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

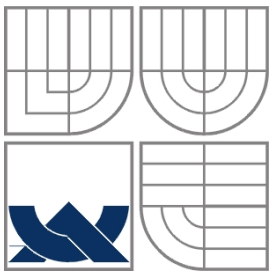
DISTRIBUTED RAY TRACING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

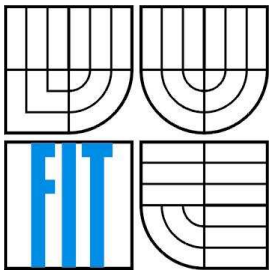
AUTOR PRÁCE
AUTHOR

BC. VÁCLAV HOŠEK

BRNO 2008



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

DISTRIBUTED RAY TRACING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc.Václav Hošek
AUTHOR

VEDOUCÍ PRÁCE — Ing. Adam Herout, Ph.D.
SUPERVISOR

BRNO 2008

Abstrakt

Tato práce se zabývá realistickým zobrazováním počítačových scén a to metodou Distributed Ray Tracing. Tato metoda jako první řešila plošná světla, měkké stíny, hloubku ostrosti a rozmazání scény pohybem.

Klíčová slova

Distributed Ray Tracing, zobrazovací metoda, Monte Carlo, stochastický výpočet integrálu, zobrazovací rovnice, zdroje světel a materiály v počítačové grafice, hloubka ostrosti, rozmazaní pohybem.

Abstract

Distributed Ray Tracing, also called distribution ray tracing and stochastic ray tracing, is a refinement of ray tracing that allows for the rendering of "soft" phenomena, area light, depth of field and motion blur.

Keywords

Distributed Ray Tracing, illumination algorithm, Monte Carlo, Monte Carlo integration, rendering equation, light and material in computer grafics, depth of field, motion blur

Citace

Václav Hošek: Distributed Ray Tracing. Brno, 2008, diplomová práce, FIT VUT v Brně.

Distributed Ray Tracing

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.

……
Bc. Václav Hošek
15.května 2008

Poděkování

Ing. Adamu Heroutovi ,Ph.D. za vedení této semestrální práce.

© Václav Hošek, 2008.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

Tato práce se zabývá problematikou realistických zobrazovacích metod v počítačové grafice. Počítačová grafika již není pouze doménou vědy, stává se součástí každodenního života. Setkáváme se s ní v animovaných filmech, počítačových hrách, také v lékařství, biologii, chemii i biochemii, optické mikroskopii, strojírenství, automobilovém průmyslu, ve stavebním průmyslu. Všechny tyto a i mnohé další obory využívají počítačovou grafiku jako nástroje pro vizualizaci svých výsledků, výpočtů, simulací či návrhů. Asi je každému jasné, že se všichni snažíme o výsledky, jež se co nejvíce podobají skutečnosti. A právě za tím, aby se námi navrhnuté výsledky představily co nejvíce skutečně, za tím stojí realistické zobrazovací algoritmy. Já v této práci představím jednu z prvních realistických zobrazovacích metod – Distributed Ray Tracing.

Diplomová práce navazuje na semestrální projekt a skládá se ze dvou částí. V první je popis teoretického zázemí pro realizaci Distributed Ray Tracingu. Druhá část vychází z praktických zkušeností a problémů při vlastní implementaci zobrazovacího algoritmu.

1.1 Historie zobrazovacích metod

První algoritmy, které se zabývaly výpočtem stínování, vznikly se vznikem rastrové grafiky. Tyto algoritmy přidělovaly každému polygonu jednu barvu. Dnes jsou známy pod termínem konstantní stínovaní. Konstantní stínování již nabízelo představu o tvaru, orientaci a vzdálenosti objektu od pozorovatele.

Větší realističnost přineslo Gouraudovo (1971) a později Phongovo (1975) stínování. Oba algoritmy využívají interpolaci. Oba algoritmy se díky své jednoduchosti a efektivitě dosud hojně používají.

Ray Tracing. V roce 1979 navrhl Turner Whitted algoritmus Ray Tracing, který se posléze stal jedním z nejrozšířenějším z algoritmů počítačové grafiky. Whitted navázal na algoritmus Ray Casting, který vysílal paprsky směrem od pozorovatele do scény. Klasická verze algoritmu Ray Tracing, jak ji prezentoval Whitted, postupovala podobně a sledovala jeden paprsek pro každý pixel v obraze. Na rozdíl od Ray Castingu však paprsek nekončí v místě průsečíku se scénou, ale generuje další paprsky: stínový paprsek, paprsek lomu a paprsek odrazu. Stínový paprsek ověřuje viditelnost mezi průsečíkem se scénou a světelným zdrojem. Pomocí tohoto paprsku zjistíme, zda průsečík leží ve stínu nebo je přímo osvětlen. Paprsek lomu simuluje lom světla a paprsek odrazu slouží pro výpočet zrcadlového odrazu. Whittedův Ray Tracing sice nabídl mnohem vyšší míru skutečnosti než

předchozí algoritmy, ale přesto trpí nedostatky. Oproti skutečnosti je moc přesný, ostrý a tím nepůsobí věrohodně.

Radiozita. Algoritmus Radiozita vznikl, aby řešil problém nepřímého osvětlení. Samotný algoritmus vychází ze zákona zachování energie, předpokládá tedy, že pracujeme s energeticky uzavřenou scénou. Scéna se rozdělí na plošky (patches), přičemž každá ploška je charakterizována hodnotou „vlastní energie“ a hodnotou reflektivity. Jedná se o algoritmus víceprůchodový. V každém průchodu algoritmus spočítá množství světla dopadající na plošku ze všech ostatních plošek ve scéně.

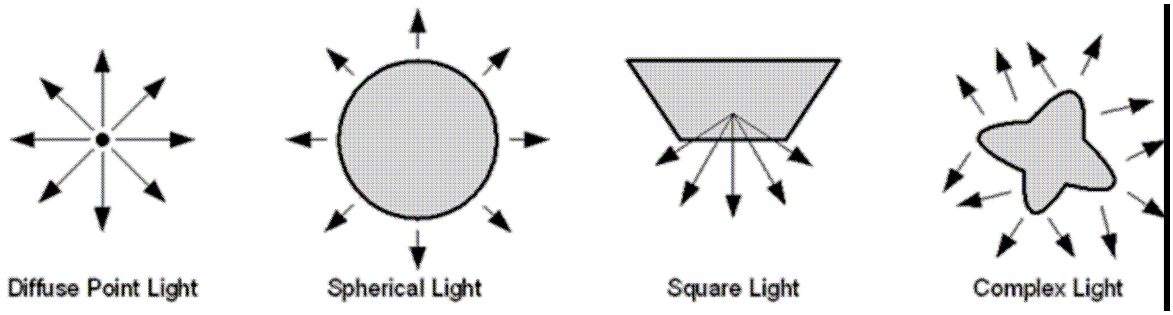
Distributed Ray Tracing byl představen v roce 1984. Na rozdíl od Whittedova Ray Tracingu tento algoritmus vysílá z jednoho pixelu hned několik paprsků, které jsou náhodně rozmístěny po celé ploše pixelu. Tento algoritmus umožnil vyšší realističnost, zejména výpočet měkkých stínů, anizotropní (glossy) odrazy, hloubku ostrosti, či rozmazání při pohybu. Poprvé jsou použita také plošná světla. Distributed Ray Tracing poprvé odhaduje hodnotu lokálního integrálu pomocí metody Monte Carlo a sleduje sekundární paprsky s ohledem na tvar BRDF.

Zobrazovací rovnice. Vznik reálných algoritmů umožnila zobrazovací rovnice. Tuto rovnici představil v roce 1986 James T. Kajiya, který navázal na Distributed Ray Tracing. Zobrazovací rovnice dala podnět pro vznik algoritmu Monte Carlo Ray Tracing, Path Tracing a další. Zde bych popis historie ukončil.

2 Reprezentace reality v počítačových scénách

2.1 Světelné zdroje

Bodový zdroj světla - světlo vzniká v jednom bodě a šíří se do celého prostoru rovnoměrně(prostor koule). Jde o nejjednodušší zdroj, který však v reálném světě spíše nenajdeme. Způsobuje ostré stíny.
Plošný zdroj světla - světlo vzniká na celé ploše a vyzařuje energii do polokoule obklopující plochu. Vznikající světelná intenzita se na ploše světla může měnit. Tento zdroj lze věrohodně použít pro nahrazení reálných zdrojů světel jako jsou okna, plošná světla, atd…
Kulový zdroj světla - světlo vzniká na povrchu koule a je vyzařováno do všech směru rovnoměrně. Často se používá k snadnému dosažení měkkých stínů. V reálnem světě jej můžeme přirovnat k slunci, žárovce.
Obecný zdroj světla – jde o komplexní pojetí světelných zdrojů. Takovýto zdroj lze popsat jeho geometrií a funkcí pro vyzařování světelné energie.



Obr. 1 –Zdroje světel

2.2 Povrch objektů, materiály

Dá se říci, že počítačová grafika zjednodušuje fyzikální vlastnosti materiálů na tři základní. Difusní, odrazový(spekulární) a průhledný(refrakční) materiál.

2.2.1 Difusní materiály

Jedná se o materiály, které energii více pohlcují než vyzařují. Energii vyzařují do všech směrů s téměř rovnoměrnou intenzitou. Velmi dobře tento druh materiálů simuluje matné plasty, omítku zdí a podobné.

2.2.2 Odrazové materiály

Energii odráží, nepohlcují ji. Energie je odražena podle zákona odrazu. Pomocí tohoto materiálu modelujeme zrcadla, kovové povrchy, všechny povrchy co odráží okolí.

2.2.3 Průhledné materiály

Energii odráží, lámou podle Snellova zákona lomu. Tímto materiálem řešíme objekty průhledné jako je sklo, voda, plasty a další.

2.2.4 Obecné materiály, funkce BRDF

Téměř žádný materiál není jen difusní, odrazivý či průhledný, vždy se jedná o materiál nesoucí kombinaci těchto základních vlastností. Proto pro popis izotropních materiálů zavedl Nicodemus obousměrnou distribuční funkci odrazu neboli BRDF (Bidirectional Reflectance Distribution Function). Jde o funkci, která dává do vztahu odražený jas ve směru r a příchozí osvětlení ve směru i v bodě dopadu.

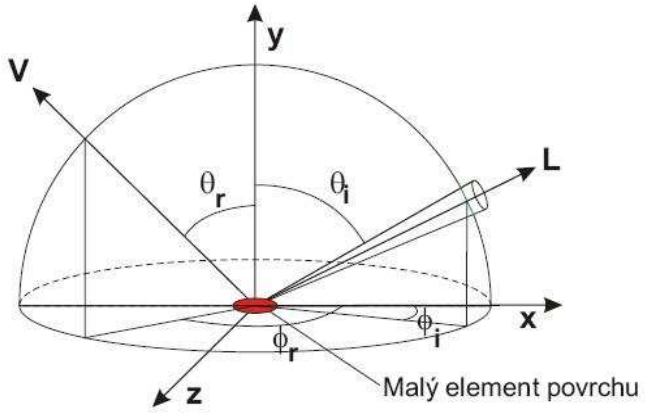
$$f_\lambda(\phi_r, \theta_r, \phi_i, \theta_i) = \frac{I_{r\lambda}(\phi_r, \theta_r)}{I_{i\lambda}(\phi_i, \theta_i).\cos\theta_i.d\omega_i}[sr^{-1}]$$

(ϕ_i, θ_i) je úhel dopadu. (ϕ_r, θ_r) je úhel odrazu.

$\cos\theta_i$ cosinus úhlu mezi směrem dopadajícího paprsku a normálou povrchu

$d\omega_i$ prostorový úhel podél směru dopadajícího paprsku.

Kde f je funkce BRDF pro každou vlnovou délku jiná, Ir je vyzářený jas, Ii je jas přicházející. Pokud se na funkcí lépe podíváme, všimneme si dělení odcházejícího jasu ku příchozímu. Nejedná li se o zdroj světla, musí být hodnota funkce BRDF menší nebo rovna jedné.



Obr. 2 – BRDF funkce

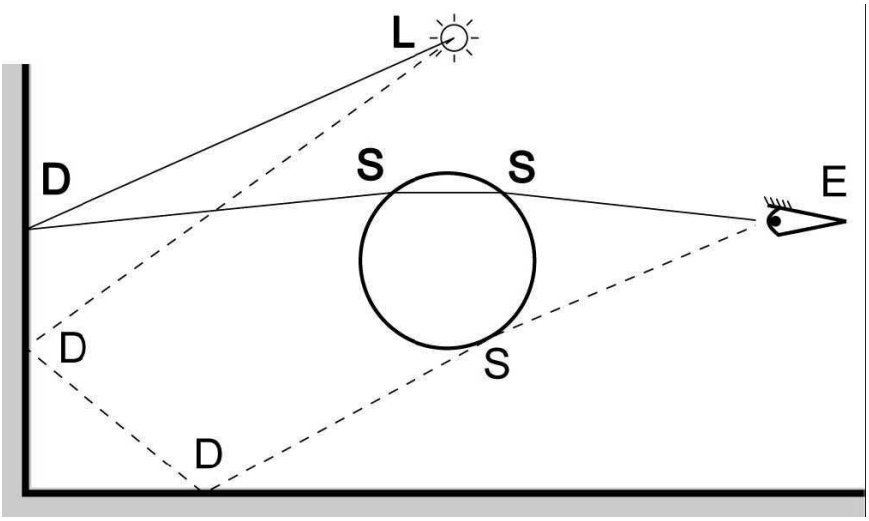
Další důležitou vlastností BRDF je **Helmholtzův princip** reciprocity, podle kterého je BRDF nezávislá na směru příchozího světla.

$$f_\lambda(\phi_r, \theta_r, \phi_i, \theta_i) = f_\lambda(\phi_i, \theta_i, \phi_r, \theta_r)$$

Tato vlastnost umožňuje sledovat světlo nezávisle jak od zdrojů, tak od pozorovatele.

2.2.5 Průchod světla scénou

Heckbert pro popis světelné cesty ve scéně zavedl notaci využívající vlastností materiálů. V této notaci mají následující symboly tento význam



Obr. 3 – Heckbertův popis světelné cesty

L – zdrojové světlo (light)

E – oko (eye)

S – zrcadlový odraz či lom (specular)

D – difusní odraz (diffuse)

Pomocí této notace lze dobře popsat vlastnosti jednotlivých zobrazovacích metod. Notace má stejný princip zápisu jako regulární výrazy. Pro plně globální osvětlovací metody platí L(D|S)*E.

2.2.6 Přehled základních metod

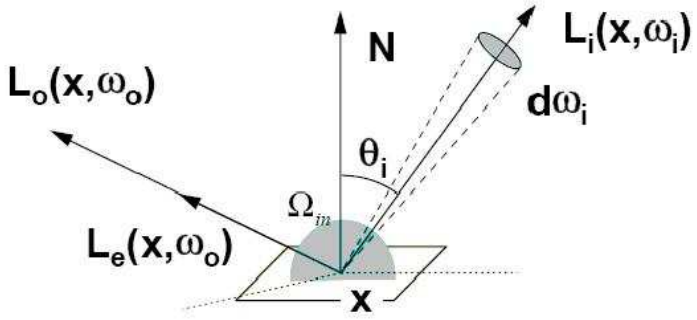
OpenGL	LDE (neumí zrcadla, lomy, difusní osvětlení)
Ray Tracing	LD?S*E (neumí kaustiky)
Distributed Ray Tracing	LD?S*E (neumí kaustiky)
Radiozita	LD*E (neumí zrcadla, lomy)
Path Tracing	L(D|S)*E
Photon Tracing	L(D|S)*E

2.3 Zobrazovací rovnice

Zobrazovací rovnice pojímá osvětlení v celé scéně, je hojně využívaná v globálních osvětlovacích metodách. Lze ji použít pro výpočet výstupního jasu v jakékoli části scény. Rovnice dává do vztahu výstupní jas Lo jako součet odraženého jasu Lr a jasu vzniklého vlastním zářením Le.

$$L_o(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) \int_{\Omega} f_r(x,\omega_i,\omega_o) L_i(x,\omega_i) \cos\theta_i \, d\omega_i$$

kde Lo je odchozí jas, Le je jas vyzářený z bodu x (jen u světelných zdroju), fr je distribuční funkce BRDF, Li je příchozí jas do bodu x.



Obr. 4 – Vizualizace zobrazovací rovnice

Jelikož světlo se při průchodu scénou mnohokrát odráží, je vhodné si zobrazovací rovnici představit rekurzivně. Pomocí toho pak budeme schopni vyhodnotit jas v jakémkoli místě na základě jasu odraženého od všech okolních objektů. Jas odražený získáme sčítáním osvětlení zdroje s jasem z prvního odrazu až v obecném případě n-tého odrazu. Příspěvku prvnímu až n-tému odrazu se pak souhrnně říká nepřímé osvětlení.

3 Metody Monte Carlo

Metody Monte Carlo představují široce používanou třídu algoritmů pro simulaci chování různých matematických a fyzikálních systémů. Termín Monte Carlo vznikl v roce 1940 a označoval metody, které prováděly náhodné statistické vzorkování s využitím výpočetní techniky. Tyto metody vynalezli pro simulaci pohybu neutronů Stanislaw Ulam, John von Neumann a Nicholas Metropolis, kteří se podíleli na vývoji nukleárních zbraní. Předpokládejme, že chceme vyřešit složitý matematický problém, např. výpočet vícerozměrného integrálu. Jelikož jsou metody Monte Carlo založeny na provádění náhodných experimentů nad jistou doménou, řešil by se výpočet integrálu definováním takové náhodné proměnné, jejíž předpokládaná hodnota je řešením problému. Nad touto náhodnou proměnnou se provede vzorkování, ze vzorků se provede průměr a výsledek je odhadem očekávané hodnoty této proměnné. Výsledná očekávaná hodnota je pak aproximací skutečného řešení problému.

3.1 Výhody a nevýhody metody

Hlavní výhodou algoritmů Monte Carlo je jejich konceptuální jednoduchost. Pokud nalezneme odpovídající náhodnou proměnnou, stačí provést vzorkování a získáme odhad výsledku. Další výhodou je jejich aplikovatelnost na širokou škálu problémů.

Nevýhodou je pomalá konvergence. I když byly vyvinuty různé techniky pro snížení rozptylu a urychlení konvergence, tak obecně metody Monte Carlo konvergují poměrně pomalu a není vhodné je používat, pokud existuje jiná alternativa. V případě složitějších problémů, např. vícerozměrných integrálů, ovšem jiné algoritmy často neexistují nebo konvergují ještě pomaleji. Typickým příkladem pro použití metody Monte Carlo je výpočet výše zmiňované zobrazovací rovnice.

3.2 Matematické pozadí a terminologie

Před tím, než se pustíme do integrovaní pomocí metody Monte Carlo, musíme zavést několik důležitých pojmů a definicí jako je spojitá náhodná veličina, hustota rozdělení, distribuční funkce, střední hodnota a rozptyl.

3.2.1 Náhodná veličina a její popis

Náhodná veličina je důležitým pojmem teorie pravděpodobnosti a matematické statistiky. Přiřazením číselných hodnot elementárním jevům nebo výsledkům realizací pokusů dostáváme náhodné veličiny. Náhodné veličiny lze definovat jako zobrazení, které přiřazuje každému jevu jevového pole určité číslo a určitou pravděpodobnost. Náhodná veličina je určena rozdělením pravděpodobnosti. Náhodné veličiny mohou být diskrétní nebo spojité. Pravděpodobnostní chování náhodných veličin lze popsat

mnoha způsoby. Jedním z nejobvyklejších je popis funkcí hustoty pravděpodobnosti, jejíž tvar podává obraz o důležitých vlastnostech rozdělení. Další obvyklou formou je distribuční funkce. Oba způsoby popisu charakterizují rozdělení náhodných veličin úplně, tedy např. pokud mají dvě veličiny stejné distribuční funkce, mají i stejná rozdělení a naopak.

3.2.2 Hustota pravděpodobnosti a distribuční funkce

3.2.2.1 Hustota pravděpodobnosti

Rozdělení pravděpodobnosti spojité náhodné veličiny se určuje prostřednictvím funkce, kterou označujeme jako hustota rozdělení pravděpodobnosti (hustota pravděpodobnosti).

Je-li ρ(x) hustota pravděpodobnosti spojité náhodné veličiny X, pak platí

$$\int_{\Omega} \rho(x)dx = 1$$

kde Ω je definiční obor veličiny X. Pro hodnoty x mimo definiční obor Ω je hustota pravděpodobnosti nulová, tzn. ρ(x) = 0 pro x nepatří do Ω.

Ze znalosti hustoty pravděpodobnosti ρ(x) lze určit pravděpodobnost, že náhodná veličina X bude mít hodnotu z intervalu < x1,x2>, tedy

$$P[x_1 \leq X \leq x_2] = \int_{x_1}^{x_2} \rho(x)dx$$

3.2.2.2 Distribuční funkce spojité veličiny

Pro spojitou náhodnou veličinu s hustotou pravděpodobnosti ρ(x) lze definovat distribuční funkci vztahem

$$F(x) = \int_{-\infty}^{x} \rho(x)dx$$

Vlastnosti distribuční funkce:

$$F(-\infty) = 0; F(\infty) = 1$$

$$\rho(x) = \frac{dF(x)}{dx}$$

3.2.3 Charakteristiky náhodné veličiny

Charakteristiky náhodné veličiny jsou vhodně vybrané číselné údaje, které shrnují základní informace o rozdělení pravděpodobnosti náhodné veličiny. Charakteristiky nám o náhodné veličině poskytují pouze základní a hrubou představu, neboť charakteristiky (obvykle) nepostačují k jednoznačnému popisu rozdělení pravděpodobnosti. Naproti tomu rozdělení pravděpodobnosti sice poskytuje jednoznačný popis náhodné veličiny, není však dostatečně přehledné.

Střední hodnotu dané náhodné veličiny můžeme určit jako integrál

$$E(f(x)) = \int_{\Omega} f(x)\rho(x)dx$$

Rozptyl jednorozměrné náhodné veličiny je definován jako

$$D(y) = E([y - E(y)]^2) = E(y^2) - E^2 y$$

Chyba měření je dána standardní odchylkou

$$\sigma(y) = \sqrt{D(y)}$$

3.3 Integrace pomocí metody Monte Carlo

V této části se podíváme, jak pomocí metody Monte Carlo určit hodnotu integrálů.

Naším úkolem je spočítat určitý integrál reálné funkce obecně *n* proměnných na určité oblasti.

$$I = \int_{a_n}^{b_n} \ldots \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x_1, x_2, \ldots, x_n) dx_1 dx_2 \ldots dx_n$$

Pro odhad střední hodnoty této funkce na dané oblasti platí, že

$$E(f(x_1, x_2, \ldots, x_n)) = \frac{I}{(b_1 - a_1)(b_2 - a_2)(b_n - a_n)} = \frac{I}{V}$$

kde *V* vyjadřuje v podstatě objem dané oblasti, platí nasledující

$$I = E(f(x_1, x_2, \ldots, x_n))V$$

Monte Carlo metoda integrace vychází právě z potřeby určení střední hodnoty funkce *f*. A to takto

$$E(f(x_1, x_2, \ldots, x_n)) = \frac{1}{N} \sum_{i=1}^{N} f(\varepsilon_1, \varepsilon_2 \ldots, \varepsilon_n)$$

kde *N* je námi zvolený počet bodů (čím více tím lépe) a $\varepsilon_i = a_i + \gamma_i (b_i - a_i)$

tedy *Ei* je náhodně zvolené číslo na intervalu <a,b> (jestliže *yi* je náhodné číslo z **rovnoměrného rozdělení** v intervalu <0, 1>.

Zjednodušeně lze říci, že střední hodnotu funkce *f* na dané oblasti spočítáme tak,že do této oblasti náhodně rovnoměrně nastřílíme určitý počet (*N*) bodů a z funkčních hodnot funkce *f* v těchto bodech určíme aritmetický průměr.

Pokud jsou vzorkovací body vybírány náhodně dle hustoty pravděpodobnosti p. Pak platí

$$I = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\varepsilon_1, \varepsilon_2 ..., \varepsilon_n)}{p(\varepsilon_1, \varepsilon_2 ..., \varepsilon_n)}$$

Přitom ale hustota pravděpodobnosti p musí být nenulová tam, kde je funkce f nenulová.

Chyba v určení hodnoty integrálu je závislá na počtu náhodně vyslaných bodů N

$$\sigma(y) = \sqrt{\frac{k}{N}}$$

Jinak řečeno. Chceme-li snížit chybu výsledku 10 krát, musíme 100 krát zvýšit výpočetní náročnost.

3.3.1 Strategie vzorkování

Při výpočtu matematického problému pomocí metody Monte Carlo je vhodné zvolit správnou strategii výběru náhodných vzorků. Výběr vhodného vzorkování vede k menším chybám a rychlejší konvergenci ke správnému výsledku.

3.3.1.1 Vzorkování podle důležitosti

Vzorkování podle důležitosti je jedna z nejčastěji používaných metod pro redukci rozptylu. Její princip vychází z pozorování, že některé části funkce přispívají k odhadu více, než jiné části. Typicky se jedná o místa s velkou hodnotou nebo s rychlými a změnami v průběhu funkce. Cílem je zaměřit se při vzorkování na tyto oblasti. Vzorkování je řízeno hustotou pravděpodobnosti f(x), která je tvarem co nejpodobnější integrované funkci a zároveň dostatečně jednoduchá. Metoda se ukazuje jako velice efektivní v případech, že integrovaná funkce obsahuje ostré výkyvy hodnot.

3.3.1.2 Složené vzorkování

Jinou možností, jak zvýšit přesnost odhadu, je zajistit, aby vzorky byly odebírány víceméně rovnoměrně v celé doméně integrálu. Myšlenkou složeného vzorkování je rozdělit vzorkovanou doménu na několik menších a pak vyhodnocovat integrál jako součet integrálů nad jednotlivými poddoménami. Běžně je potom brán pouze jeden vzorek pro každou poddoménu. Tato metoda přináší

pro většinu funkcí citelné zlepšení přesnosti odhadu, protože rozptyl v každé poddoméně bude obvykle menší, než v celé doméně integrálu.

3.3.1.3 Kvazi-Monte Carlo metody

Místo použití náhodných vzorků je možné v Monte Carlo odhadu použít vzorky deterministicky vybrané. Výhodou takovéhoto postupu je rychlejší konvergence za určitých podmínek, nevýhodou je možnost vzniku aliasingu. Detailnější vysvětlení těchto metod najdete v [7].

3.3.2 Generování vzorků

Hlavní podmínkou úspěchu Monte Carlo metod je možnost rychlého a efektivního generování vzorků. Existuje celá řada postupù pro vzorkování rovnoměrně rozdělené náhodné veličiny na jednotkovém intervalu. Zde shrneme nejčastěji používané techniky vyvinuté pro generování vzorků jiných distribucí.

3.3.2.1 Transformační techniky

Transformační techniky jsou základní a nejčastěji používané techniky. Jejich principem je převod náhodné veličiny s danou hustotou pravděpodobnosti na jinou náhodnou veličinu, jejíž vzorkování je jednodušší. V Monte Carlo metodách zobrazování je tato technika obvykle používána pro vzorkování specifických distribucí jako je funkce odrazivosti BRDF.

3.3.2.2 Inverzní transformace

Vychází z inverzní podoby distribuční funkce cílového rozložení. U některých rozložení
nelze použít (např. když distribuční funkci nelze vyjádřit elementárními funkcemi)

3.3.2.3 Zamítací Monte Carlo

Pokud je transformační technika obtížně řešitelná, je možné použít zamítací vzorkování. Metoda je založena na generování náhodných vzorků a zamítání těch, které nesplňují jistá daná kritéria. Výhodou tohoto přístupu je poměrně snadná možnost vzorkování libovolného rozdělení. Ale taky pokud je ale zamítáno velké množství vzorků, může být efektivnost velmi nízká.
Jako příklad uvedu podmínku generovaní náhodných vzorků do kruhu. Mějme Rx a Ry dvě náhodné proměnné, nabývající hodnot <0;1> s rovnoměrným rozdělením pravděpodobnosti.
Pak podmínka **if (Rx*Rx + Ry*Ry) <= 1** zaručí generovaní náhodných vzorků do kruhu. Vzorky, které nesplní podmínku jsou zahozeny.

3.3.2.4 Metropolis vzorkování

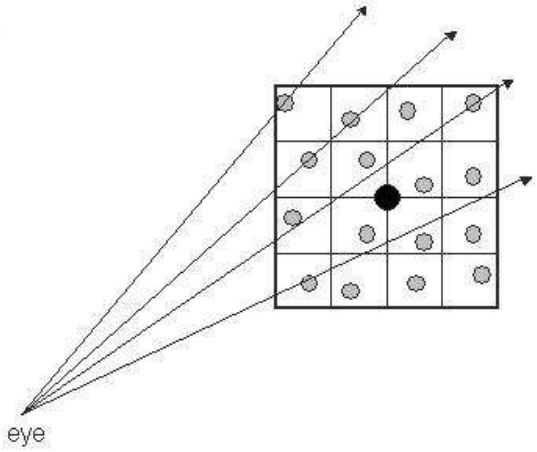
Metropolis vzorkování je pokročilá technika, která umožňuje vzorkovat jakoukoli funkci hustoty pravděpodobnosti v libovolném počtu dimenzí. Vzorky jsou generovány pozměněním předchozího vzorku. Zároveň je proveden test přijatelnosti, nový vzorek je případně zamítnut a použita opět stejná hodnota. Stinnou stránkou je silná korelace mezi jednotlivými vzorky. Podrobně je tato technika popsána v [6].

4 Distributed Ray Tracing

Na několika následujících stránkách představím Distributed Ray Tracing, tak jak ho v červenci v roce 1984 představili Robert L. Cook, Thomas Porter a Loren Carpenter. Přesně podle jejich popisu se taky pokusím tuto realistickou zobrazovací metodu naimplementovat. Až se mi podaří naplnit tento plán, budu do základní implementace přidávat další z mnoha možných rozšíření, o které se tato zajímavá metoda od svého vzniku obohatila. Asi nejvíce na tuto metodu navázal James Kajiya, který ji použil jako základ pro svůj Path Tracing.

4.1 Vyhlazování, antialiasing

Když se znovu podíváme na klasický Ray Tracing, určitě si po chvíli všimneme vysokého aliasingu hran objektů. Distributed Ray Tracing se snaží tento nedostatek řešit vzorkováním jednotlivých pixelů. Přes každý pixel není vyslán jen jeden paprsek, ale je jich distribuováno několik.



Obr. 5 – Princip multi-samplingu

4.2 Osvětlení, stínování

Celkové osvětlení určitého bodu scény je řešeno pomocí zobrazovací rovnice, pro připomenutí

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

(ϕ_i, θ_i) je úhel dopadu (ϕ_r, θ_r) je úhel odrazu

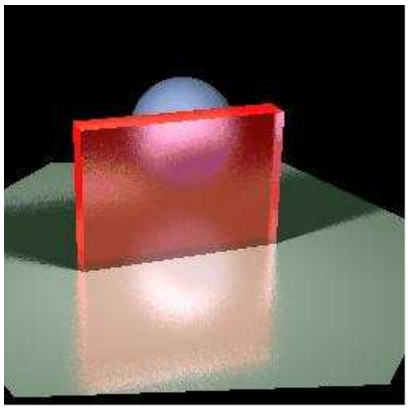
Intenzita I odraženého světla v bodu na povrchu je integrál přes polokouli nad povrchem z funkce osvětlení (illumination) L a funkce odrazivosti R. Odrazivost R vlastně specifikuje optické vlastnosti materiálu a je to funkce BRDF.

Klasický Ray Tracing využívá následující zjednodušující myšlenky:

1. Pokud je L nulová s výjimkou směru do bodových světel, pak integrál přejde na sumu (ostré stíny).
2. Pokud předpokládáme, že dopadající světlo je ze všech směru stejné (L nezávisí na úhlu dopadu, lze ho vytknout před integrál) to znamená, že R můžeme nahradit průměrnou (ambientní) odrazivostí.
3. Pokud předpokládáme, že povrch je zrcadlo a odráží jen v úhlu dokonalého odrazu. Dostáváme ostré odrazy.

Bez těchto zjednodušujících předpokladů je nutné analytické vyjádření integrálu příliš složité. Distributed Ray Tracing integrál vyhodnocuje bodovým vzorkováním distribuováním paprsku. Stínové paprsky nebudeme sledovat jen v jednom směru, ale v závislosti na funkci osvětlení L Odražené paprsky nebudeme sledovat jen ve směru dokonalého odrazu, ale budeme je distribuovat podle funkce odrazivosti R (BRDF).

4.3 Neostré odrazy a průsvitnost

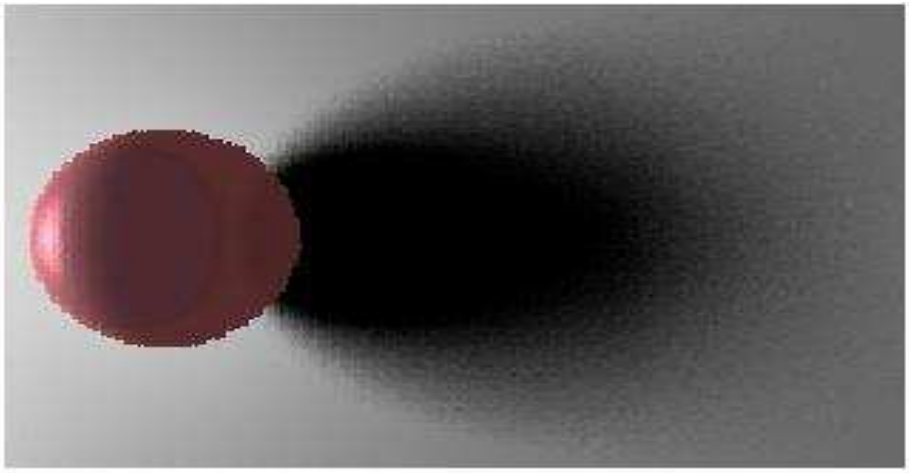


Obr. 6 – Neostrá průhlednost

Klasický Ray Tracing řeší odraz a lom naprosto přesně podle zákona odrazu a Shnellova zákona lomu. Výsledné obrázky ale nepůsobí realisticky. Tento nedostatek R.L. Cook vyřešil distribucí několika paprsků podle distribuční funkce popisující materiál, na kterém se paprsek odráží. U většiny odrazivých (průhledných materiálů) jsou paprsky vrhány v okolí paprsku pro přesný odraz (lom).

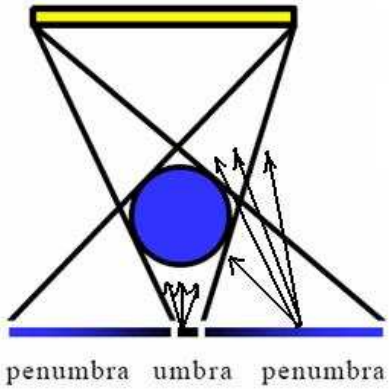
Obrázek 6 jsem převzal z dokumentu [5]

4.4 Polostíny, penumbras



Obr. 7 – Měkké stíny

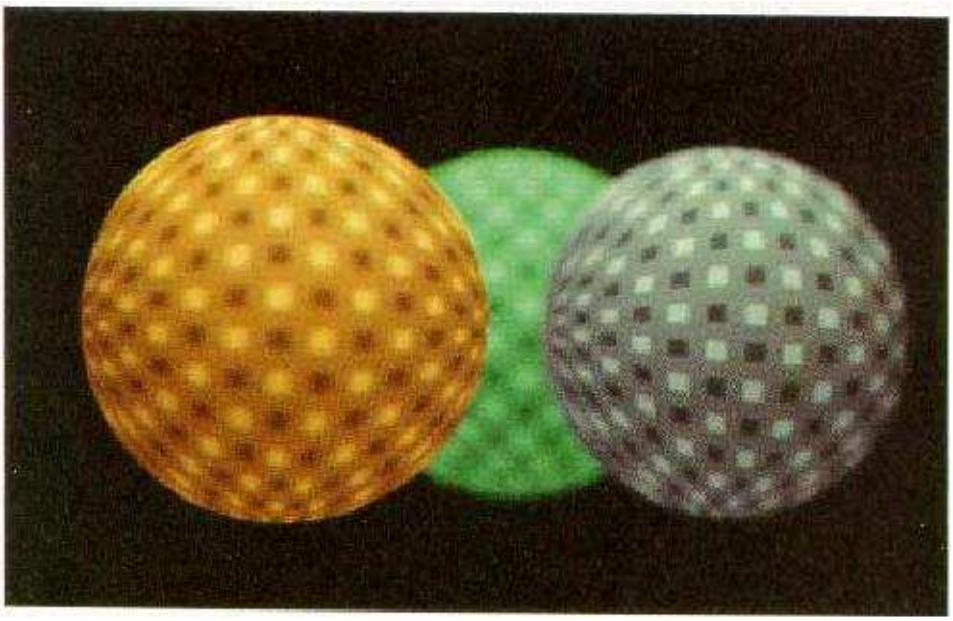
Polostíny vznikají a objevují se tam, kde je světelný zdroj z části zakryt. Příchozí intenzita je úměrná prostorovému úhlu viditelné části světelného zdroje. Polostíny se vypočítají distribucí stínových paprsků k náhodně rozmístěným bodům na světelných zdrojích. Distribuce je vážena jasem jednotlivých částí zdroje. Množství stínových paprsků vyslaných do každého světelného zdroje by mělo být úměrné množství světelné energie dodávané zdrojem, kdyby byl tento zdroj nezakrytý.



Obr. 8 – Vznik měkkých stínů

Obrázek 7 jsem převzal z dokumentu [5]

4.5 Hloubka ostrosti, Depth of Field



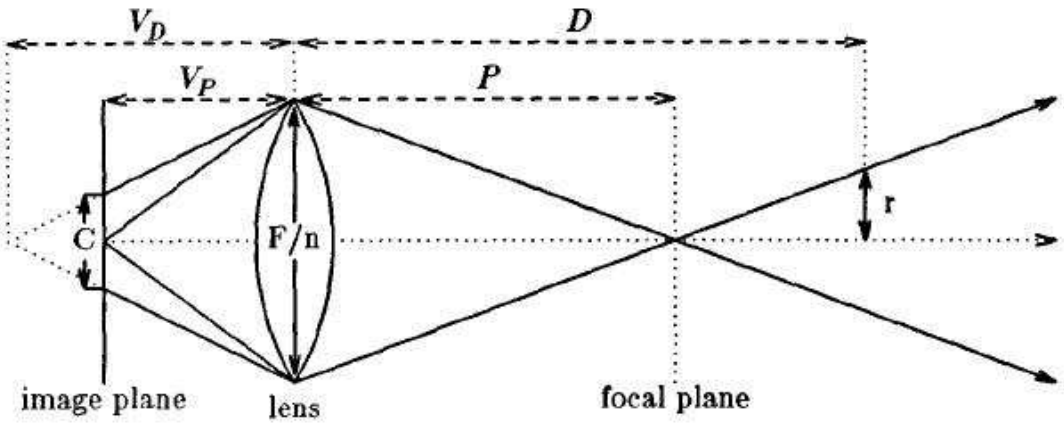
Obr. 9 – Hloubka ostrosti

Hloubka ostrosti vyjadřuje rozdíl vzdálenosti nejbližšího a nejvzdálenějšího předmětu, které se na výsledném obrázku ještě lidskému oku jeví jako ostré. Zavádění hloubky ostrosti do renderovacích algoritmů se může zdát zbytečné. Každý přeci chce mít na výsledném vyrenderovaném obrazu vše dokonale ostré. Srovnejme dva obrazy jeden všude dokonale ostrý, druhý rozmazaný zaostřením nějakého objektu. Po chvilce zjistíme, že dokonale ostrý obrázek neodpovídá realitě, neodpovídá tomu, na co je lidské oko zvyklé. Je to dáno tím, že kamera, oko a podobné optické soustavy mají konečný počet čoček. Díky tomu mají výsledné obrazy konečnou hloubku ostrosti.

Hloubka ostrosti vyjadřuje rozdíl vzdálenosti nejbližšího a nejvzdálenějšího předmětu, které se na výsledném obrázku ještě lidskému oku jeví jako ostré. Proto každý dobrý fotorealistický zobrazovací algoritmus musí s hloubkou ostrosti počítat.

Mějme následující situaci čočku: (lens) , ohniskovou rovinu (focal plane), stínítko a zobrazovací rovinu(image plane).

Autor obrázku 9 je R.L. Cook, převzal jsem jej z dokumentu [2]



Obr. 10 – Matematické pozadí pro hloubku ostrosti

Předpokládejme, že máme scénu umístěnou vpravo od čočky. Pak každý bod ve scéně se přes čočku zobrazí na stínítku jako kružnice. Tato kružnice se nazývá kružnice splývání(Circle of confusion). Na obrázku je značena C. Pokud původní bod scény leží v ohniskové rovině, pak je poloměr kružnice splývání roven velikosti zobrazovaného bodu. V každém jiném případě má kružnice splývání poloměr větší a tím vlastně dochází k neostrosti obrazu.

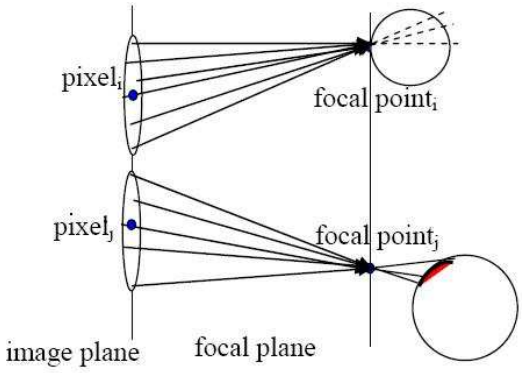
4.5.1 Implementace hloubky ostrosti

Ještě dříve než Robert L. Cook reprezentoval Distributed Ray Tracing se tento problém řešil následovně. Dokonale ostré objekty obrazu byly vyrenderovány metodou, která simuluje dírkovou kameru(např Ray Tracing). Poté byl na obraz aplikovaný filtr rozostření. Tato metoda je výpočetně náročná a nedávala ve všech případech dobré výsledky.

Robert L. Cook navrhl následující implementaci.

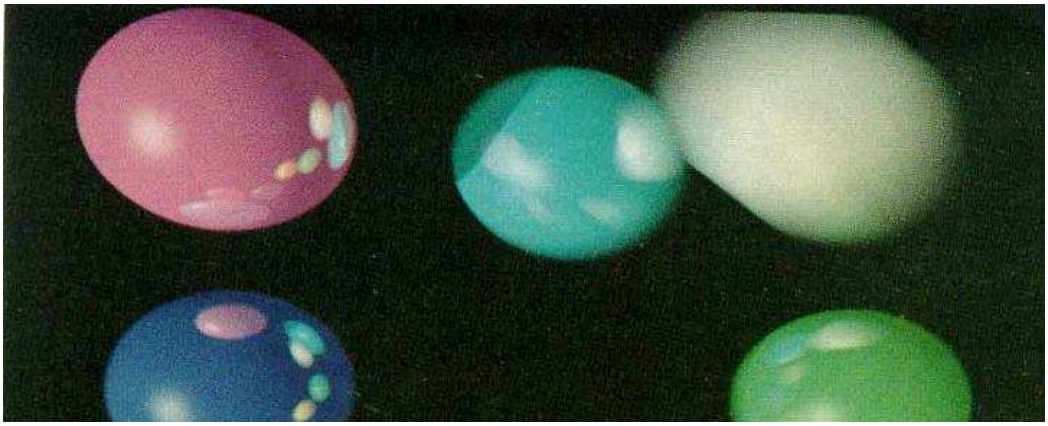
1. Mějme definovanou konečnou optickou soustavu s ohniskovou vzdáleností, a umístěme ji před promítací rovinu.
2. Vytvořme paprsek z oka (středu optické soustavy) do bodu na promítací rovině. Spočítejme průsečík tohoto paprsku s ohniskovou rovinou optické soustavy. Průsečík označme focal point.
3. Na čočce náhodně zvolíme množinu bodů(je dokázané že je vhodné využít poissonova rozložení nah. čísel), sestrojíme paprsky do bodu focal point na ohniskové rovině. Sečteme jednotlivé příspěvky paprsků, a určíme barvu výsledného pixelu.

Autor obrázku 10 je R.L. Cook, ve své práci [2]



Obr. 11– Zjednodušený postup pro implementaci hloubky ostrosti

4.6 Rozmazání objektu pohybem, Motion Blur



Obr. 12– Rozmazání objektů pohybem

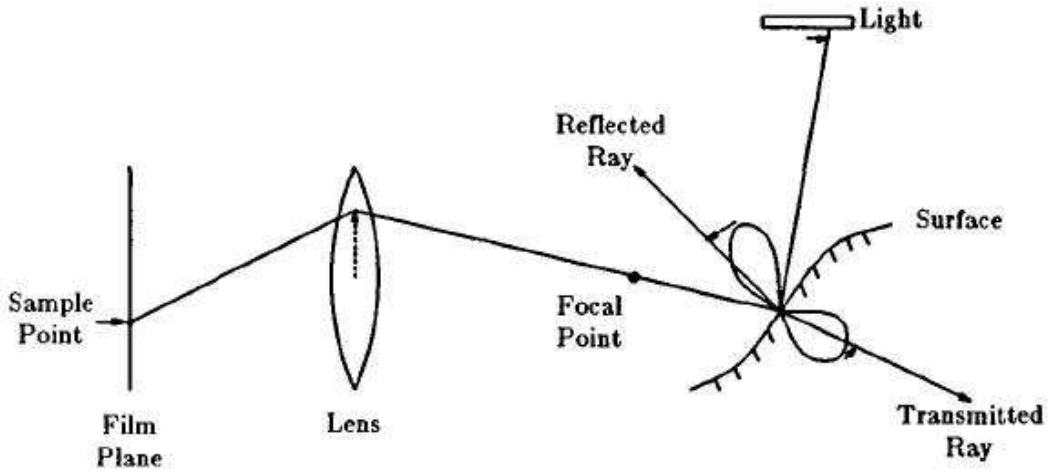
Distributed Ray Tracing jako první správně řeší problém rozmazání pohybujících se objektu ve scéně. Metoda nahradila výpočetně náročné metody, které nejdříve spočetly výslednou scénu v klidu, poté na ni aplikovaly pohybovou rozmazávací funkci. Takto založené metody byly časově náročné a navíc v některých případech nepodávaly uspokojivé výsledky(např. když je pohybující se objekt skryt za statickým).

Robert L. Cook navrhl řešení rozmazání pomocí distribuce náhodných paprsků v čase. Toto řešení vrhá do scény paprsky v různých časových okamžicích. Metoda musí umět zjistit, na jaké pozici se objekt v daný časový okamžik nachází, dále metoda nepotřebuje žádné další extra výpočty. Velkou výhodou je, že toto řešení zachovává správné stínování, osvětlení a hloubku ostrosti.

Autor obrázku 12 je R.L. Cook, ve své práci [2]

4.7 Algoritmus

1. Zvolíme časový okamžik, pro který budeme scénu renderovat a přesuneme objekty na správnou pozici.
2. Vytvoříme paprsek z oka (středu čočky) do bodu v zobrazovací rovině. Náhodně zvolíme pozici na čočce a sestrojíme paprsek přes ohnisko původního paprsku. Spočítáme průsečík paprsku se scénou.
3. Spočítáme stíny. Na každém světelném zdroji vybereme náhodně množinu bodů, pozic. Počet bodů by měl být přímo úměrný výkonu světelného zdroje.Sestrojíme stínové paprsky tak, že spojujeme průsečík určený v kroku 2 s množinou bodů na světelném zdroji.
4. Odraz určíme tak, že vyšleme svazek paprsků náhodně rozmístěných kolem přesného odrazu.Počet vyslaných paprsků je úměrný intenzitě světla přicházejícího ze směru odrazu.
5. Průhlednost spočítáme podobně jako odraz. Vytvoříme svazek paprsků náhodně rozmístěných kolem přesného lomu. Počet vyslaných paprsků je úměrný intenzitě světla přicházejícího ve směru přesného lomu.



Obr. 12– Algoritmus Distributed Ray Tracingu

Autor obrázku 13 je R.L. Cook, ve své práci [2]

4.7.1 Shrnutí algoritmu

Výsledná intenzita každého pixelu ve vyrenderovaném obrázku je řešením složité analytické funkce(zobrazovací rovnice). Distributed Ray Tracing řeší tuto funkci pomocí několika vložených integrálů. Integrací přes časovou doménu, integrací přes plochu pixelu(antialiasing), integral přes čočku a dále integrál příchozího osvětlení řešící odraz/lom. Pro řešení těchto složitých integrálů použijeme metodu Monte Carlo, která využívá náhodného vzorkování.

5 Závěr teoretické části

Myslím, že jsem dostatečně nastudoval teorii kolem zobrazovací metody Distributed Ray tracing. Během následujícího období chci tuto metodu naimplementovat tak, jak ji představil Robert L. Cook. Předpokládám, že nejobtížnější a zároveň nejzajímavější bude odhadnout a generovat náhodné vzorky. Musím najít tu nejvhodnější distribuční funkci pro různé materiály. Mým cílem je získat správnou zobrazovací metodu a co nejvíce se přiblížit obrázkům od R.L. Cooka.

6 Realizace vlastního Distributed Ray Tracingu

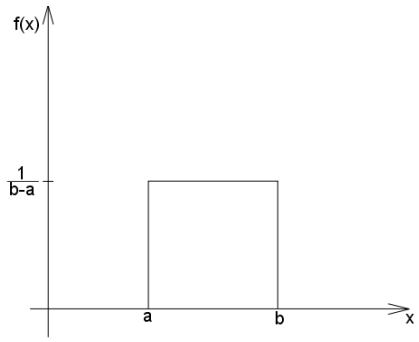
Až doposud jsem nastudoval a popsal teorii potřebnou pro implementaci Distributed Ray Tracingu. Ve zbylé části své diplomové práce uvedu praktické zkušenosti z realizace. Při tvorbě programu jsem se snažil co nejvíce držet nastudované teorie. Ne vždy to však bylo možné. Některé poznatky z implementované metody jsem si zjednodušil, některé jsem naopak rozšířil. Výsledkem jsou kvalitní a přitom jednoduchá zobrazení. Těch uvádím na následujících stránkách dostatečné množství. Lze je porovnat se zobrazeními od R.L. Cooka a zhodnotit výsledky mé práce.

6.1 Generování vzorků

Nějdříve vysvětlím, jak jsem řešil problematiku generování správných náhodných vzorků. Záměrně začínám touto kapitolou, je důležité ji projít jako první. Na generování vzorků se budou odkazovat téměř všechny další kapitoly.

Pro generování vzorků všech použitých druhů rozdělení jsem jako základ použil generátor náhodných čísel v jazyce C. Tento generátor je obsažen v knihovně `stdlib` a generuje náhodná celá čísla od 0 do RAND_MAX. Tato náhodná čísla si pak upravuji tak, abych dostal potřebné rovnoměrné, normální a poissonovo rozdělení pravděpodobnosti.

6.1.1 Rovnoměrné rozdělení



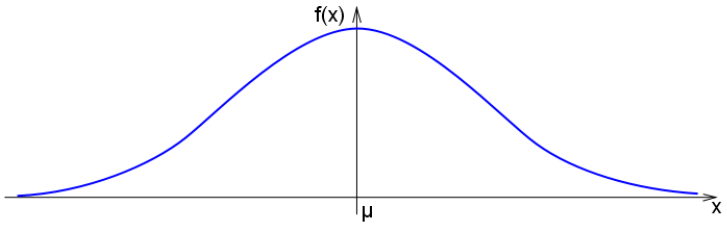
Obr. 13– Funkce pravděpodobnosti rovnoměrného rozdělení

Toto rozdělení přiřazuje všem hodnotám náhodné veličiny stejnou pravděpodobnost. Implementoval jsem spojité rozdělení generované pro interval <a,b>. Toho jsem snadně dosáhl matematickou úpravou generátoru náhodných čísel z jazyka C. Matematickou úpravu uvedu v pseudokódu.

```
Double Function generuj (int a, int b){
        Return ((rand()/RAND_MAX) * (b-a) – (b-a)/2);
}
```

6.1.2 Normální rozdělení

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



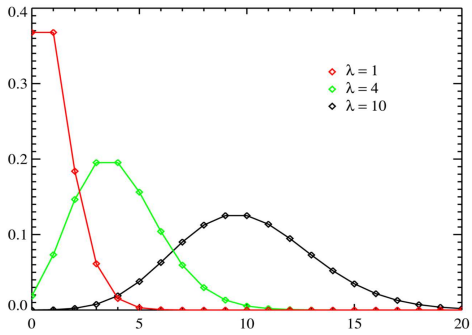
Obr. 14– Funkce pravděpodobnosti normálního rozdělení

Normální (nebo Gaussovo) rozdělení pravděpodobnosti je jedno z nejdůležitějších rozdělení pravděpodobnosti spojité náhodné veličiny.

Toto rozdělení dostanu z rovnoměrného rozdělení pomocí metody Box-Muller transform. Jedná se o metodu využívající inverzní transformace. Inverzní transformace byla popsána v teoretické části. Podrobněji je celá Box-Muller metoda je popsaná v dokumentu [12]

6.1.3 Poissonovo rozdělení

$$f(k,\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$



Obr. 15– Funkce pravděpodobnosti poissonova rozdělení

Poissonovo rozdělení bývá označováno jako rozdělení řídkých jevů, neboť se podle něj řídí četnosti jevů, které mají velmi malou pravděpodobnost výskytu.

Toto rozdělení generuji podle algoritmu od pana Donalda Knutha. Při realizaci jsem zpracoval jeho pseudokód. D.Knut při tvorbě tohoto algoritmu opět využil inverzní transformace.

```
function poisson random number (){
    Let L ← e−λ, k ← 0 and p ← 1.
    do:
         k ← k + 1.
         Generate uniform random number u in [0,1] and let p ← p × u.
    while p ≥ L.
    return k − 1.
}
```

6.1.4 Jednotlivá rozdělení v prostoru.

Generování vzorků jednotlivých rozdělení pravděpodobností bylo nutné rozšířit do prostoru. Teprve tyto 3D generátory jsem mohl použít pro vzorkování směru odražených paprsků.

Při převodu obyčejných generátorů na 3D generátory jsem využil **sférických** souřadnic. Bod v této soustavě souřadnic má složky [r,fi,theta], kde r je vzdálenost bodu od počátku, fi je odklon (úhel) průvodiče od osy x a theta je odklon průvodiče od osy z. Více o této soustavě najdete v dokumentu [14].

Hlavní myšlenka pro dosažení 3D rozdělení je generovat úhel fi pomocí námi požadovaného výsledného rozdělení, úhel theta se generuje pomocí rovnoměrného rozdělení. Poté následuje převod do kartézkých souřadnic. Nakonec je potřeba vygenerovaný bod ještě otočit kolem vektoru udávající směr natočení výsledného generátoru.

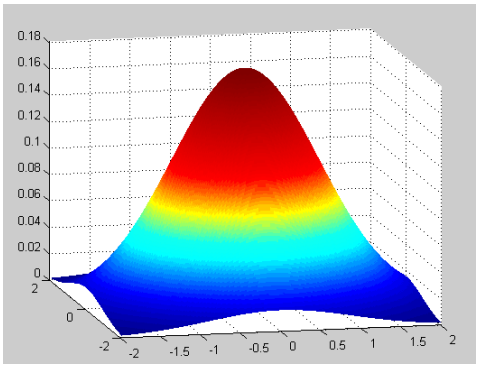
Pseudokód:

```
phi,theta,u,v,w;
phi =  generuj bod požadovaného rozdělení
theta = generuj odklon od osy x rovnoměrným rozdělením;

//nasleduje převod do kartézkých souřadnic
u = sin(phi) * cos(theta);
v = sin(phi) * sin(theta);
w = cos(phi);

rotuj výsledný bod podle vektoru udávající směr natočení výsledného
3D generátoru.
return bod;
```

6.1.4.1 Rovnoměrné rozdělení

V tomto případě se generuji vzorky o souřadnicích [x,y,z], které tvoří povrch koule. Tohoto generátoru se využívá pro určení směru při difusním odrazu.

6.1.4.2 Normální rozdělení



Obr. 16– Funkce pravděpodobnosti normálního rozdělení v prostoru

Tento generátor generuje vzorky o souřadnicích [x,y,z], které tvoří Gaussovu plochu. Tohoto generátoru využívám pro nepřesný odraz a lom. Výše uvedený obrázek jsem udělal v matematickém programu Matlab. Zdrojový kód najdete v přílohách.

6.2 Reprezentace scény

V této kapitole uvedu své pojetí světel, objektů a materiálů v počítačové scéně. Načítání scény se provádí parsováním XML souboru. Formát a sémantiku tohoto vstupního XML souboru rozeberu později.

6.2.1 Světla

Ve své práci jsem implementoval světla plošná, kruhová. Tyto zdroje vyzařují energii rovnoměrně celým svým povrchem. Do scény je možno přidat libovolný počet těchto světel. Každé světlo ovšem zpomaluje výpočet a vykreslení celé scény. Při implementaci světel jsem použil zamítací Monte Carlo.

Ukázka specifikace světla ve vstupním XML souboru pro popis scény:

```
<Light Px="300" Py="250" Pz="500" Radius="40" Ir="1.0" Ig="1.0" Ib="1.0" />
```

Světlo má střed v bodě [300,250,500], poloměr je 40 a I označuje světelnou energii pro tři základní barvy. Z toho jde vidět, že mohu tvořit barevná světla, ale bohužel jen třísložková.

6.2.2 Objekty

Můj Distributed Ray Tracing podporuje dva základní objekty. Je to polorovina a koule. U těchto objektů bylo třeba naprogramovat metody pro průsečík objektu s paprskem, metodu pro určení zda se nacházíme uvnitř objektu, metodu na procedurální texturování. Dále jsem přidal i bump-mapping, což je matematická změna normál na povrchu objektu. Tato změna působí dojmem 3D nerovností na povrchu.

Ukázka specifikace poloroviny:

```
<Plane Px="0" Py="0" Pz="0" Vu="0" Vv="1" Vw="0" Material="BlueLambert" bumpEnable="0" EnableTexture= ”1” />
```

Jde o rovinu, která obsahuje bod [0,0,0] a normálový vektor roviny je (0,1,0). U roviny je neaktivovaný bump mapping, ale je aktivované procedurální texturování. Materiál roviny je `BlueLambert`

Ukázka specifikace koule:

```
<Sphere Px="150" Py="90" Pz="500" Radius="80" Material="Mirror"/>
```

Takže jde o zrcadlovou kouli se středem v [150,90,500] s poloměrem 80. Koule nemá zapnuto ani texturování ani bump-mapping.

6.2.3 Materiály

Správný popis/reprezentace materiálů a k nim příslušných BRDF funkcí je dle mého úsudku nejobtížnější úkol. Můžeme mít sebelepší zobrazovací algoritmus, pokud ale zvolíme špatný popis materiálu, stejně se nám výsledné obrázky nebudou líbit. To, jaký materiál s jakými vlastnostmi zvolíme, má rozhodující vliv na konečnou věrohodnost obrázků.

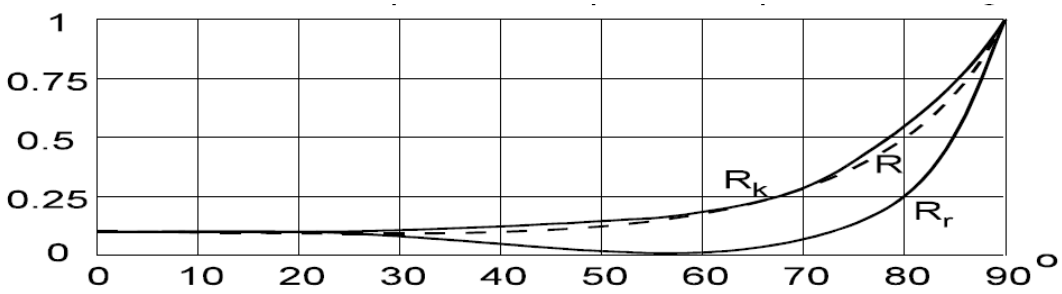
Já se snažím materiál popsat podle jeho rozboru v úvodní teoretické části. To znamená, že jej rozlišuji na: **difusní**, **odrazivý** a **průhledný**. Samozřejmě se snažím popsat i materiál obecný, který je kombinací difusního odrazivého a průhledného.

Difusní materiál – materiál podobný omítce.To jak moc je materiál difusní, popisuji konstantou **difuse**, která je v intervalu <0,1> a udává míru matného osvětlení. Barva difusní složky materiálu je vyjádřena pomocí třech základních barev, kterým odpovídají konstanty kdr, kdg, kdb. Všechny tyto konstanty jsou v rozsahu <0,1>.

Odrazivý materiál – materiál podobný lesklému kovu, zrcadlu ale i vodní hladině .Míru odrazivosti materiálu popisuji konstantou **reflective** a ta je v intervalu <0,1>. Přesnost odrazu definuji pomocí konstanty n (0,nekonečno). Čím větší je n, tím je odraz přesnější. Jako materiál s indexem n rovným 1000 lze simulovat zrcadlo. Materiál s n rovno 10 můžeme modelovat například povrch kovu opracovaného pískováním. Jen pro upřesnění, pískování je metoda opracování materiálu, kdy jsou tisíce pískových zrn vrhány pod tlakem vzduchu na opracovávanou součást. Dále jsem definoval tři konstanty ksr, ksg, ksb. Všechny tyto konstanty jsou opět v rozsahu <0,1> a udávají barvu pro odrazené světlo.

Průhledný materiál – materiál podobný sklu, vodě. Míru průhlednosti materiálu popisuji konstantou **transmit**, která je v intervalu <0,1>. Opět tu definuji jak přesný je lom, a to pomocí konstanty n (0,nekonečno). Čím větší je n, tím je lom přesnější. Např. materiál s n rovným 1000 lze simulovat čiré sklo. Materiál s n rovno 10 můžeme modelovat sklo s nějakou drsnou povrchovou úpravou. U průhledných materiálů navíc přidávám index lomu. Ten udává optickou hustotu materiálu. Opět i u tohoto materiálu jsou tři konstanty ktr, ktg, ktb v rozsahu <0,1> a udávají barvu pro lomené světlo.

Pro průhledné materiály platí i další vlastnost. Průhlednost záleží na úhlu dopadajícího paprsku. Tato vlastnost je nám důvěrně známa sledováním vodní hladiny. Když se do vody díváme s hora, vidíme co se nachází pod hladinou. Naopak když se budeme dívat na vodu pod menším úhlem, neuvidíme nic jiného, než odraz okolí na vodní hladině. Za tento jev může polarizace světla.



Obr. 17– Empiricky změřená závislost energie zalomeného paprsku na dopadajícím úhlu ve skle

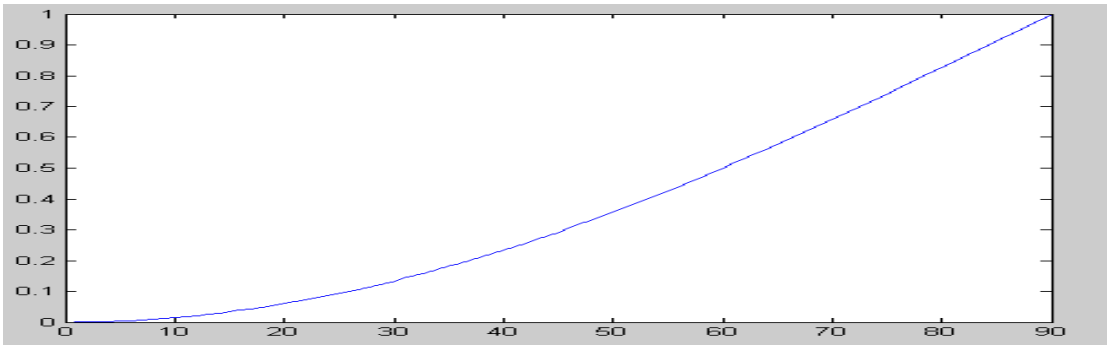
Na předchozím obrázku je naznačeno, kolik energie vstupního paprsku se zalomí do skla při vstupním úhlu od 0 až do 90 stupňů.

Tuto vlastnost jsem se pokusil taky ve své práci nezanedbat. Aproximoval jsem ji pomocí následující rovnice, kde alfa je dopadový úhel.

$$f(x) = 1 - \cos(\alpha)$$

Po rozboru této rovnice v programu Matlabu jsem dostal následující graf.(soubor transmit.m). Myslím, že následující rovnice mi dává dostatečnou aproximaci. Toto řešení dále ještě vylepšuji umocněním funkce, čímž se aproximující řešení ještě více přiblíží skutečnosti.

.



Obr. 18– Matematická aproximace závislosti energie zalomeného paprsku na dopadajícím úhlu ve skle

Další zajímavou vlastností pro průhledné materiály je **totální reflexe**. Ta nastane jen když světelný paprsek přechází z opticky hustšího do opticky řidšího prostředí. Například při přechodu rozhraní ze skla do vzduchu. Navíc paprsek opouštějící hustější prostředí musí dopadat pod větším úhlem než je úhel mezní. Pak nastane situace, že paprsek prostředí neopustí, nezalomí se, ale je odražen podle zákona odrazu. Tuto vlastnost jsem ve své práci implementoval a testoval. Podrobnější informace o této problematice najdete v dokumentu [15].

Nakonec uvedu několik příkladů definice materiálu v XML souboru pro popis scény.

1.<GrayLambert kdr="0.8" kdg="0.8" kdb="0.8" diffuse="1.0" reflective="0.0" transmitt="0.0" /> - Jedná se o čistě difusní povrch světle šedé barvy. Povrch nebude žádnou energii odrážet ani lámat.

2.<AquaSpecular kdr="0.1" kdg="0.8" kdb="0.8" diffuse="0.6" reflective="0.4" transmitt="0.0" /> - Takto jsem popsal napůl difusní a napůl odrazivý materiál modro-zelené barvy. Povrch bude v 50 procentech odrážet okolí.

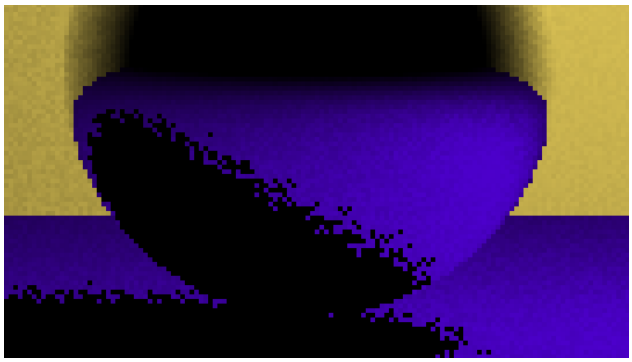
3. <Glass ktr="0.1" ktg="0.8" ktb="0.8" diffuse="0.0" reflective="0.0" transmitt="1.0" index="1.6" n="1000"/> - Zde jsem popsal průhledný materiál modro-zelené barvy. To znamená, že světlo průchozí tímto povrchem se zbarví do modro-zelena. Materiál má optickou hustotu rovnu index lomu tedy 1.6.Materiál je navíc velmi vyleštěný. Bude tedy docházet téměř k dokonalé průhlednosti.

6.3 Distributed Ray Tracing

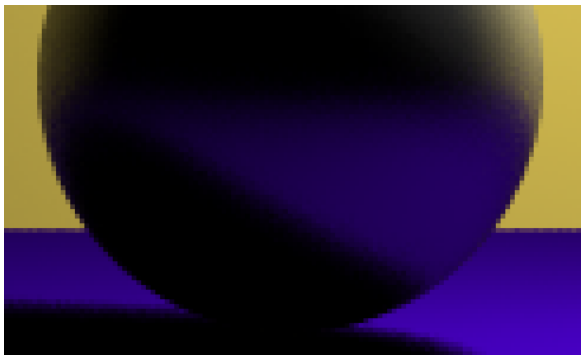
V této kapitole projdu to, jak jsem řešil vlastní jádro zobrazovacího algoritmu. Pozastavím se nad každou z částí Distributed Ray Tracingu. Budu ji prezentovat vlastními obrázky. Realizoval jsem všechny dílčí podalgoritmy Distributed Ray Tracingu. Většina se mi podařila vyřešit velmi zdařile.

6.3.1 Vyhlazování, antialiasing

Vyhlazování jsem implementoval multi-samplingem, neboli nadvzorkováním. Jedná se o metodu, kdy pro jeden pixel je provedeno více výpočtů osvětlení v různých místech pixelu. Pixel vzorkuji plošně rovnoměrným rozdělením. Vzniknou subpixely. Následně každým subpixelem vyšlu světelný paprsek a spočítám osvětlení. Počet paprsků vyslaných jedním pixelem je dáno konstantou. Ta je definována v XML souboru pro popis scény. Přínos každého světelného paprsku se dá vyjádřit 1/celkový počet paprsků.



Obr. 19– Antialiasing 1 vzorek na pixel



Obr. 20– Antialiasing 10 vzorků na pixel

6.3.2 Osvětlení, stínování

Pro výpočet osvětlení difusních povrchů využívám jednoduchého Lambertova osvětlovacího modelu.

$$I = c_d I_d (NL)$$

Význam jednotlivých členů v tomto vztahu je následující:

I představuje celkovou intenzitu světla tak, jak je vnímána uživatelem.

Id představuje intenzitu difúzní složky světla, tj. té části světla, která dopadá na těleso z jednoho světelného zdroje a odráží se do všech směrů rovnoměrně.

cd je koeficient materiálu pro difúzní složku světla.

N je použita normála k povrchu tělesa.

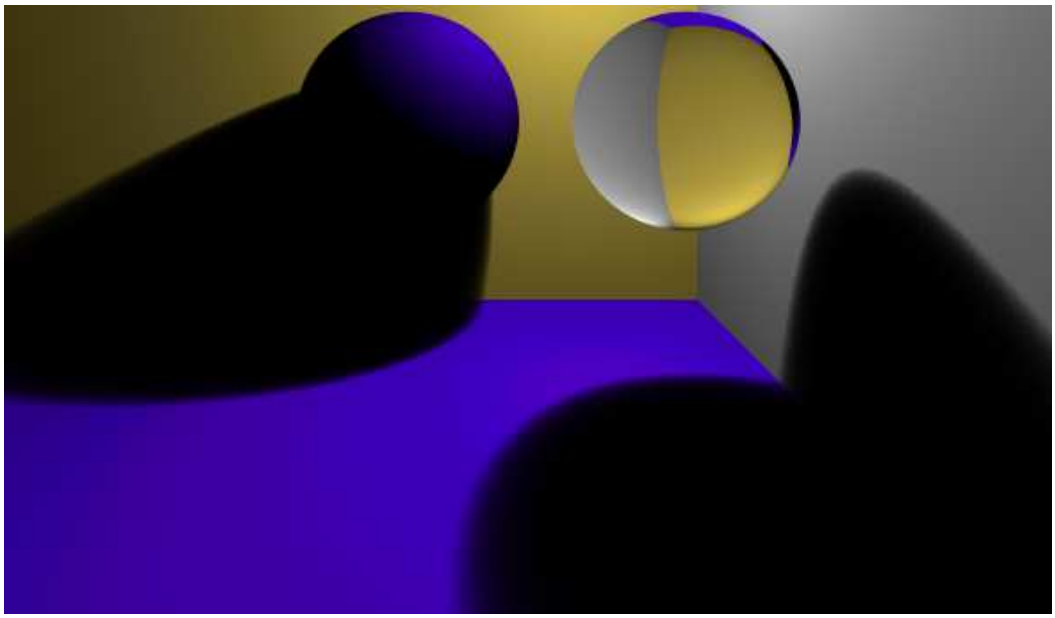
L a vektor ze světelného zdroje do bodu na povrchu tělesa, jehož barvu počítám.

Odraz a lom řeším distribucí více sekundárních paprsků.

6.3.2.1 Měkké stíny

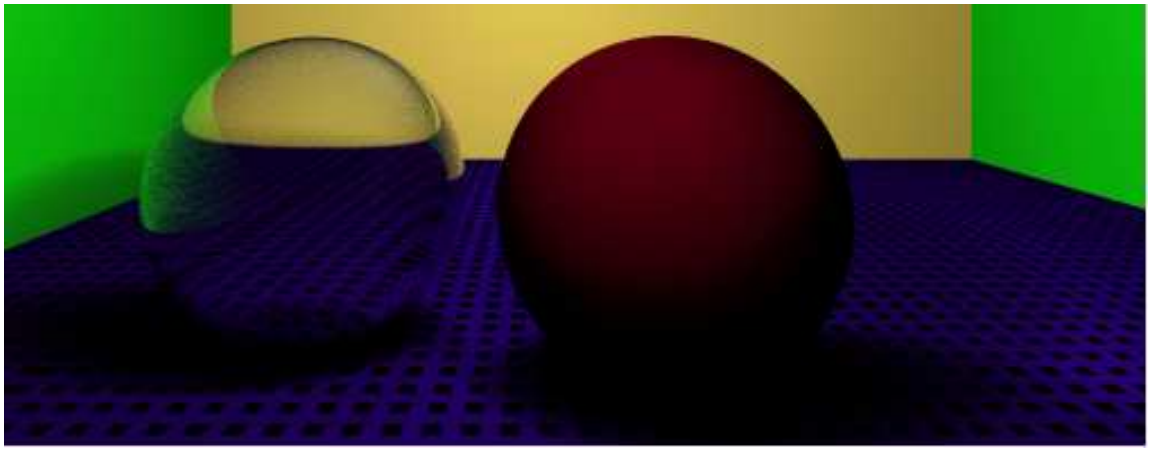
Měkké stíny mohou vzniknout dvojím způsobem:

1. **Plošné zdroje světel**. Plošná světla generuji rovnoměrným rozdělením pravděpodobnosti pomocí zamítacího Monte Carla. Pro každý stínový paprsek se zvolí náhodná pozice na zdroji světla a spočítá se osvětlení.



Obr. 21– Měkký stín jeden zdroj světla

2. **Více zdrojů světel**. Do scény je možno přidat libovolný počet světel. Výsledné osvětlení určím vygenerováním více stínových paprsků do náhodných zdrojů světel. Tyto stínové paprsky jsou následně zprůměrovány a tím dostanu měkké stíny.



Obr. 22– Měkký stín více (dva) zdrojů světla

6.3.3 Hloubka ostrosti

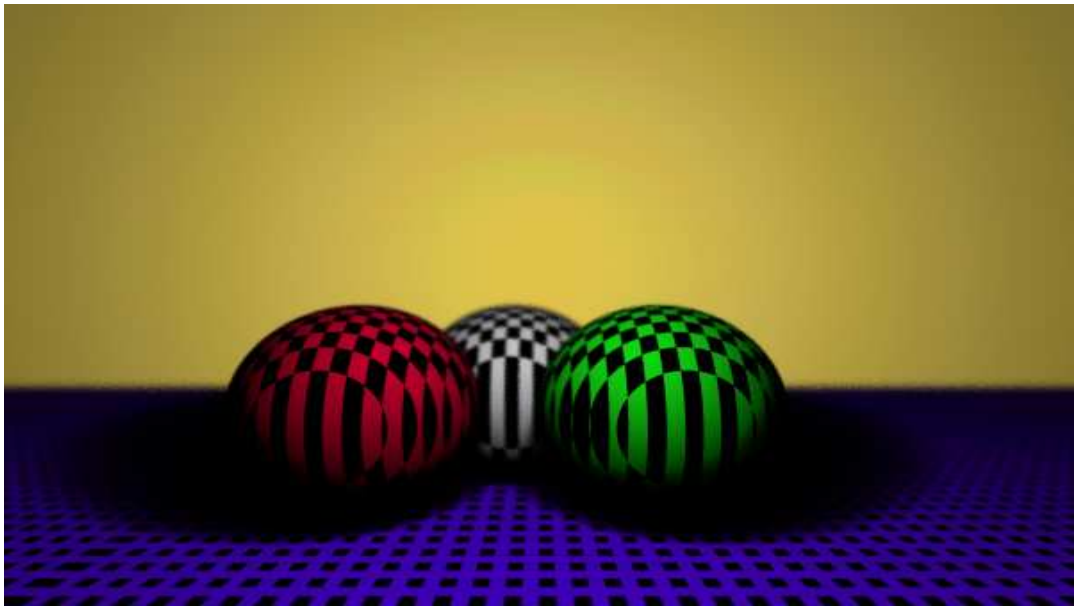
Zaostřování neboli hloubku ostrosti se mi podařilo úspěšně naimplementovat. Zde je postup jak k zaostřování přistupuji.

Nejdříve se načtou údaje o čočce ze vstupního XML souboru. Pomocí těchto údajů se sestaví ohnisková rovina pro danou optickou soustavu. Tato rovina se vytvoří jen jednou. Poté se generují paprsky z oka do scény, kde se vypočítá jejich průsečík s ohniskovou rovinou. Tímto průsečíkem a náhodně vygenerovanou pozicí na čočce se vytvoří nový světelný paprsek. Tento paprsek už řeší hloubku ostrosti. Generování náhodných vzorků na čočce se provádí pomocí poissonova rozdělení pravděpodobnosti.

Ukázka definice parametrů optické soustavy v XML souboru.

```
<DepthOfField Radius = "15" Focal = "200"/>
```

Jedná se o čočku, která má velikost R 15 a ohniskovou vzdálenost rovnu 200.
Následuje ukázka. Ohnisko pro tento obrázek je zaostřeno na obě přední koule, pozadí je rozostřené.



Obr. 23– Hloubka ostrosti

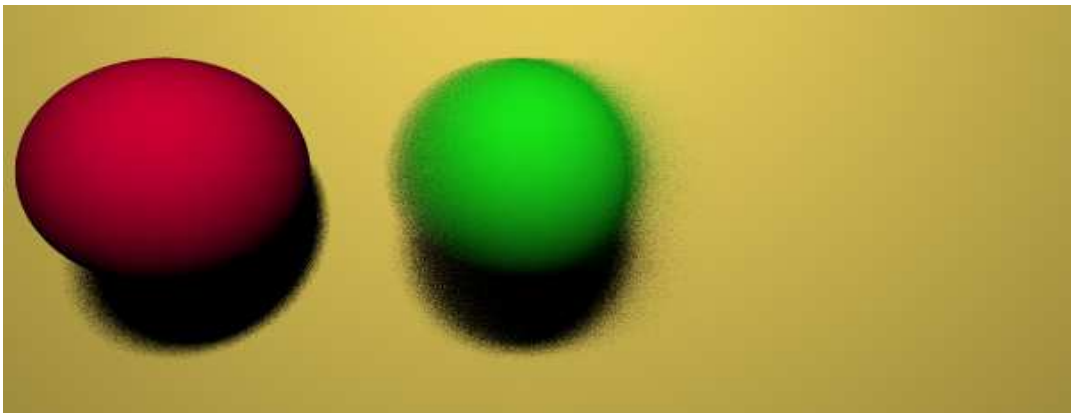
6.3.4 Rozmazání pohybem

Neboli Motion Blur. Pro nasimulování tohoto jevu je potřeba ve scéně zavést čas, časovou informaci. Proto jsem každému objektu ve scéně umožnil definovat jeho pozici v minulém čase. Tyto pozice se opět definují v XML souboru pro popis scény a uvedu zde krátký příklad.

```
<Sphere Px = "275" Py = "300" Pz = "150" Radius ="100" Material="Mirror2" >
  <MotionBlurPosition Px="276" Py = "300" Pz = "150" />
 <MotionBlurPosition Px="278" Py = "300" Pz = "150" />
```

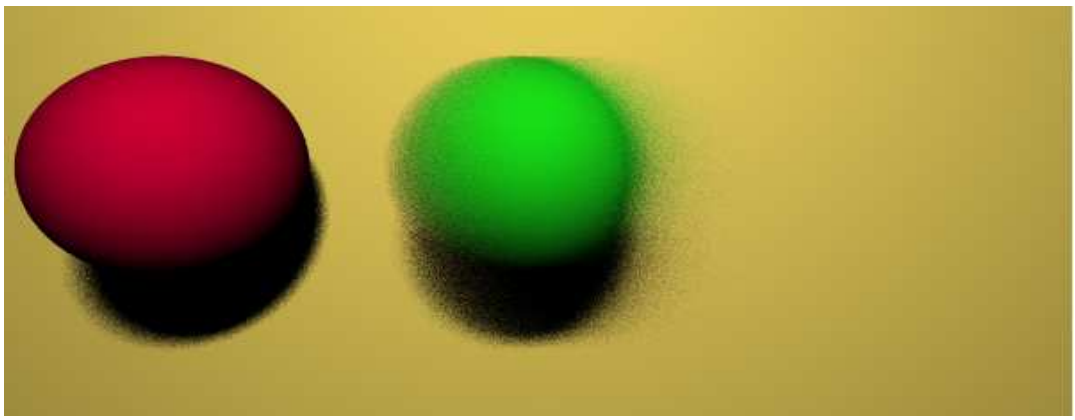
```
  <MotionBlurPosition Px="279" Py = "300" Pz = "150" />
   <MotionBlurPosition Px="280" Py = "300" Pz = "150" />
  <MotionBlurPosition Px="281" Py = "300" Pz = "150" />
  <MotionBlurPosition Px="283" Py = "300" Pz = "150" />
  <MotionBlurPosition Px="284" Py = "300" Pz = "150" />
  <MotionBlurPosition Px="285" Py = "300" Pz = "150" />
</Sphere>
```

Takto definuji kouli, pro kterou udržuji historii devíti pozic v minulosti. Orientace času podle zápisu od shora dolů odpovídá času od současnosti do minulosti. V popisovaném příkladě je kulička na pozici [275,300,150]. Před devíti časovými jednotkami se kulička nacházela na pozici [285,300,150]. Vlastní princip generování Motion Blur je následující. Pro každý světelný paprsek náhodně zvolím časový okamžik, pro který se bude scéna vykreslovat. Zde jsem narazil na problém. Bylo nutno rozhodnout, s jakým rozdělením pravděpodobnosti mám generovat minulost. Jako první variantu jsem zvolil normální rozdělení času se střední hodnotou v současnosti. Na následujícím obrázku vidíte výsledek.



Obr. 24– Motion Blur, generovaný pomocí normálního rozdělení pravděpodobnosti v čase

Jako druhou variantou jsem použil rovnoměrné rozdělení v čase. Pro srovnání uvedu stejnou scénu s rovnoměrným rozdělením.

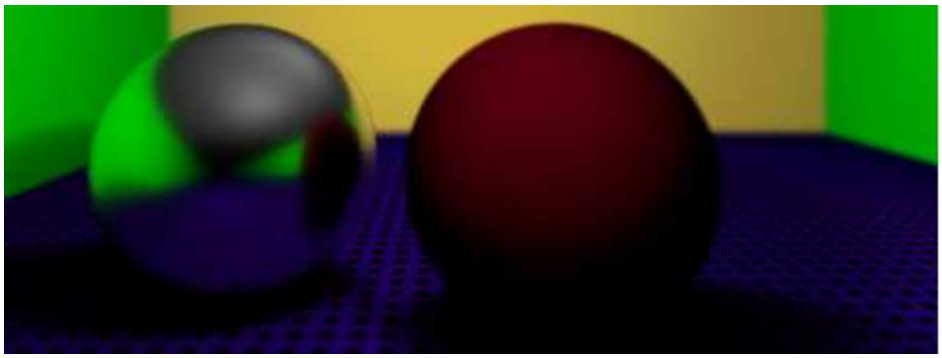


Obr. 24– Motion Blur, generovaný pomocí rovnoměrného rozdělení pravděpodobnosti v čase

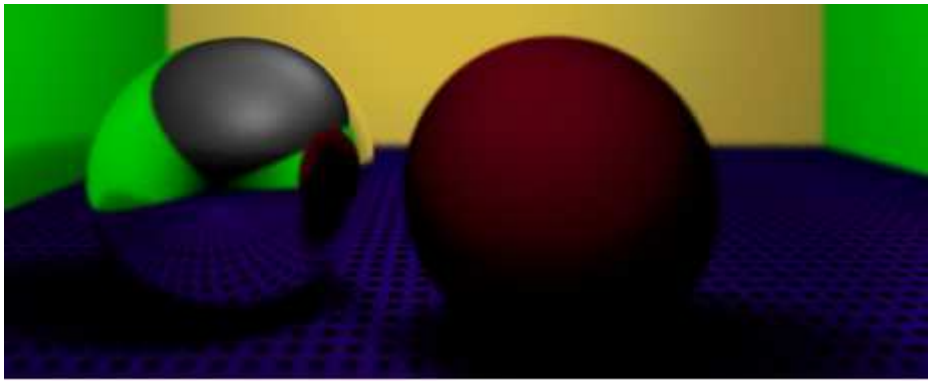
Jako správnější považuji variantu rovnoměrného rozdělení v čase, která více odpovídá skutečnému řešení rozmazání objektu jeho pohybem. Proto jsem se nakonec přiklonil k tomuto řešení.

6.3.5 Odrazy

Distributed Ray Tracing umí řešit neostré odrazy. Jak už jsem uvedl dříve ostrost odrazu popisuji konstatou n. Vlastní generování odrazeného paprsku realizuji pomocí 3D generátoru normálního rozdělení. Toto normální rozdělení má střední hodnotu ve vektoru pro přesný odraz. A rozptyl sigma je roven vztahu (1/n). Z toho je jasně vidět, že čím je konstanta n větší, tím přesnější je odraz.



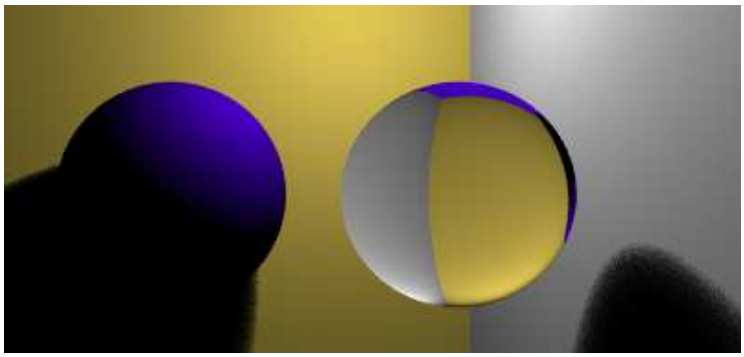
Obr. 25– Nepřesný odraz, n=10,sigma=0.1



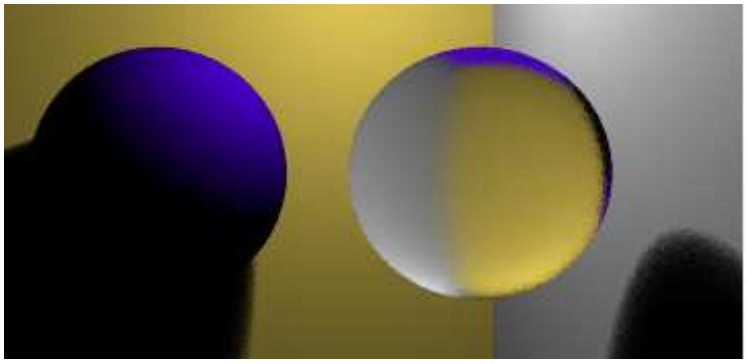
Obr. 26– Přesný odraz, n=500,sigma=0.002

6.3.6 Průhlednost

Podobně jako odrazivé materiály se řeší materiály průhledné. Znovu implementuji ostrost, čistotu průhledného materiálu. Tato čistota se definuje indexem n. Určení lomeného paprsku realizuji pomocí 3D generátoru normálního rozdělení. Toto normální rozdělení má střední hodnotu ve vektoru pro přesný lom s indexem lomu materiálu. Rozptyl sigma je roven vztahu (1/n).

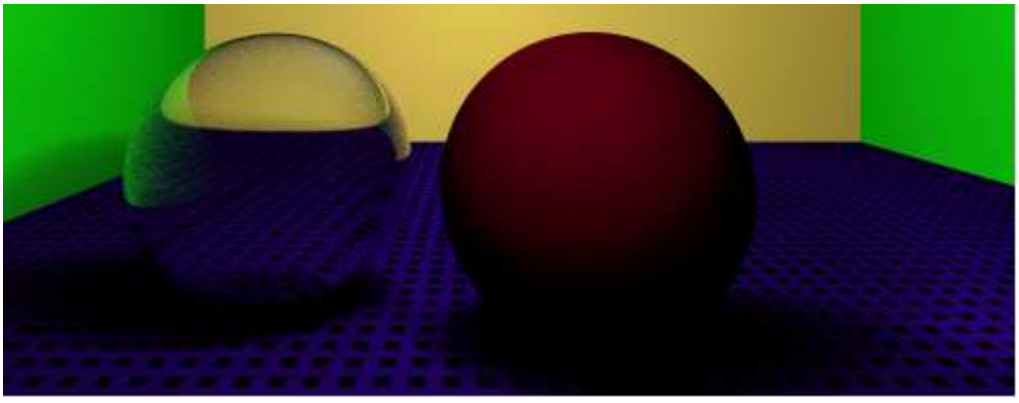


Obr. 27– Dokonale přesný lom, n=500,sigma=0.002



Obr. 28– Nepřesný lom, n=10,sigma=0.1

Poslední ukázka je pro n rovno 500, jedná se o ostrý lom. Navíc je zde implementace lomu v závislosti na úhlu dopadu. Některé paprsky se nezalomí, ale odrazí se, podle vztahu tak, jak jsem ho popsal dříve. Tento efekt jde pozorovat hlavně na okrajích koule.



Obr. 29– Přesný lom závislý na úhlu dopadu světelného paprsku, n=500,sigma=0.002

6.3.7 Shrnutí algoritmu

To, jak jsem realizoval hlavní část Distributed Ray Tracingu nyní vyjádřím v pseudokódu.

```
DRT () {
Inicializace:
Načti scénu.
Sestroj ohniskovou rovinu pro depth of field.
Pro každý pixel výsledného obrázku opakuj.
      For (1:Počet_paprsků_na_pixel){
            Vygeneruj náhodný čas, a do tohoto času přesuň objekty ve scéně
            Generuj náhodně pozici pro pixel, sestroj paprsek oko, pixel.
            Sestroj průsečík P, jako průsečík paprsku s ohniskovou rovinou.
            Generuj náhodný bod D na čočce. Využij poissonova rozdělení.
            Generuj paprsek DP a pošli jej do scény
            Vypočítej průsečík K paprsku DP s objektem scény.
            Generuj n náhodných stínových paprsků a vypočítej osvětlení
      v bodě K
            if(odrazivý_materiál){
                  Generuj odražený paprsek DP a rekurzivně určí osvětlení
      pro tento paprsek
            }
            if(průhledný_materiál){
                  Generuj lomený paprsek DP a rekurzivně určí osvětlení pro
      tento paprsek
            }
      }
}
```

6.4 Implementace

Distributed Ray Tracing jsem naprogramoval v jazyku C/C++. Program je multiplatformní a přenositelný (testoval jsem ho na systémech Windows a Linux). Vstupní data čte z XML souboru, výstupní data jsou uložena do souboru formátu BMP. Všechny zdrojové kódy jsou dokumentované pomocí automatického nástroje Doxygen. Podrobnou dokumentaci vygenerovanou Doxygenem můžete najít na přiloženém CD mediu ve složce /doc/.

V této kapitole jen velmi zhruba nastíním základní rozvržení tříd, pomocí nichž jsem Distributed Ray Tracing implementoval.

Vektory , problematiku vektorů jsem umístil do jedné třídy **DRT_Vector.** Tato třída implementuje základní matematické operace s vektory(skalární a vektorový součin vektorů,sčítání a odečítání vektorů, normalizace, transformace vektorů, a další). Mimo tyto matematické operace se v této třídě nachází i metody fyzikální a to pro výpočet přesného odrazu a lomu vektoru.

Scéna, jak už jsem uvedl scéna se načítá z XML souboru. A právě o načtení scény se stará třída **DRT_Scene.** Tato třída využívá TinyXML, což je malá knihovna na parsování XML. Dále má třída **DRT_Scene** atributy pro objekty ve scéně. Jedná se o **kolekci světel a kolekci objektů**.

Světla, jsou implementovaná v **DRT_Lights**. Mimo základní atributy je zde metoda pro realizaci plošných světel.

Objekty, realizuji v programu v třídě **DRT_Object**. Jde o abstraktní třídu, od které dědí objekty scény(roviny, koule). Tato virtuální třída poskytuje základní metody pro objekty.

`virtual bool DRT_Inter(DRT_Ray ray,double *point);` spočte průsečík

`virtual DRT_Vector DRT_NormalVec(DRT_Point point);` vrátí normálu k povrchu

`virtual int DRT_Texture(DRT_Point point);` procedurální texturování

`virtual bool IsInside (DRT_Point point);` test zda se nacházíme uvnitř objektu

`virtual DRT_Point GetPositionInTime(int time);`přesune se do daného času,Motion Blur

Dále **DRT_Plane DRT_Sphere** dědí od **DRT_Object.** Implementují zděděné virtualní metody.

DRT_Ray v této třídě můžete najít reprezentaci světelných paprsků.

DRT_Material, jde o třídu popisující vlastnosti materiálů. Navíc jsou zde metody pro výpočet nepřesných odrazů a lomů.

DRT_Random_Generator, jedná se o soubor s funkcemi pro generování nahodných čísel. Soubor poskytuje generátory rovnoměrného, normálního a poissonova rozdělení.

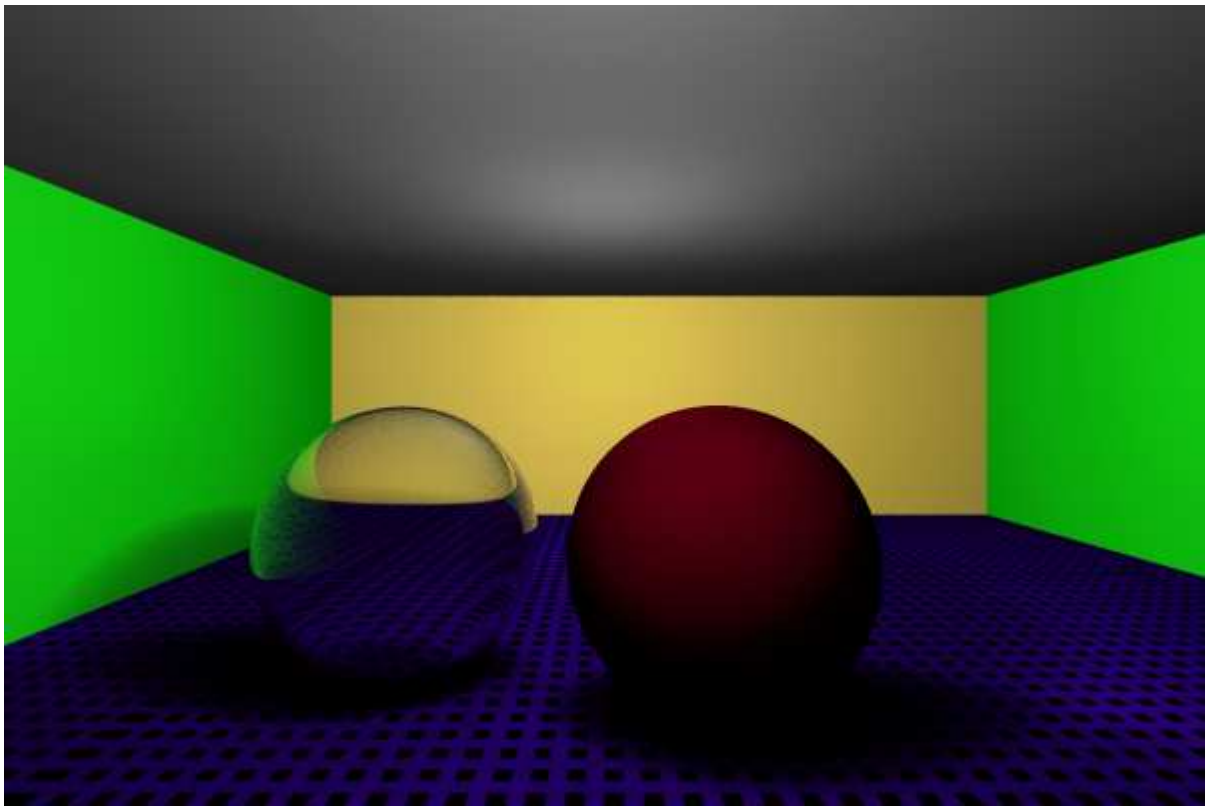
DRT_DRT, tato třída konečně řeší vlastní algoritmus Distributed Ray Tracingu. Mimo něj v souboru s touto třídou najdete i realizace dílčích algoritmů - jako algoritmus pro hloubku zaostření, rozmazání pohybem, výpočet osvětlení podle Lamberta a další.

DRT_View v tomto souboru je funkce main, ta volá metody třídy DRT_DRT. Následně uloží obrazek ve formátu BMP.

7 Testování, hodnocení výsledků

V této předposlední kapitole uvedu několik obrázků vygenerovaných mým Distributed Ray Tracingem. U každého obrázku uvedu statistické údaje při jeho vzniku. Následně obrázek okomentuji a zhodnotím výsledek.

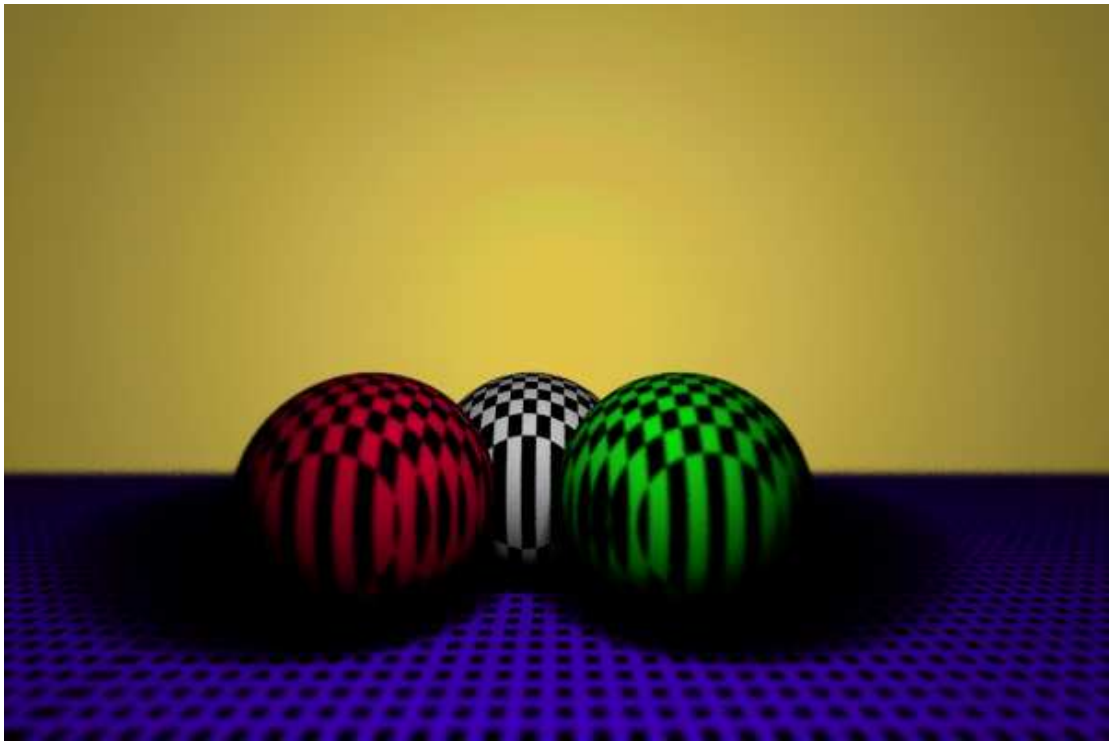
Testováni jsem prováděl na svém notebooku ASUS FJ3C.



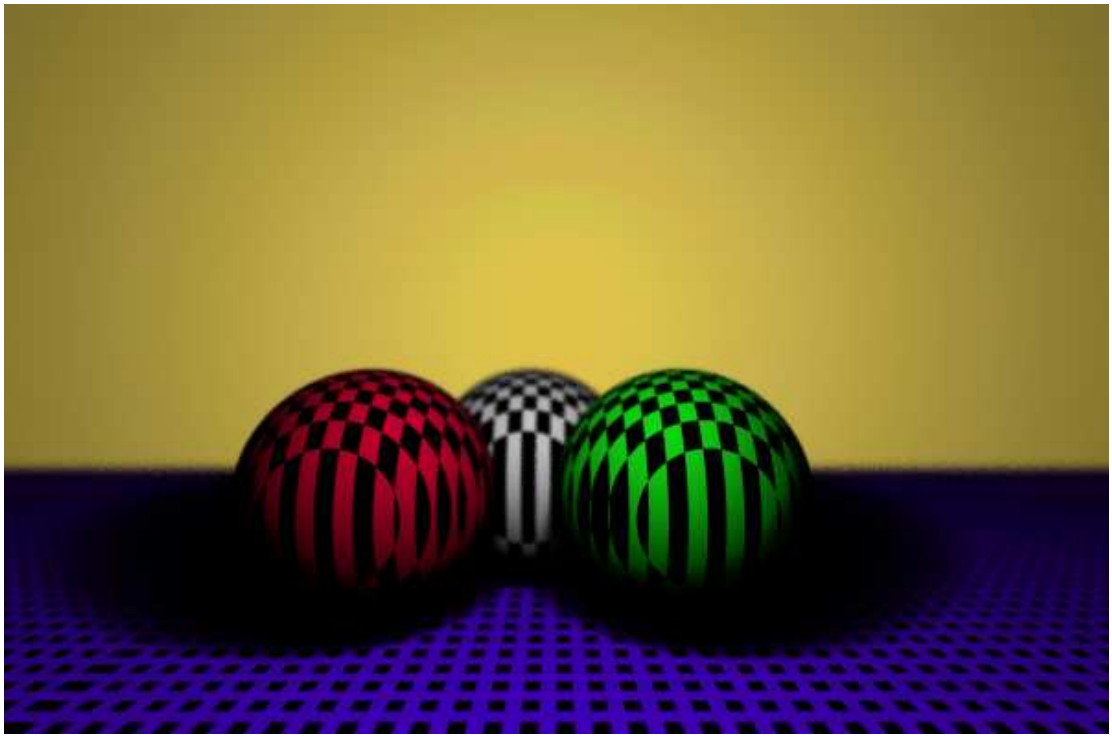
Obr. 29– Test1, počet vzorků na pixel:100, stínových paprsků: 10, čas renderování: 11 minut 43 sekund

Scéna obsahuje pět rovin difusního povrchu, jedna rovina je pokryta procedurální texturou. Dále je ve scéně jedna difusní koule a jedna koule skleněná. O osvětlení ve scéně se starají dvě světla. Depth of field i Motion Blur je vypnut.

Podle mě se jedná o velmi zdařilý obrázek, kde lze velmi dobře pozorovat lom světla a měkké stíny.

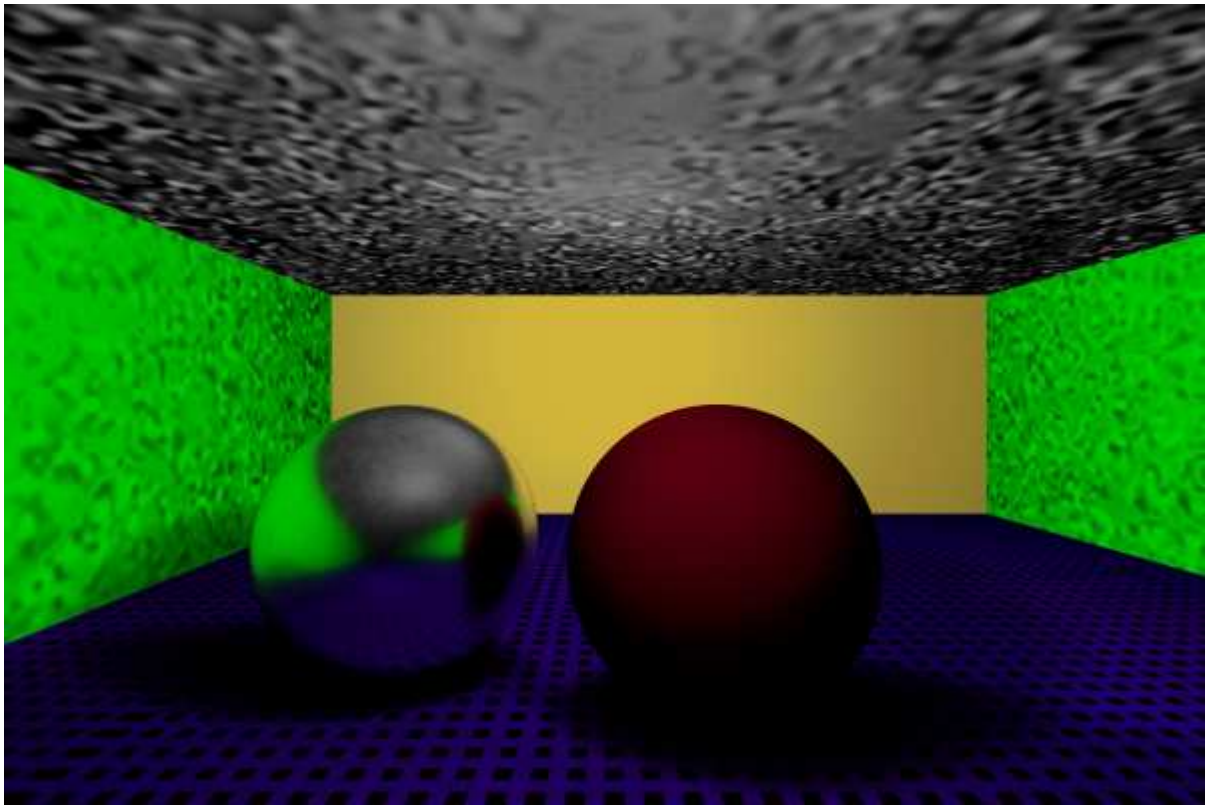


Obr. 30– Test2, počet vzorků na pixel:100, stínových paprsků: 10, čas renderování: 10 minut 53 sekund



Obr. 31– Test3, počet vzorků na pixel:100, stínových paprsků: 10, čas renderování: 11 minut 08 sekund

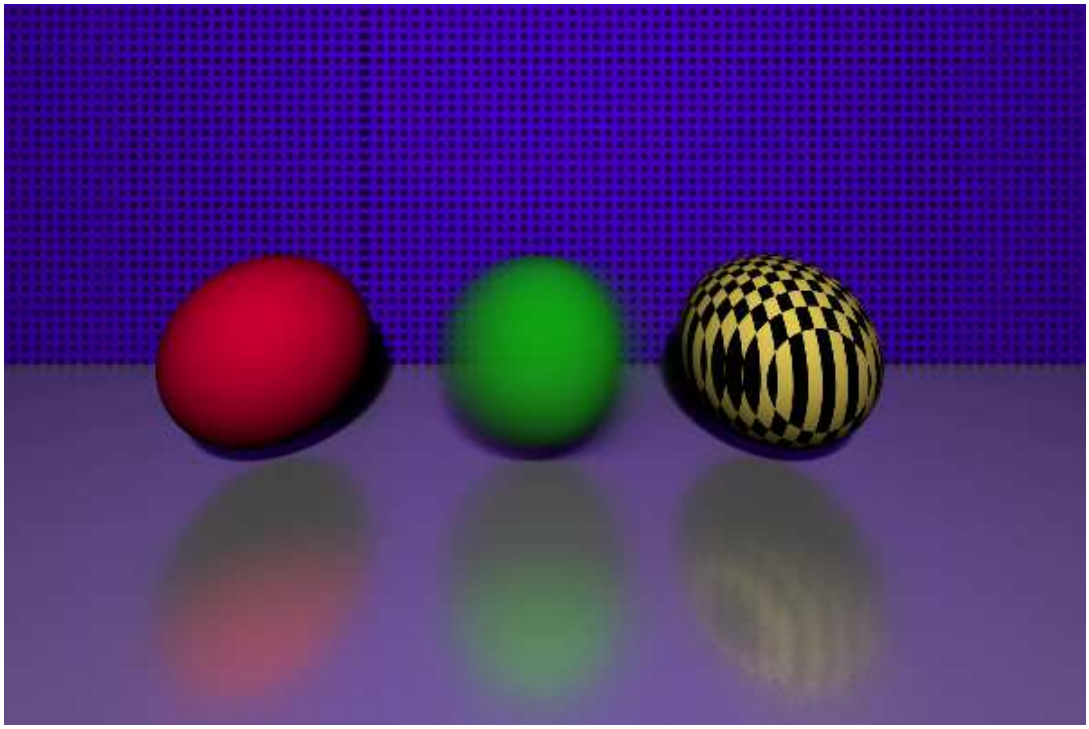
Obrázky 30 a 31 obsahují dvě difusní roviny a tři difusní koule. Většina objektů má aktivované procedurální textury. Oba obrázky ukazují efekt Depth of Field. Na prvním obrázku je čočka zaostřena na zadní bílou kouli, na spodním obrázku jsem přeostřil na přední koule. Hloubka ostrosti je zde jasně znatelná.



Obr. 32– Test4, počet vzorků na pixel:400, stínových paprsků: 15, čas renderování: 70 minut 12 sekund

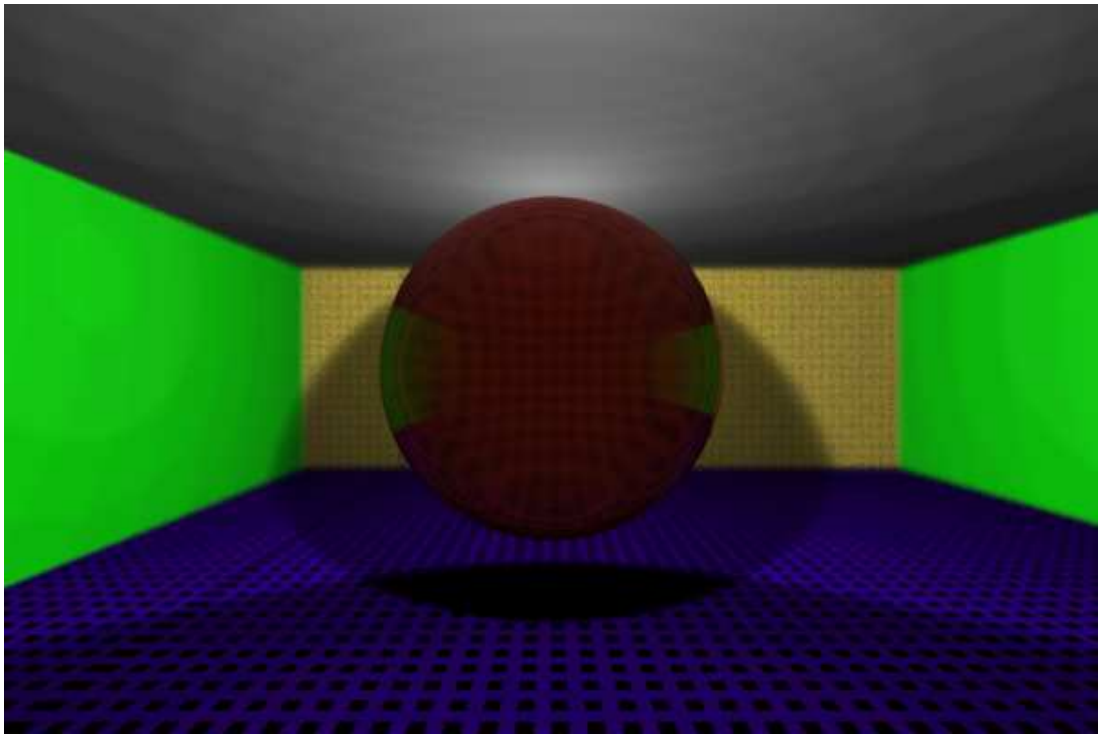
V této scéně je pět rovin, jedna rovina je pokryta procedurální texturou. Na třech rovinách je zapnut bump-mapping. Dále je ve scéně jedna difusní koule a jedna koule matně odrazivá. O osvětlení ve scéně se starají dvě světla. Depth of field i Motion Blur je vypnut.

U tohoto obrázku kvalita nevyvážila čas nutný pro renderování. Ukazalo se, že 400 vzorků na pixel je už zbytečně mnoho. Na výsledku si cením matného odrazu na kovové kouli, měkkých stínu a bump-mappingu.



Obr. 33– Test5, počet vzorků na pixel:120, stínových paprsků: 8, čas renderování: 12 minut 14 sekund

Tato scéna se skládá z tří difusních koulí. Jedna je v pohybu (zelená). Dále je zde matně odrazivá podlaha. Osvětlení je jedním zdrojem světla. Je aktivováno mírné rozostření pomocí Depth of Field.



Obr. 34– Test6, počet vzorků na pixel:120, stínových paprsků: 8, čas renderování: 14 minut 14 sekund

Tento obrázek uvádím kvůli červené průhledné kouli, která barví pozadí do červena. Dva polostíny vržené koulí splývají ve stín. Ve scéně je navíc aktivováno rozostření.

8 Závěr

Díky této diplomové práci jsem porozuměl metodě realistického zobrazování Distributed Ray Tracing. Následně jsem tuto metodu realizoval v programovacím jazyku C/C++. Z hlediska dosažených výsledků považuji svou práci za zdařilou. Myslím si, že se výsledky mé práce kvalitativně přiblížily k obrázkům pana R.L.Cooka. U některých scén si troufám říci, že obrázky vyrenderované mou implementací Distributed Ray Tracingu vykazují dokonce lepší výsledky. Především ve scénách řešících odraz a lom světla, kterým jsem věnoval ve své práci maximum času. V části Distributed Ray Tracingu řešící rozmazání objektu pohybem se mi podařila implementace, ale nejsem spokojen s výsledným zobrazením scény. Domnívám se, že pan R.L. Cook pro pohybující se objekty použil dokonalejší popis pohybujících se objektů.

V rámci pokračování na tomto projektu navrhuji přidání dalšího objektu - trojúhelníku. S jeho pomocí lze modelovat všechny 3D objekty. Tím by se projekt obohatil o možnost zobrazení všeho kolem nás. Dalším přínosným krokem by bylo přidání nových specifických materiálů a jim příslušných BRDF funkcí. Takto vylepšené realistické zobrazování by výsledným scénám dalo reálnou podobu. V minulosti jsem již implementoval zobrazovací metodu Photon Mapping. Do budoucna bych chtěl využít zkušeností se znalosti obou metod, sloučit jejich výhody, a tím dosáhnout co nejreálnějšího zobrazení scén.

Literatura

[1] Robert L Cook, Stochastic sampling in computer graphics ACM Transactions on Graphics , 1986.

[2] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. Computer Graphics.1984, ACM Siggraph.

[3] Wikipedia: http://en.wikipedia.org/

[4] Peter Shirley, Changyaw Wang: Distribution Ray Tracing Theory and Practice. Computer Science Department,Indiana University.

[5] Allen Martin. Distributed Ray Tracing. http://web.cs.wpi.edu/~matt/courses/cs563/talks/dist_ray/dist.html

[6] E. Veach and L. J. Guibas: Metropolis Light Transport, Addison-Wesley, (SIGGRAPH'97), August 1997.

[7] E. Veach. Robust Monte Carlo methods for light transport simulation. Stanford university, 1997.

[8] Zdena Rábová, Milan Češka, Jaroslav Zendulka, Petr Peringer, Vladimír Janoušek, Modelování a simulace.

[9] Martin Krahulík. Reprezentace BRDF v počítačové grafice. Diplomová práce. https://dip.felk.cvut.cz/browse/pdfcache/krahulik_2005dipl.pdf

[10] Zdeněk Jelínek. Algoritmy pro výpočet globálního osvětlení. Diplomová práce.

[11] Václav Hošek. Zobrazování scén metodou photon tracing. Bakalářská práce.

[12] Box Muller, wikipedie, otevřená encyklopedie, http://en.wikipedia.org/wiki/Box-Muller_transformation.

[13] Bram de Greve, Reflections and Refractions in Ray Tracing, http://www.flipcode.com/archives/reflection_transmission.pdf, 2004.

[14] Spherical coordinate system, wikipedie, otevřená encyklopedie, http://en.wikipedia.org/wiki/Spherical_coordinates.

Seznam obrázků

Přílohy

1. CD s dokumentací a zdrojovými kódy