, tým je zmenšenie menej patrné, ale tiež výpočtovo náročnejšie. Vplyv množstva sondovacích lúčov na kvalitu zobrazuje Obrázok 35. Z neho je patrný efekt zhoršenia kvality polotieňa pri nízkom počte sondovacích lúčov (Obrázok 35b), ale zároveň je pri nízkom počte dosiahnuté najvyššie urýchlenie. Optimálny počet sondovacích lúčov je približne 50% z celého zväzku tieňových lúčov. Pri tomto

čísle dosahuje raytracer urýchlenie skoro 40%, pričom dopad negatívneho efektu zmenšenia polotieňových oblastí je minimálny (Obrázok 35c). Ak pre výpočet predpovede použijeme aspoň 75% z celkového počtu tieňových lúčov, získame vyrenderovaný obraz scény takmer totožný s renderom bez použitia heuristickej techniky, no urýchlenie bude maximálne 25% (Obrázok 35d).



a) bez predpovede | *b) predpoveď 25% urýchlenie 66%* | *c) predpoveď 50% urýchlenie 45%* | *d) predpoveď 75% urýchlenie 25%*

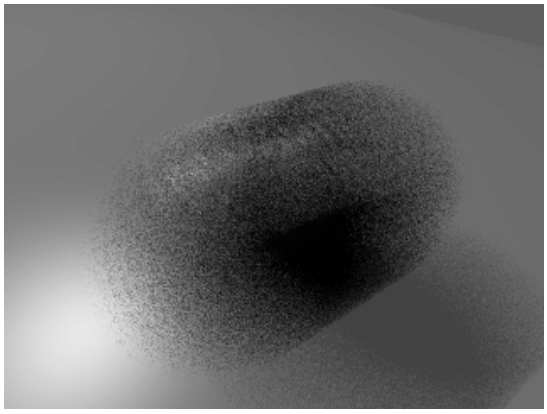
Obrázok 35: Vplyv počtu lúčov na veľkosť a kvalitu polotiena, obrázok a) je renderovaný bez heuristiky, obrázky b), c) a d) sú renderované s určitým počtom sondovacích lúčov

3.6 Optimalizovaný „motion blur“

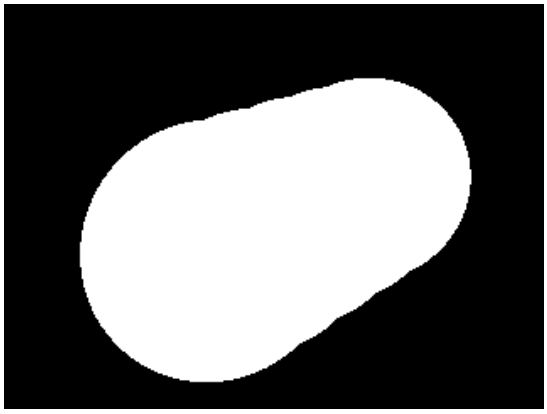
Efekt „motion blur“ vzniká pohybom objektov, netýka sa statických objektov. Pre určenie farby pixlu vysielame do scény lúče v rôznych dobách. Ak lúč v každú dobu dopadne na rovnaké miesto v scéne, bude mať farbu tohto miesta. Ak počas časového intervalu ale dopadne na rôzne miesta, jeho výsledná farba bude daná všetkými miestami, na ktoré dopadol. V prvom prípade je vysielanie viacerých lúčov zbytočné, lebo vždy vrátia ten istý výsledok. V druhom prípade lúč preťal pohybujúce sa objekty a teda časť obrazu bude rozmazaná. V scéne často-krát dominuje prvý prípad (v biliarde sa pohybujú len gule, stôl a zvyšok scény je statický), že je veľa lúčov vysielaných zbytočne do statických miest. Preto je vhodné zistiť v ktorých pixloch obrazu sa vyskytujú pohybujúce objekty, a pri výpočte len do týchto pixlov vysielať lúče v rôznych dobách. Pre statické pixle sa vyšle len jeden lúč. Teda je potrebné vytvoriť masku vyznačujúcu dynamické objekty, a podľa nej voliť množstvo vyslaných lúčov (pre statické jeden, pre dynamické viac).

Vytvorenie masky prebieha v niekoľkých krokoch, je to prípravná operácia (podobne ako vytvorenie kd-stromu), takže maska musí byť vytvorená pred samotným renderingom. Jej výpočet zaberá určitý čas, ktorý ale je v porovnaní s ušetreným časom malý. Veľkosť masky odpovedá rozlíšeniu výsledného obrazu. Prvým krokom je inicializácia všetkých hodnôt masky na 0. Druhým krokom je vyznačenie pohybujúcich sa objektov do masky, resp. označenie pixlov ktorými prechádzajú pohybujúce sa objekty (Obrázok 37). Na zistenie či daný pixel pokrýva pohybujúci sa objekt vyšleme cez neho do scény zväzok v čase rovnomerne rozptýlených lúčov, ak lúč dopadne aspoň na jeden pohybujúci sa objekt, označíme pixel ako dynamický (masku mu nastavíme na 1). Vyslanými lúčmi hľadáme len najbližšie objekty, v mieste dopadu nepočítame osvetľovací model. V treťom kroku pridáme do masky všetky zatienené miesta (Obrázok 38). Posledným krokom je rozšírenie takto vytvorenej masky o 4 pixle v každom bode. Zmyslom tejto operácie je vyhladenie výslednej masky, pretože po predchádzajúcich operáciách môže byť zašumená.

Tento postup ukazujú nasledujúce obrázky



Obrázok 36: Pohybujúci sa objekt



Obrázok 37: Maska pohybu

Obrázok 38: Tieňová maska

Obrázok 39: Výsledná maska

3.7 5d raytracing

Postup výpočtov jednotlivých efektov v distribuovanom raytracingu je rekurzívny, takže jeden efekt sa počíta z druhého. Poradie výpočtu jednotlivých efektov je nasledovné:

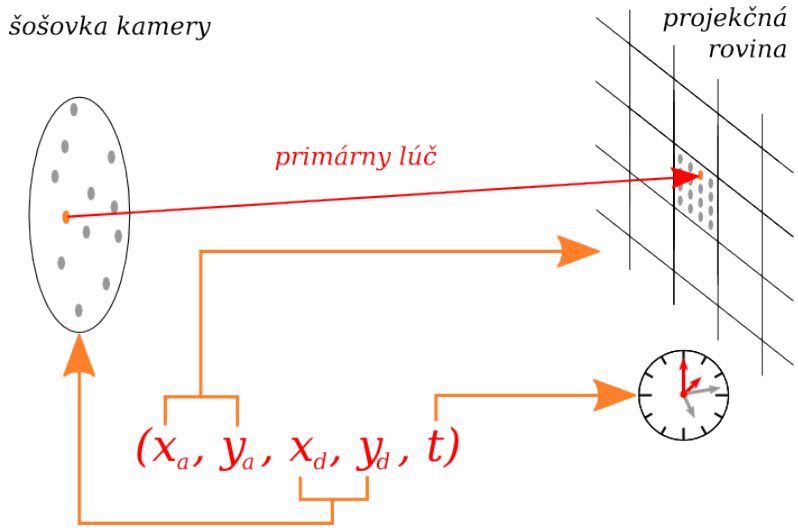
pre každý pixel:

- **vyšli zväzok antialiasingových lúčov**
  - **vyšli zväzok časových „motion blur“**
    - **vyšli zväzok lúčov hĺbky ostrosti**
      - **výpočet farby (distribúcia tieňových, reflexných a refrakčných lúčov)**

Takýto postup generuje veľké množstvo primárnych lúčov, napríklad ak chceme vyrenderovať obrázok s efektom hĺbky ostrosti (16 lúčov) a antialiasingom (16 lúčov). tak na jeden pixel obrazu pripadá zväzok 256 primárnych lúčov (oproti jednému pri klasickom raytracingu). Výpočet prvých troch efektov produkujúcich primárne lúče je možné upraviť tak, že každý jeden vyslaný primárny lúč bude počítať všetky tieto efekty súčasne. Distribúcia lúča bude potom daná päticou (x_a, y_a, x_d, y_d, t) , kde x_a, y_a reprezentuje antialiasingové vychýlenie lúča, x_d, y_d rozptyl na šošovke kamery (hĺbka ostrosti) a t distribúciu lúča v čase („motion blur“). Pretože je distribúcia lúča daná touto päticou je táto technika nazvaná 5d raytracing. Táto technika zjednoduší rekurzívny výpočet raytracingu na:

pre každý pixel:

- **vyšli zväzok primárnych lúčov (pre každý generuj novú** (x_a, y_a, x_d, y_d, t) **)**
  - **výpočet farby (distribúcia tieňových, reflexných a refrakčných lúčov)**



Obrázok 40: Primárny lúč v 5d raytracingu

Jednotlivé hodnoty z pätice (x_a, y_a, x_d, y_d, t) sú náhodne generované pre každý jeden primárny lúč, pričom rešpektujú požiadavky jednotlivých efektov. Takže x_a, y_a sú hodnoty z intervalu $(-0.5, 0.5)$ dané pravidelnou mriežkou. Hodnoty x_d, y_d udávajú rovnomerný rozptyl lúčov na šošovke kamery s polomerom R . Časový údaj t je generovaný presne ako vyžaduje efekt „motion blur“ (viz. 3.1.3 Distribúcia lúčov pre „motion blur“).

Výhodou tohto prístupu je že jeden lúč počíta tri efekty súčasne, a teda pracuje efektívnejšie ako lúč v rekurzívnom raytracingu. Napríklad ak by sme chceli vyrenderovať scénu s antialiasingom (16 lúčov) a efektom hĺbky ostrosti (16 lúčov), budeme v klasickom raytracingu na jeden pixel vysielať 256 primárnych lúčov. Naproti tomu je možné dosiahnuť podobný výsledok s menším počtom primárnych lúčov ak použijeme 5d raytracing, preto je táto technika vhodná pri renderingu s použitím viacerých efektov súčasne.

3.8 Viacvláknový rendering

V súčastnosti je trend vývoja procesorov smerovaný na viacjadrové procesy, pretože ich vývoj narazil na hranicu možného výkonu jedno jadrových procesorov. Z tohto dôvodu by mali byť náročné algoritmy použité v programe optimalizované pre využitie viac jadier (paralelný výpočet) . Keďže raytracing je časovo náročný, je pravdepodobné že bude počítaný na výkonných viac jadrových počítačoch.

V raytracingu je možné naplno využiť plný výkon viacjadrového procesoru tak, že obraz rozdelíme na malé štvorce (napr. 32x32 pixlov), ktoré uložíme do fronty. Každé vlákno dostane na spracovanie jeden štvorec z obrazu, po jeho spracovaní si z fronty vezme ďalší. Takéto spracovanie pokračuje až do vyprázdnenia fronty. Výhodou je rovnomerné rozdelenie záťaže výpočtu pre všetky renderovacie vlákna. Optimálnou hodnotou počtu vlákien je počet procesorových jadier systému.

Viacvláknové programovanie vyžaduje opatrný prístup k zdieľaným zdrojom aplikácie, v tomto prípade je kritickým miestom obrazová fronta. K nej je potrebné riadiť prístup pomocou synchronizačného semafóru. Tento objekt zaistí výlučný prístup vždy len jedného vlákna k fronte.

4 Implementácia

Táto kapitola sa zaoberá návrhom, popisom implementácie a ovládania demonštračného raytraceru. Tento program bol vyvíjaný prevažne na Linuxe, ale s dôrazom na prenositeľnosť do Windows. Ako vývojové prostredie bolo bolo použité volne dostupné prostredie Code::Blocks s prekladačom GCC. Celý program je implementovaný v objektovo orientovanom jazyku C++, pretože objektovo orientovaný prístup uľahčuje návrh a poskytuje dobrý výkon. Raytracer bol vyvíjaný postupne, najprv bol zostrojený klasický jednoduchý raytracer a postupne bol rozširovaný o nové vlastnosti.

4.1 Návrh architektúry

Samotnej fáze implementácie predchádzal návrh programu, v ktorom sa určili ciele a funkcie očakávané od programu. Tieto ciele boli nasledovné:

- 2 režimy práce: interaktívny a z príkazového riadku
- výstup prostredníctvom obrázkov
- načítanie scény zo súboru
- podpora viacvláknového renderingu
- prenositeľnosť na Linux aj Windows
- jednoduchá simulácia biliardu
- podpora načítania modelov z OBJ formátu

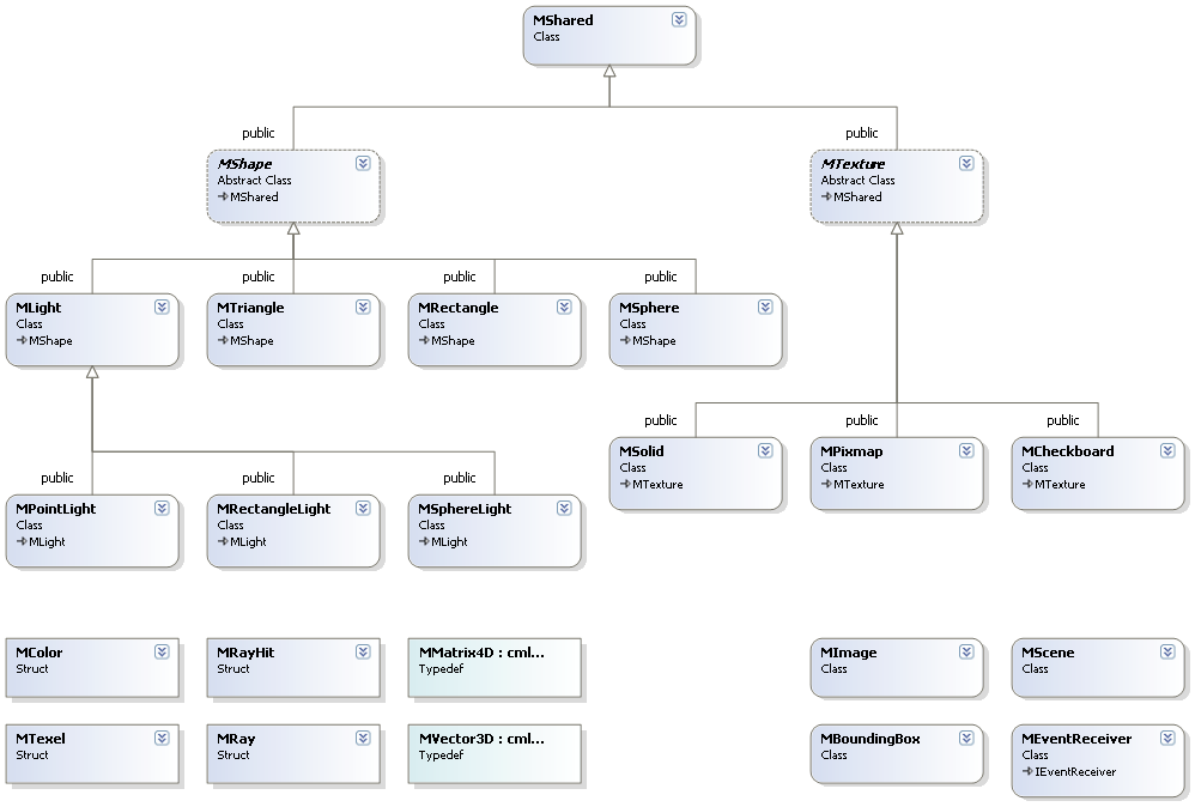
Pre riešenie určitých častí programu bolo lepšie použiť existujúce knižnice, ako si ich vytvárať od základu. Na grafické užívateľské rozhranie a interaktívnu časť je použitá knižnica Irrlicht (GUI, ovládanie klávesnicou a myšou, načítanie modelov z OBJ súborov, podpora obrazových formátov BMP, JPG, PNG). Simuláciu fyziky biliardu je dobré ponechať knižnici ODE, ktorú používa aj mnoho komerčných projektov. Ďalej program využíva generátory náhodných čísiel z knižnice Boost. Samotné sledovanie lúča vyžaduje prácu s maticami a vektormi, na ktorú je použitá knižnica CML. Súčasťou návrhu je aj vlastný súborový formát založený na XML štruktúre. Načítanie a spracovanie XML súborov je použitá knižnica TinyXML. Podporu vlákien zaisťuje knižnica SDL.

Program má pracovať v dvoch režimoch, v interaktívnom a z príkazového riadku. Interaktívny režim obsahuje jednoduchú scénu miestnosti s biliardom. V tejto scéne môže užívateľ hrať biliard, a ktorýkoľvek okamžik hry si nechať vyrenderovať distribuovaným raytracerom. Druhý režim je spúšťaný z príkazového riadku, načíta vstupný súbor so scénou a parametrami renderu a okamžite

prevedie rendering. Priebežný výstup bude zobrazovaný v okne programu a výsledný obrázok sa uloží do súboru. Ovládanie aplikácie sa venuje podkapitola 4.3 Ovládanie programu.

4.2 Popis tried

Postupným návrhom programu vznikal diagram tried, ktorý začínal s malým počtom základných tried, postupne sa tieto triedy rozrastali o nové funkcie, a vznikali nové odvodené triedy (hlavne z triedy *MShape*),. Konečný diagram najdôležitejších častí programu zobrazuje nasledovný obrázok.



Obrázok 41: Diagram tried aplikácie

Nasleduje popis najvýznamnejších tried a štruktúr programu, detailnejšiu dokumentáciu je možné nájsť priamo v zdrojovom kóde programu.

MColor

Je štruktúra uchovávajúca farbu v modeli RGB, jednotlivé farebné zložky sú reprezentované číslami s plávajúcou desatinou čiarkou, povolený rozsah hodnot je $\langle 0,1 \rangle$.

MVector3D

Trieda reprezentujúca vektor v trojrozmernom priestore, prístup k položkám umožňuje operátor []. Táto trieda obsahuje metódy pre základnú prácu s vektormi, napríklad normalizácia vektoru, výpočet dĺžky, ...

MMatrix4D

Matica s rozmermi 4x4, v programe slúži predovšetkým na vyjadrenie rôznych zložených transformácií v trojrozmernom priestore (hlavne posunutí a rotácií objektov a lúčov). Podobne ako trieda vektoru aj matica má implementované základné maticové operácie (inverzia, rotácia, posun, jednotková matica, ...).

MShared

Je jednoduchá trieda implementujúca objekt s počítadlom referencií. Počítadlo referencií elegantne rieši správu pamäti pri zdieľaní objektu rôznymi časťami programu. Táto trieda má dve metódy, jednu pre zvýšenie počítadla referencií, a druhú pre jeho zníženie. Ak pri znižovaní počítadlo dosiahne hodnotu 0, značí to že objekt už nikto nepoužíva a preto sa zruší. Príkladom je textúra používaná viacerými objektami v scéne. Keď objekty postupne začnú zanikať, postupne budú aj znižovať počítadlo referencií, pri zaniknutí posledného objektu sa jeho hodnota klesne na 0, a textúra sa uvoľní z pamäte.

MShape

Táto abstraktná trieda definuje rozhranie objektov scény. Je rodičovskou triedou každého grafického prvku scény. Pre pridanie nového telesa, napríklad kužeľa je nutné implementovať toto rozhranie, ktoré obsahuje tri základné funkcie:

intersect - výpočet najbližšieho priesečníka lúča s objektom, tiež určí textúrovacie súradnice, normálu, a smer v mieste dopadu.

shadowHit - metóda na zistenie či tieňový lúč križuje objekt, a v akej vzdialenosti ho križuje. Je odľahčenou formou *intersect*, pretože pri počítaní tieňových lúčov nepotrebujeme určiť textúrovacie súradnice, smer ani normálu v mieste dopadu.

prepare - táto metóda má na starosti prípravu objektu na rendering, predpočítanie pomocných hodnôt a výpočet hraničného kvádra. Je vždy volaná pred začiatkom renderovania scény.

Každému objektu triedy *MShape* je priradený materiál, a pozícia v priestore. Ďalej môže obsahovať textúru a animáciu pohybu.

MTexture

Definuje rozhranie dvojrozmernej textúry. Pozíciu na textúre určujú dve súradnice $u, v \in \langle 0, 1 \rangle$. Týmto rozhraním je možné implementovať rôzne procedurálne textúry ako aj bitmapové textúry.

MLight

Trieda poskytujúca rozhranie zdrojom svetla. Je rodičovskou triedou bodového zdroja svetla (*MPointLight*), aj plošných zdrojov (*MSphereLight*, *MRectangleLight*). Každé svetlo má nastaviteľnú farbu a počet použitých tieňových lúčov, ktoré sa budú k nemu vysielať.

MImage

Trieda bitmapového obrázku, slúži ako most napájajúc sa na triedu *IImage* z knižnice *Irrlicht*, ktorá má kompletnú implementáciu bitmáp, vrátane uloženia do rôznych formátov (napr. PNG).

MScene

Táto trieda je najzložitejšou v celom programe, implementuje kompletnú scénu aj raytracer, a nieje určená pre dedenie. V sebe zahŕňa objekty scény, nastavenie kamery, kd-strom, a aj algoritmus distribuovaného raytracingu. Umožňuje hľadanie objektov križujúcich sa s lúčom prostredníctvom kd-stromu (rýchle hľadanie) alebo testovaním lúča s každým objektom (pomalé, predovšetkým pre testovacie účely).

4.3 Ovládanie programu

Demonštračná aplikácia *TurboRAY* pracuje v dvoch režimoch, a každý má svoje špecifické ovládanie. Prvým režimom je interaktívny, ktorý sa zapne pri spustení programu bez parametrov. Druhou možnosťou je neinteraktívny režim práce s príkazového riadku.

4.3.1 Interaktívny režim

Po spustení interaktívneho módu sa zobrazí v okne aplikácie miestnosť s biliardovým stolom, na ktorom je možné hrať jednoduchý biliard, pričom je možné nechať si ktorýkoľvek okamih hry vyrenderovať distribuovaným raytracerom s rôznymi parametrami a efektami. Účelom tohto režimu nieje simulovať plnohodnotnú biliardovú hru, ale demonštrovať distribuovaný raytracing.

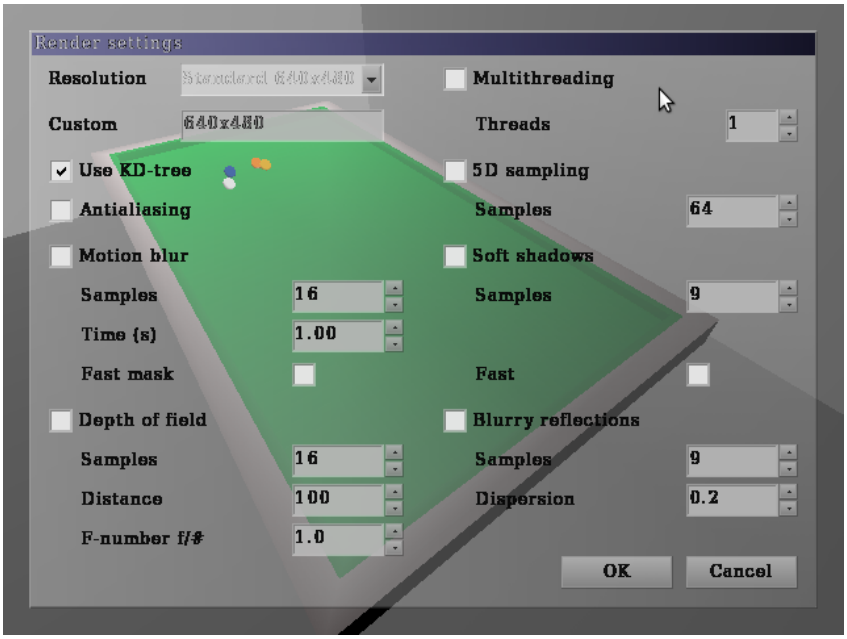
Pohyb kamery sa jednoducho ovláda myšou a smerovými šípkami. Okrem pohybu kamery je možné udierať do jednotlivých gúľ na biliardovom stole, a to zamierením nášho pohľadu na guľu v smere úderu (mierenie uľahčuje pomocný kríž v strede obrazovky). Stlačením a pridržaním ľavého

tlačítka myši sa volí sila úderu, ktorej aktuálna veľkosť je znázornená graficky silomerom v pravom dolnom rohu okna. Uvoľnením tlačítka sa prevedie úder. Po spustení sú na stole prichystané 4 biliardové gule, ďalšie sa dajú zobraziť alebo skryť v paneli nastavenia gúľ, tento panel sa zapína stlačením klávesy *P*. V tomto paneli je možné aj nastavenie pozície jednotlivých gúľ.

Ako už bolo spomenuté vyššie, tento režim umožňuje rendering hociktorého okamihu hry. Postup je nasledovný:

- stlačenie klávesy *Enter* nás prepne do režimu nastavenia kamery a času (stlačením *Escape* sa vrátime späť)
- po nastavení kamery opäť treba stlačiť Enter, po ktorom sa zobrazí okno s nastavením renderu (výsledné rozlíšenie obrázka, antialiasing, mäkké tiene, ...), stlačením *Escape* resp. tlačítka *Cancel* sa vrátime spať do predchádzajúceho režimu nastavenia kamery a času
- rendering začne prebiehať okamžite po kliknutí na tlačítko *OK*, pričom v okne aplikácie sa priebežne zobrazuje vyrenderovaný obrázok
- výsledný obrázok sa uloží do súboru `output.png`
- rendering je možné prerušiť klávesou *Escape*

Panel nastavenia renderu obsahuje rôzne voľby ovplyvňujúce kvalitu výsledného obrázku, jednotlivé efekty sa zapínajú príslušným zaškrtávacím tlačítkom, po jeho zaškrtnutí sa sprístupní jeho nastavenie. Nasledujúca tabuľka popisuje možnosti nastavenia renderu.



Obrázok 42: Program - nastavenie renderu

Názov	Popis
Resolution	voľba výsledného rozlíšenia, buď jedno z prednastavených alebo vlastné
Use kd-tree	zapína použitie kd-stromu
Antialiasing	povoľuje antialiasing 4x4
5d raytracing	zapína 5d raytracing
Samples	počet primárnych lúčov na jeden pixel obrazu
Multithreading	zapína viacvláknový rendering
Threads	počet renderovacích vlákien
Motion blur	zapína efekt rozmazania pohybom
Samples	počet časových lúčov
Time	dĺžka uzávierky
Fast	použije urýchľovaciu masku (viz. Optimalizovaný „motion blur“)
Depth of field	zapína efekt hĺbky ostrosti
Samples	počet lúčov efektu hĺbky ostrosti
Distance	vzdialenosť roviny hĺbky ostrosti (v cm)
F-number	charakteristické číslo použitej šošovky (napr. 3.5, 11, 22, ...)
Soft shadows	nahradenie bodových zdrojov svetla plošnými (štvorcovými)
Samples	počet tieňových lúčov
Fast	zapína použitie heuristiky (viz. Mäkké tiene s predpoveďou)
Blurry reflection	zapne rozmazané odrazy na guliach
Samples	počet reflexných lúčov
Dispersion	veľkosť rozmazania odrazov

Tabuľka 2: Parametre renderu

4.3.2 Práca s príkazového riadku

Pre spustenie renderignu z príkazového riadku je potrebné zadať dva parametre programu, a to vstupný súbor a výstupný obrázok. Vstupným súborom sa myslí súbor popisujúci scénu vo formáte *RAY*. Úplný príkaz je nasledovný:

```
./TurboRAY scene.ray scene.png
```

Po spustení program okamžite začne načítavať a spracovávať vstupný súbor, ktorý popisuje všetky objekty v scéne, materiály objektov, nastavenie renderu a použité efekty. Po spracovaní tohto súboru začne aplikácia renderovať scénu. Priebežný stav sa zobrazuje v okne aplikácie. Samotný

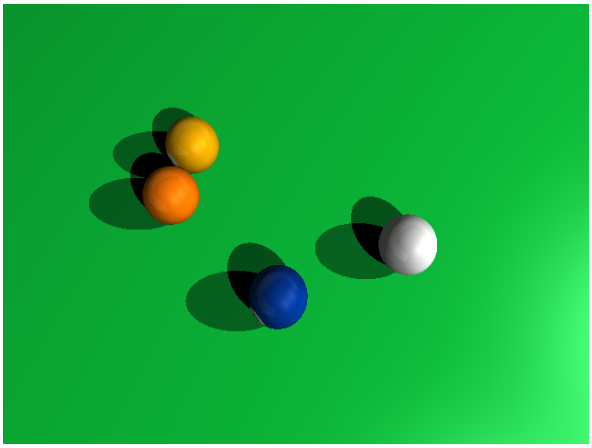
rendering je možné prerušiť klávesou *Escape*. Vyrenderovaný obrázok je uložený do súboru zadaného prostredníctvom druhého parametru aplikácie.

5 Výsledky

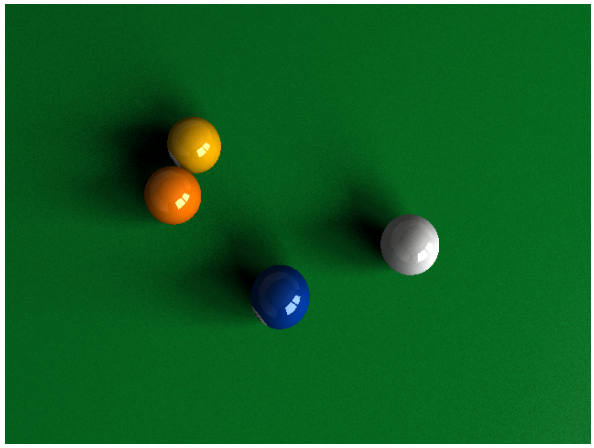
Táto kapitola prezentuje a hodnotí efekty, ktoré je možné dosiahnuť použitím distribuovaného raytracingu. Súčasne s nimi pre porovnanie ukazuje rovnaké scény renderované s použitím klasického raytracingu. Tiež porovnáva výsledky distribuovaného raytracingu pri použití urýchľovacích techník popísaných v kapitole 3 Rýchlosť a kvalita. Všetky prezentované obrázky sú vytvorené s použitím demonštračnej aplikácie TurboRAY, pričom bol použitý počítač s procesorom AthlonXP 2600, 1.5GB operačnej pamäte a operačným systémom Ubuntu Linux 9.04. Pri renderovaní bol zapnutý kd-strom, bez neho by namerané časové hodnoty boli podstatne dlhšie.

5.1 Mäkké tiene

Pridaním plošných zdrojov svetla do scény môžeme dosiahnuť realisticky pôsobiace mäkké tiene. Tento efekt nepôsobí len na oblasti polotieňov, ale aj na ostatné, kde pridáva určitú mieru náhodnosti do výpočtov Phongovho osvetľovacieho modelu. Vplyv tohto efektu je možné porovnať na obrázkoch Obrázok 43 a Obrázok 44 (oba boli renderované v rozlíšení 640x480). Kde na prvom je renderovaná scéna s dvoma bodovými zdrojmi svetla, takže každý objekt vrhá dva tiene. Na druhom obrázku je vyrenderovaná rovnaká scéna, ale bodové zdroje svetla boli vymenené za dva plošné (ktoré sa pekne odrážajú na všetkých guliach), tento obrázok pôsobí už veľmi realisticky. Na dosiahnutie takéhoto efektu sa použilo približne 48krát viac tieňových lúčov v porovnaní s ostrými tieňmi, a celkový čas výpočtu sa tiež predĺžil približne 48krát. Množstvo lúčov je vhodné zvoliť od danej scény, pretože niekedy stačí aj malý počet 16 lúčov (napr. ak je zdroj svetla malý).

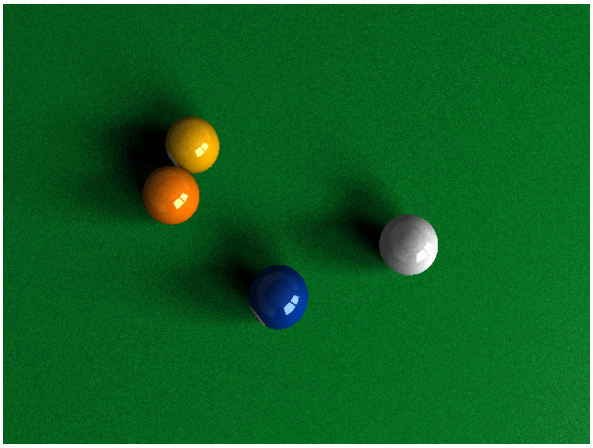


Obrázok 43: Raytracing, ostré tiene, čas 8 sekúnd, počet tieňových lúčov 642tis.

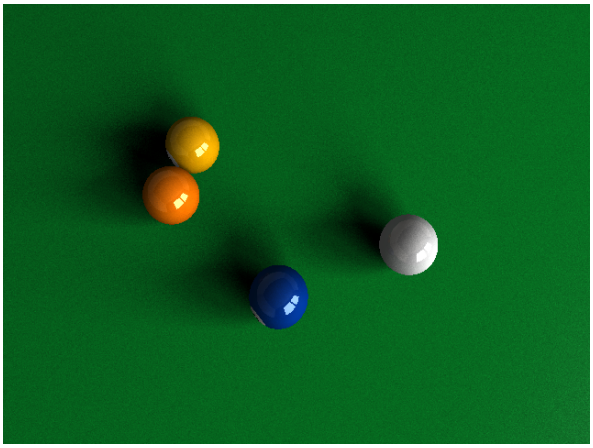


Obrázok 44: Mäkké tiene, 48 tieňových lúčov, čas 6 minút, celkový počet tieňových lúčov 30mil.

Znížiť veľké množstvo tieňových lúčov môžeme použitím heuristiky. Kvalitu výsledku a čas výpočtu potom ovplyvňuje počet sondovacích lúčov. Ideálny počet sondovacích lúčov podľa vizuálnej kvality výsledných obrázkov vychádza na číslo okolo 50% (Obrázok 46) z celkového počtu tieňových lúčov. Pri renderovaní s nižším množstvom dochádza k zmenšeniu polotieňových oblastí a zašumeniu obrazu (Obrázok 45).



Obrázok 45: Mäkké tiene s predpoveďou, počet tienových/sondovacích lúčov 48/6, čas 105 sekúnd, celkový počet tieňových lúčov 11mil.

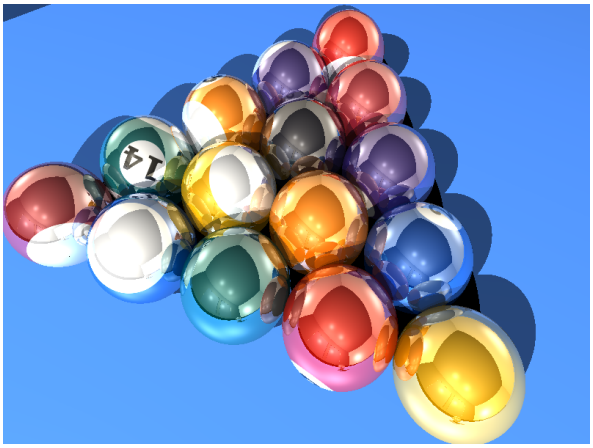


Obrázok 46: Mäkké tiene s predpoveďou, počet tieňových/sondovacích lúčov 48/24, čas 220 sekúnd, celkový počet tieňových lúčov 23mil.

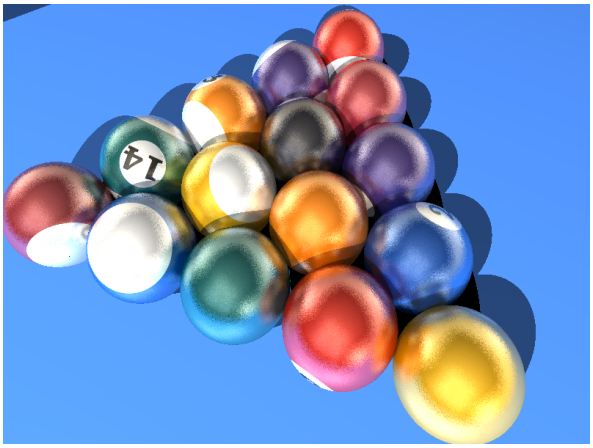
5.2 Rozostrené odrazy

Táto technika distribuovaného raytracingu posúva výsledné obrázky zas o krôčik bližšie k realistickému zobrazeniu, pretože simuluje rôzne členitý povrch materiálu. Tým zjemňuje pôvodné ostré odrazy. Pre dosiahnutie tohto efektu sa využíva distribúcia reflexných paprskov, vďaka tomu je veľmi náročná na výpočet. Počet reflexných lúčov rastie s rekurziou geometrickou radou, preto je vhodné znížiť celkový stupeň rekurzie. Tiež je vhodné obmedziť celkový počet objektov s distribuovanou reflexiou, aby nevznikalo príliš veľké množstvo odrazov.

Výsledný efekt je možné porovnať na obrázkoch 47 a 48, kde na prvom z nich majú všetky gule zrkadlové odrazy. Zapnutie distribúcie reflexných lúčov sa prejaví zmatnením zrkadlových odrazov na povrchu gúľ (materiál nadobudol chrómový lesk, viz. Obrázok 48). Samozrejme je možné výsledok ešte skrášliť pridaním ďalších efektov, vhodné je najmä zapnúť mäkké tiene a antialiasing (čím opäť vzrastie celkový čas výpočtu, aj na niekoľko hodín).



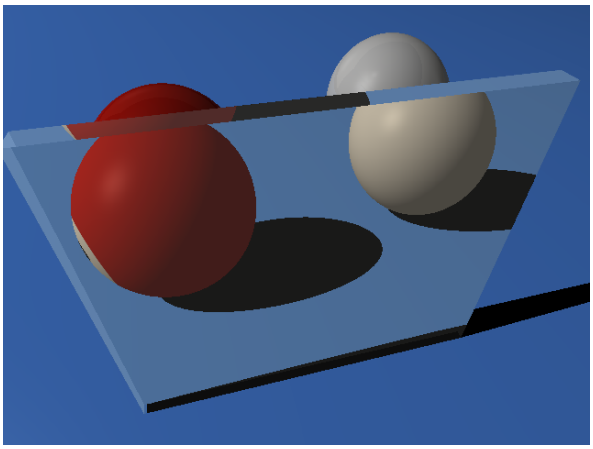
Obrázok 47: Jednoduchá reflexia, 1 reflexný, bodové zdroje svetla lúč, čas renderu 29 sekúnd



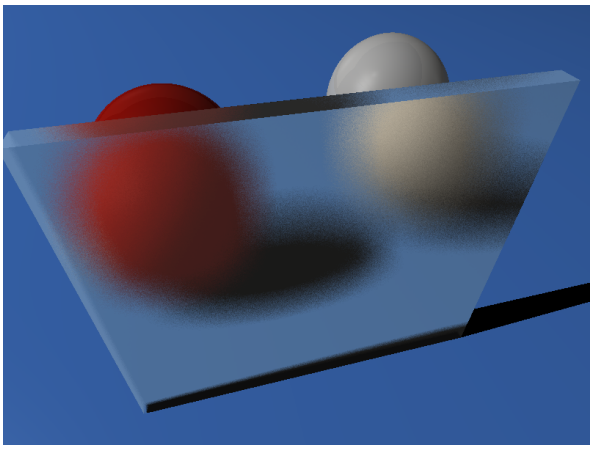
Obrázok 48: Distribuovaná reflexia, 9 reflexých lúčov, rozptyl 0.5, bodové zdroje svetla, čas renderu 180 sekúnd

5.3 Matná priehľadnosť

Ďalším z rady efektov distribuovaného raytracingu je simulácia matne priehľadného materiálu. Tento efekt je založený na podobnej distribúcii lúčov ako predchádzajúci, len namiesto reflexných lúčov distribuujeme refrakčné lúče vstupujúce do materiálu. Výsledkom je potom napríklad matné sklo, matne priehľadný plast, ... Stupeň matnosti sa reguluje veľkosťou rozptylu. Týmto efektom možné simulovať širokú škálu matnej priehľadnosti.



Obrázok 49: Jednoduchá priehľadnosť, rozlíšenie 640x480, 1 reflakčný lúč, čas renderu 15 sekúnd



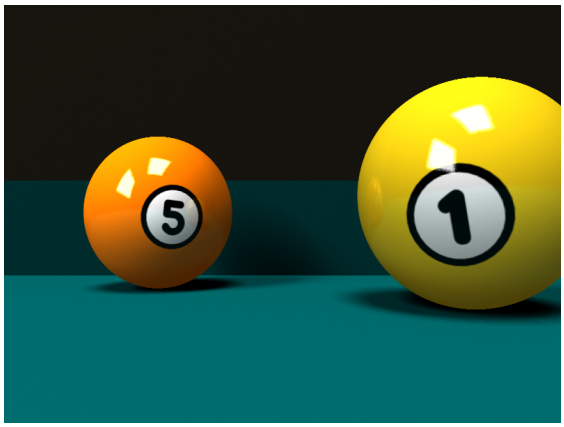
Obrázok 50: Distribuovaná refrakcia, rozlíšenie 640x480, 32 refrakčných lúčov, čas renderu 487 sekúnd

Náročnosť tohto efektu rapídne rastie s počtom priehľadných objektov za sebou a hĺbkou rekurzie, podobne ako pri predchádzajúcom efekte. Výsledok raytracingu s jednoduchou priehľadnosťou a distribuovanou je zobrazený na obrázkoch 49 a 50. Prvý obrázok zobrazuje čisto priehľadné sklo bez rozptylu refrakčných lúčov, a pôsobí trochu umelým dojmom. Druhý obrázok simuluje matnú priehľadnosť a celkovo pôsobí reálnejším dojmom, cenou je približne 32 krát dlhší čas výpočtu. Preto je vhodné matne priehľadné materiály používať minimálne, a len na objekty kde ich je vidieť (nie na všetky priehľadné objekty v scéne).

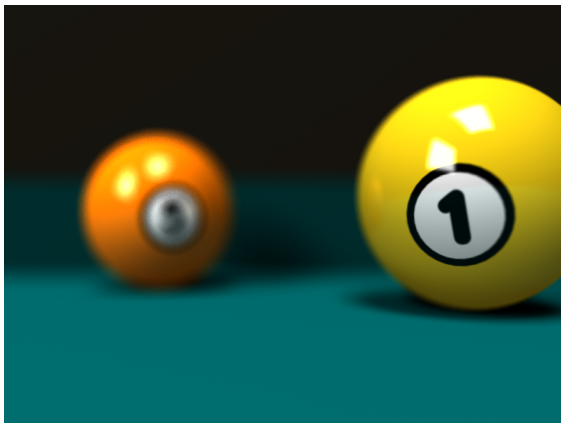
5.4 Hĺbka ostrosti

Tento efekt simuluje vlastnosti reálneho objektívu, ktorý zobrazuje ostrý obraz len v jednom bode. Hĺbka ostrosti scénam veľmi pridáva na reálnosti, ako je vidieť na nasledovných obrázkoch. Tieto obrázky sú vyrenderované vo vysokej kvalite s použitím 16 lúčov na antialiasing a 32 lúčov na hĺbku ostrosti, čo spolu dáva celkový počet 512 primárnych na jeden pixel, tiež sú zapnuté mäkké tiene s 9 lúčmi, preto je čas ich výpočtu tak obrovský. Pri tomto efekte je možné regulovať veľkosť rozmazania veľkosťou polomeru šošovky (f-číslom), pre veľkú ostrosť do diaľky sa nastavuje maximálne f-číslo, s jeho znižovaním rastie veľkosť clony a aj veľkosť rozmazania objektov stojacich v popredí a pozadí roviny ostrosti. Ďalej je možné nastaviť vzdialenosť roviny hĺbky ostrosti a tým určiť ktoré objekty sa vyrenderujú ako ostré a ktoré rozmazané (Obrázok 52 a Obrázok 53).

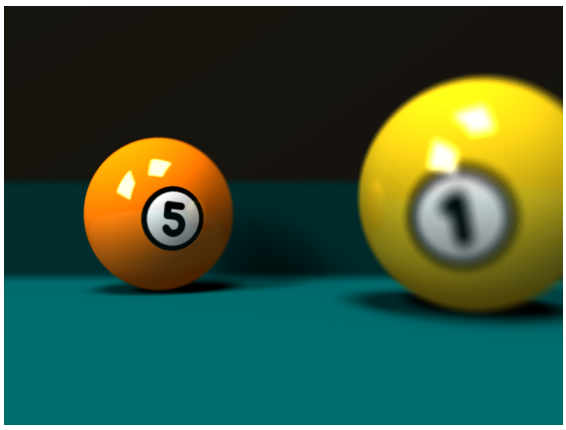
Použitie 5d raytracingu dokáže výrazne urýchliť čas výpočtu, preto je lepšie výpočet antialiasingu a efektu hĺbky ostrosti počítať súčasne, čím dosiahneme približne 3-4 násobné zrýchlenie.



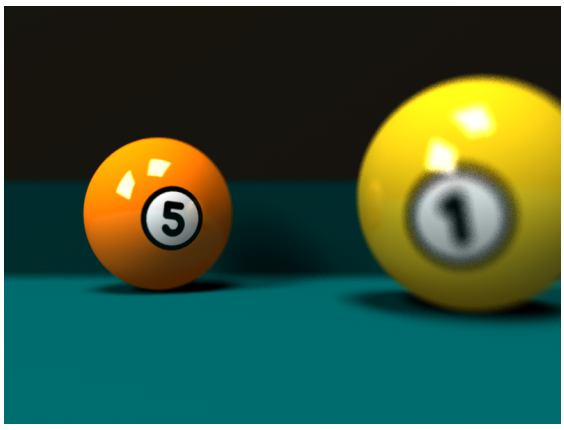
Obrázok 51: Bez efektu hĺbky ostrosti, len s antialiasingom 4x4, mäkké tiene, rozlíšenie 800x600, čas renderu 10 minút



Obrázok 52: Efekt hĺbky ostrosti (32 lúčov), zaostrenie na guľu v popredí, rozlíšenie 800x600, antialiasing 4x4, mäkké tiene, čas renderu 5 hodín



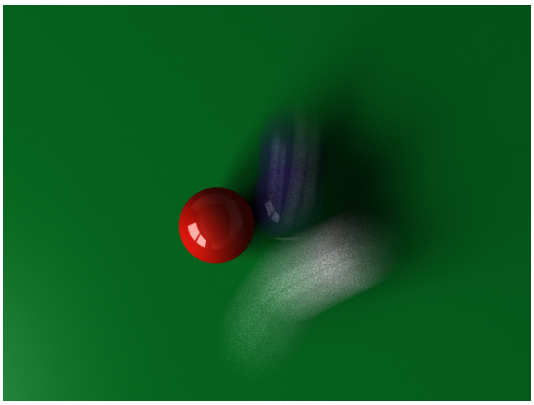
Obrázok 53: Efekt hĺbky ostrosti (32 lúčov), rovina ostrosti nastavená na guľu v pozadí, rozlíšenie 800x600, antialiasing 4x4, mäkké tiene, čas renderu 5 hodín



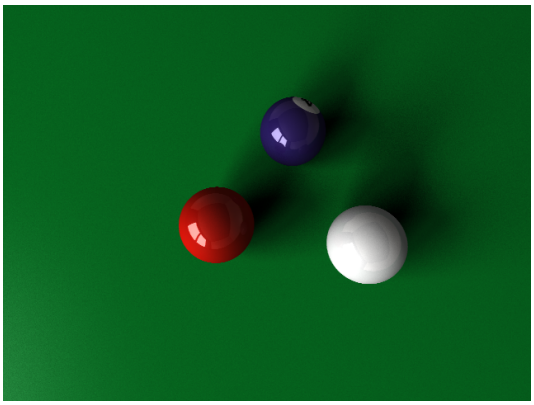
Obrázok 54: Efekt hĺbky ostrosti s antialingom renderované 5d raytracingom (128 lúčov), rozlíšenie 800x600, 4x4, čas renderu 1 hodina 20 minút

5.5 Motion blur

Tento efekt je vhodný na zdôraznenie pohybu v scéne, kde pohybujúce objekty budú svojím pohybom rozmazané podľa veľkosti ich pohybu, rýchlejšie objekty viacej, pomalšie menej. Nasledujúce dva obrázky ukazujú tú istú scénu troch gúľ, kde pohybujúca sa biela guľa narazí do stojacej fialovej gule, po tomto náraze sa biela guľa spomalí, zmení svoj smer a fialová sa dá do pohybu. Prvý je vyrenderovaný distribuovaným raytracingom so zapnutým efektom „motion blur“, na ňom je vidieť ako biela guľa narazí do fialovej, tá sa odrazí a biela guľa potom zmení svoj smer. Pre porovnanie bol druhý obrázok renderovaný klasickým raytracingom, takže vidíme len scénu v poslednom časovom okamžiku bez dynamiky.

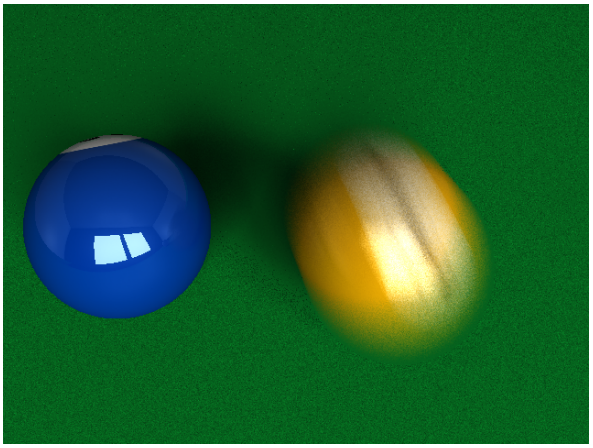


Obrázok 55: Motion blur 64 lúčov + mäkké tieňe 3 lúče, rozlíšenie 640x480

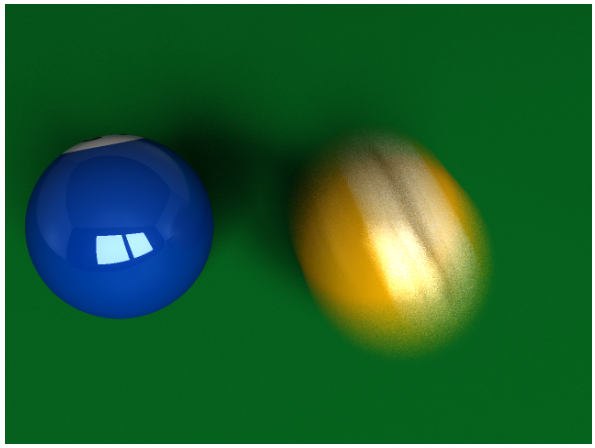


Obrázok 56: Klasický raytracing, rozlíšenie 640x480

Na kvalitu výsledného obrázku (znázornenie dynamiky objektov) veľmi vplýva počet použitých časových lúčov, nesmie ich byť málo, aby scéna nepôsobila zašumené, na druhú stranu veľký počet lúčov spomaľuje výpočet. Samotný efekt „motion blur“ dobre vyhladzuje hrany pohybujúcich sa objektov, ale hrany statických objektov ostanú ostré. Preto ak nechceme mať ostré hrany na statických objektoch, musíme zapnúť antialiasing, čo značne predĺži čas renderu (podľa počtu lúčov použitých antialiasingom). Pre urýchlenie je potom vhodné zapnúť použitie masky pokrývajúcej dynamické objekty (viz. 3.6 Optimalizovaný „motion blur“). Aj toto riešenie má svoje limity, napríklad nepočíta s lesklými a priehľadnými povrchmi (značne by predlžovali výpočet masky) a tiež statické miesta na obraze sú pri použití plošných zdrojov svetla viacej zašumené oproti zvyšku. Najvhodnejšie je však použitie 5d raytracingu so súčasne zapnutým antialiasiningom a „motion blurom“. Minimálny počet použitých lúčov by mal byť aspoň 64, ideálna hodnota je 128. Nasledujúce dva obrázky ukazujú obe spomenuté varianty.



Obrázok 57: Rendering s urýchlovacou maskou, 64 časových lúčov, 6 tieňových, čas 26 minút



Obrázok 58: 5D raytracing, 64 lúčov, 6 tieňových, čas renderu 50 minút

6 Záver

Distribuovaný raytracing je veľmi kvalitná metóda realistického zobrazenia, vychádzajúca z jednoduchého raytracingu. Dosahuje krajšie a reálnejšie výsledky napr. v podobe mäkkých tieňov, ... Cenou je však omnoho vyššia náročnosť na výpočet a komplikovanejšie nastavenie parametrov renderu. Práve v konfigurovaní je potrebné nadobudnúť určitú prax, naučiť sa odhadnúť rozumné hodnoty parametrov pre jednotlivé efekty, pretože príliš veľký počet lúčov značne predĺži čas výpočtu obrázku. Tiež si je treba dávať pozor na množstvo lúčov pri kombinovaní viacerých efektov súčasne. Pri kombinovaní efektov môžeme nižším efektom (v poradí renderu posledným) znížiť počet lúčov a pritom dosiahnuť dobrú kvalitu (napr. pri kombinácii antialiasingu a mäkkých tieňov môžeme počet tieňových lúčov znížiť z 32 na 8). Urýchlenie je možné dosiahnuť aj technikami navrhovanými touto prácou. Použitím predpovede tieňových lúčov ušetríme približne tretinu zo všetkých tieňových lúčov. Taktiež zapnutím optimalizácie efektu „motion blur“ dosiahneme zrýchlenie, ktorého veľkosť predovšetkým závisí oblasti obrazu, ktorú pokrývajú pohybujúce sa objekty a ich tiene. Pre skrátenie celkového času výpočtu pri kombinovaní efektov využívajúcich primárne lúče (napr. antialiasing + motion blur) môžeme použiť techniku 5d raytracingu, a s ňou ušetriť viac ako polovicu primárnych lúčov.

Súčasťou tejto práce je aj návrh a implementácia demonštračnej aplikácie (distribuovaného raytraceru). Tento program v sebe zahŕňa jednoduchý a distribuovaný raytracer ako aj spomínané urýchľovacie techniky. Architektúra demonštračnej aplikácie bola navrhnutá s ohľadom na ďalšiu rozšíriteľnosť (napr. o NURBS plochy, procedurálne textúry, ...). Vhodná by mohla byť optimalizácia výpočtov s použitím inštrukcií SSE.

Distribuovaný raytracing je perspektívnou metódou realistického zobrazenia, kvalitný výsledok síce vyžaduje veľký čas výpočtu, ale s neustále rastúcim výkonom počítačov ustupuje toto obmedzenie do pozadia.

Literatúra

[1] Robert L. Cook, Thomas Porter , Loren Carpenter, Distributed ray tracing, Proceedings of the 11th annual conference on Computer graphics and interactive techniques, p.137-145, January 1984

[2] Havran Vlastimil: Heuristic Ray Shooting Algorithms, dizertačná práca, FEL CVUT v Prahe, 2001

[3] Wald Ingo: Realtime Ray Tracing and Interactive Global Illumination, dissertation thesis, Computer Graphics Group Saarland University, Saarbrücken, Germany, 2004

[4] Shirley Peter, Wang Changyaw: Distribution Ray Tracing: Theory and Practice , Proceedings of the Third Eurographics Workshop on Rendering, 1992

[5] Arvo James, Dutre Phil, Keller Alexander, Wann Jensen Henrik, Owen Art, Pharr Matt, Shirley Peter: Monte Carlo Ray Tracing, Siggraph 2003 Course 44, 2003

[6] Slovák Radek: Distribuované sledování paprsku, bakalářská práce, Brno, FIT VUT v Brně, 2008

[7] Allen Martin : Distributed Ray Tracing [online]. 2.1.2009 [cit. 2.1.2009]. Dostupný z WWW: <http://web.cs.wpi.edu/~matt/courses/cs563/talks/dist_ray/dist.html>

[8] Hart C. John: Distributed Ray Tracing [online]. 6.1.2009 [cit. 5.1.2009]. Dostupný z WWW: <http://luthuli.cs.uiuc.edu/~daf/courses/ComputerGraphics/Week3/distributed-final.pdf>

[9] Biker Jacco: Raytracing Topics & Techniques [online]. 1.1.2009 [cit. 1.1.2009]. Dostupný z WWW:<http://www.flipcode.com/archives/Raytracing_Topics_Techniques-Part_1_Introduction.shtml>

[10] Peringer Petr: Modelování a simulace Texty k přednáškám pro předmět IMS [online]. 2.1.2009 [cit. 2.1.2009]. Dostupný z WWW: <http://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf>

[11] Obtulovič Peter: Základy teórie pravdepodobnosti [online]. 6.1.2009 [cit. 3.1.2009]. Dostupný z WWW: <www.fem.uniag.sk/cvicenia/ksov/obtulovic/BIOŠTATISTIKA/prednaska3.ppt>

[12] prispievatelia Wikipédia: Fotoaparát [online]. 1.5.2009 [cit. 1.5.2009]. Dostupný z WWW: <http://sk.wikipedia.org/w/index.php?title=Fotoapar%C3%A1t&oldid=2244267>

Zoznam príloh

A - Popis formátu RAY

B - CD s programom, zdrojovými kódmi, dokumentáciou a výstupmi

Príloha A

Popis formátu RAY

RAY formát je súbor obsahujúci kompletný popis scény a parametrov renderu v XML jazyku. Každý XML dokument sa skladá z jedného koreňového elementu, v tomto prípade je to element `<TurboRAY>` pomenovaný podľa názvu aplikácie. Tento koreňový element obsahuje vnorené elementy popisujúce objekty scény, použité materiály a nastavenie renderu.

```
<TurboRAY>
  <Materials>
    ... popis materiálov ...
  </Materials>
  <Scene>
    ... popis objektov ...
  </Scene>
  <Render width="800" height="600">
    ... popis kamery a efektov ...
  </Render>
</TurboRAY>
```

Sekcia `<Materials>` obsahuje iba jeden druh elementov, a to elementy `<Material>`. Element `<Material>` udáva názov a charakteristické vlastnosti materiálu vhodné pre distribuovaný raytracer. Okrem základných vlastností spoločných s jednoduchým raytracerom (napr. difúzna zložka) popisuje distribúciu (`dispersion` a `samples`) reflexných a refrakčných lúčov. Parameter `highlight` udáva zosilnenie svetelných zdrojov pri odrazoch a lome svetla.

```
<Materials>
  <Material name="ball">
    <Ambient  r="0.2" g="0.2" b="0.2"/>
    <Diffuse  r="0.6" g="0.6" b="0.6"/>
    <Specular r="0.5" g="0.5" b="0.5"/>
    <Reflection r="0.1" g="0.1" b="0.1"
                samples="16" dispersion="0.2" highlight="9"/>
    <Refraction r="0.0" g="0.0" b="0.0" index="1.0"
                samples="0" dispersion="0" highlight="0.0"/>
  </Materials>
    ... materiály ...
</Materials>
```

K samotnému popisu scény je určená sekcia `<Scene>`, táto môže obsahovať nasledujúce elementy:

`Triangle` - trojuholník zadaný troma vrcholmi, normálami a textúrovacími súradnicami

`Sphere` - guľa

`Model` - model načítaný z OBJ súboru, musí byť zložený z trojuholníkov

`PointLight` - bodový zdroj svetla

`SphereLight` - guľový zdroj svetla

`RectangleLight` - štvorcové svetlo

Jednotlivé objekty sa rozmiestňujú posuvom a rotáciami, tieto transformácie sa zapisujú do teľa konkrétnych objektov. Podporované transformácie sú:

```
<Translate x="5" y="0" z="0"/>  - posun objektu
<RotateX angle="15"/>           - rotácia okolo osi X
<RotateY angle="250"/>          - rotácia okolo osi Y
<RotateZ angle="-56"/>          - rotácia okolo osi Z
```

Popis objektov:

`<Triangle>` – reprezentuje trojuholník zadaný troma bodmi, každému je možné určiť normál a textúrovacie súradnice.

```
<Triangle material="mat1">
  <Vertex x="-10.0" y="10.0" z="5"/>
  <Vertex x="-10.0" y="20.0" z="5"/>
  <Vertex x="-10.0" y="10.0" z="5"/>
</Triangle>
```

`<Sphere>` – reprezentuje guľu so stredom $(0,0,0)$, pre umiestnenie je vhodné použiť transformácie objektu.

```
<Sphere radius="5.3" material="mat1">
  <Translate x="86.3" y="-5" z="9"/>
</Sphere>
```

`<Model>` – umožňuje vloženie geometrie modelu vytvoreného v 3d modelovacom programe, napríklad Blender alebo 3D Studio Max. Tento model musí byť vo formáte OBJ a zložený z trojuholníkov.

```
<Model file="room.obj" material="room"/>
```

`<PointLight>` – je bodovým zdrojom svetla, jeho jediným atribútom je farba.

```
<PointLight>
  <Color r="1.0" g="1.0" b="1.0"/>
</PointLight>
```

`<SphereLight>` – guľový zdroj svetla s počiatkom $(0,0,0)$, okrem farby je potrebné zadať jeho polomer a počet k nemu vysielaných tieňových lúčov.

```
<SphereLight radius="5.3" samples="32">
  <Color r="1" g="1" b="1"/>
```

```
  <Translate x="86.3" y="-5" z="9"/>
</SphereLight>
```

`<RectangleLight>` - štvorcový zdroj svetla zadaný štyrmi vrcholmi, ďalej farbou počtom k nemu vysielaných tieňových lúčov.

```
<RectangleLight samples="32">
  <Vertex x="-202" y="120" z="-80"/>
  <Vertex x="-202" y="120" z="80"/>
  <Vertex x="-202" y="0" z="80"/>
  <Vertex x="-202" y="0" z="-80"/>
  <Color r="1" g="1" b="1"/>
</RectangleLight>
```

Posledná sekcia v RAY súbore začína elementom `<Render>`, a nastavuje parametre renderu ako je počítané rozlíšenie, antialiasing, zapína použitie jednotlivých efektov:

```
<Render width="800" height="600" aa="0" raytrace5d="1">
  ... efekty ...
</Render>
```

`aa`	- zapnutie antialiasingu 4x4
`raytrace5d`	- zapnutie 5d raytracingu
`samples`	- počet lúčov 5d raytracingu
`threads`	- počet použitých vlákien

Vnútro sekcie `<Render>` môže obsahovať element `<DepthOfField>` zapínajúc efekt hĺbky ostrosti, jeho syntax je nasledovná:

```
<DepthOfField distance="50" samples="32" fnumber="11">
```

`distance`	- vstialenosť roviny hĺbky ostrosti
`samples`	- počet použitých lúčov
`fnumber`	- charakteristické číslo použitej šošovky