VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

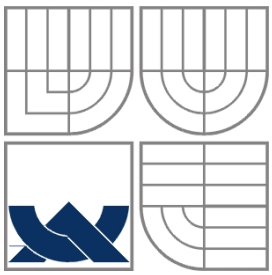
MINIMALISTICKÝ OBJEKTOVĚ ORIENTOVANÝ „RAY TRACER“

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

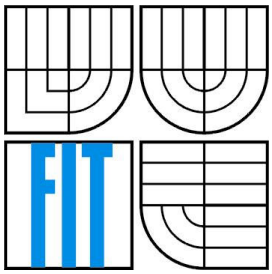
AUTOR PRÁCE
AUTHOR

BC. MÁRIO ROŽENSKÝ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MINIMALISTICKÝ OBJEKTOVĚ ORIENTOVANÝ „RAY TRACER“

MINIMALISTIC OBJECT ORIENTED „RAY TRACER“

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MÁRIO ROŽENSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2008

Abstrakt

Práce podává přehled o vykreslování scény pomocí metody ray tracing. Zabývá se jednotlivými aspekty při tvorbě aplikace využívající tuto metodu, jako jsou výpočty průsečíků, výpočet osvětlení apod. Je zde popsán základní algoritmus výpočtu jednoho snímku scény. Detailně jsou rozebrány jednotlivé třídy objektového návrhu. U každé je podrobný popis co daná třída dělá, proč byla do modelu zařazena a jsou vysvětleny důležité metody, které daná třída používá. Součástí je také ukázková aplikace demonstrující využití modelu v praxi a jeho snadnou použitelnost.

Klíčová slova

Ray tracing, ray tracer, osvětlovací model, osvětlení, průsečík, CSG, model tříd

Abstract

This thesis brings an overview about scene rendering using the ray tracing method. It describes aspects used when creating the application which uses this method such as intersection computation, lighting and shading models etc. It also describes the basic algorithm used for rendering one frame. Each class of the object oriented design is described. There is also detail explanation what is the purpose of the each class in the model and what are the most important used methods. The work also contains demonstration application showing the usage of model in practice.

Keywords

Ray tracing, ray tracer, shading model, intersection, CSG, class diagram

Citace

Bc. Roženský Mário: Minimalistický objektově orientovaný „ray tracer“. Brno, 2008, Diplomová práce, FIT VUT v Brně.

Minimalistický objektově orientovaný „ray tracer“

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. Mário Rožesnký
20.4.2008

Poděkování

Ing. Adam Herout, Ph. D. – významná pomoc při návrhu modelu tříd a odborné konzultace
Lukáš Michl – rady a zkušenosti s vlastní implementací raytraceru

Obsah

Seznam obrázků

1 Úvod

V počítačové grafice se vykreslována scéna skládá z různě složitých objektů, světel, materiálů. Jsou známy různé metody, jak takovou scénu převést na monitor počítače, neboli převést trojrozměrný prostor do dvojrozměrného. Tyto metody se liší dle své výpočetní náročnosti, a tím i výsledné kvality obrazu. Rozeznáváme metody v reálném čase (tyto jsou většinou urychlovány grafickým hardwarem) a metody, jenž jsou sice pomalejší ve výpočtu, ovšem kvalita dosaženého výsledného obrazu je mnohonásobně vyšší a daleko více se blíží realitě. Mezi tyto metody patří také ray tracing. Jinými metodami, jsou například radiosita nebo photon mapping.

Tato práce se bude zabývat ray tracingem. Nejprve po stránce teoretické, a to popisem metody, jednotlivých jejích součástí, jako jsou průsečíky, výpočet osvětlení apod. Dále bude uveden návrh modelu tříd pro minimalistický ray tracer. Jednotlivé třídy budou podrobně popsány a jejich existence v modelu odůvodněna. Také budou okomentovány některé důležité metody daných tříd.

Tato diplomová práce navazuje na Semestrální projekt a na předchozí zkušenosti s psaním 3D aplikací a raytraceru. V Semestrálním projektu byl navržen model tříd, který byl dále upravován a rozšiřován. Dále bylo navázáno na testovací aplikaci, která byla přepsána dle nového modelu a v mnohém vylepšena.

Práce bude rozdělena na kapitoly a doplněna o obrázkovou přílohu výstupů z vytvořené aplikace.

Druhá kapitola popisuje ray tracing. Nejprve je v ní vysvětlen výpočet osvětlovacího modelu, dále difúzní, odražená a ambientní světla a jejich význam pro výsledný obraz. Poté je popsán postup výpočtu ray traceovaného obrazu s okomentovaným pseudokódem algoritmu. Dále je zde uvedeno praktické využití ray tracingu v moderních aplikacích, jako jsou různé CAD systémy, a také ve filmovém průmyslu. Jsou zde uvedeny ukázky aplikací. Následuje ukázka výpočtu průsečíků s jednoduchými tělesy a také složitější výpočty při konstrukci CSG těles. Následuje matematický základ pro výpočet odražených a lomených paprsků a také pro výpočet směru primárního paprsku.

Ve třetí kapitole je rozebrán objektový návrh ray traceru. Nejprve je uveden návrh tříd pro minimalistický ray tracer a jsou detailně popsány jednotlivé třídy spolu s rozebráním důležitých metod dané třídy. Nejprve třída Raytracer, jež je hlavní třídou, která provádí hlavní vykreslovací cyklus ray tracingu. Následuje třída Transform, jež se stará o transformace pozice objektů a jejich otočení. Poté je popsána třída Material, která uchovává barevné a strukturální informace o objektu, jako např. barvu nebo texturu, ale také i informace, o tom, zda objekt je průhledný a jak láme či odráží světlo. Následuje tříа Scene, uchovávající seznam objektů a světel, které se nachází v prostoru a počítající průsečíky s objekty. Další třídou je Camera, tato se stará o generování primárního paprsku v závislosti na parametrech „čočky“. A nakonec jsou popsány třídy Light, Object a Shape.

Čtvrtá kapitola pojednává o možných rozšířeních a vylepšeních, která jsou používána v ray tracingu. Základním prvkem pro popis barvy povrchu objektu jsou textury, proto je jim věnován úvod. Následují metody anti-aliasingu pro vyšší kvalitu obrazu. Posledním tématem jsou možnosti urychlení výpočtu.

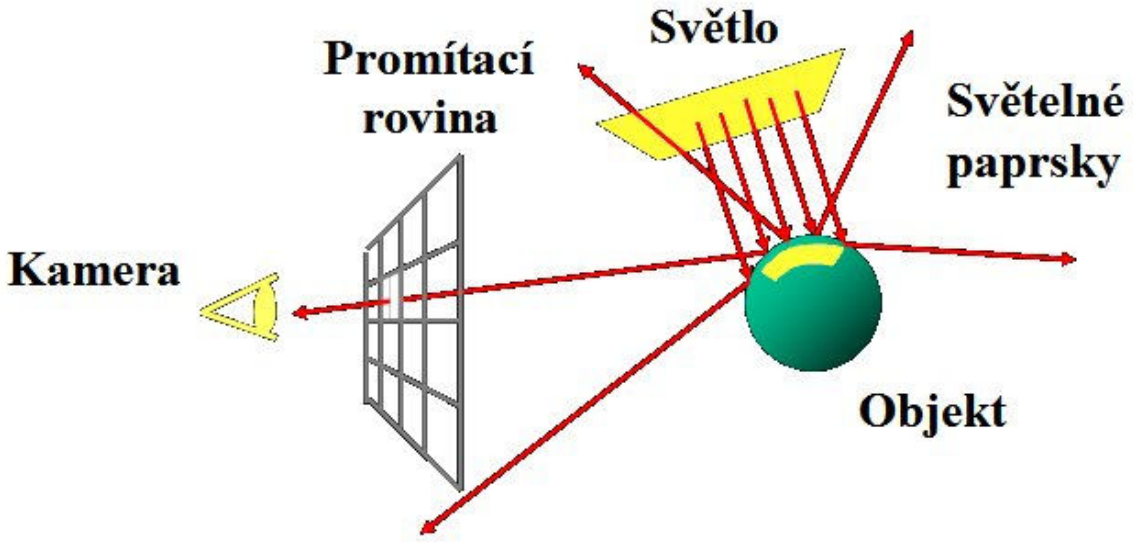
Pátá kapitola je věnována vlastní implementaci modelu v praxi. Je zde popsána aplikace, její funkce. Dále jsou provedeny testy výkonnosti. Dosažené výsledky jsou okomentovány a zdůvodněny. Také jsou zde dány návrhy na možná rozšíření této aplikace do budoucna.

V závěru jsou diskutovány dosažené výsledky a je shrnuta a zhodnocena celá práce.

Jako poslední součást práce bude obrázková příloha ve formě obrázků s popisem, které byly vytvořeny v demonstrační aplikaci.

2 Ray tracing

Ray tracing se překládá do češtiny jako sledování paprsku. Jedná se o algoritmus pro převod prostorové reprezentace scény a objektů do dvourozměrného obrazu. Tato metoda je založena na fyzikálním principu průchodu světla prostorem. Šíření světla v prostoru je možné zjednodušit na šíření pomocí nekonečného množství paprsků vycházejících ze světelných zdrojů a šířících se prostorem. Při ray tracingu tohoto využijeme a budeme tyto paprsky sledovat a ty, jenž dopadnou do pozorovací roviny, zobrazíme. Ovšem touto metodou bychom získali velké množství paprsků, které nikdy pozorovací rovinu neprotnou, proto využijeme opačný postup a budeme vrhat paprsky z pozice pozorovatele pozorovací rovinou a sledovat jejich interakci s objekty ve scéně. Při této interakci může dojít k odražení, či lomu paprsku. Tyto interakce mohou nastat rekurzivně až do maximálního počtu odrazů, nebo pokud vlivem útlumu dojde k poklesu intenzity pod určitou mez. Protože ve scéně vzniká velké množství paprsků, a takovýto výpočet je tudíž velmi časově náročný, existují ray tracery v reálném čase, ovšem jsou spíše experimentální, vykreslují scénu v nižším rozlišení apod. Ovšem pro využití všech předností této metody a její vysoké realističnosti je potřeba velkého výpočetního výkonu, a tudíž i dlouhého času výpočtu. Možnosti využití této metody jsou nesčetné. Od vizualizací v technických a návrhových aplikacích až po filmový průmysl.



Obr. 2-1: Raytracing

Ray tracing je nástupce ray castingu. Tento byl poprvé prezentován Arthurem Appelem v roce 1968. Principem této metody je vrhání paprsků z kamery směrem do scény a hledání nejbližšího objektu, se kterým se paprsek protne. Pokud dojde k protnutí, dochází k výpočtu osvětlení v tomto bodě na základě dopadajících světel a materiálu objektu. Tato metoda neprodukuje žádné sekundární paprsky.

Rozšíření předchozího algoritmu prezentoval Turner Whitted v roce 1979. Na rozdíl od předchozího algoritmu toto rozšíření neskončí při nalezení prvního průsečíku, ale pokračuje dále. V místě dopadu mohou vzniknout až tři nové paprsky: odrazový, zlomový a stínový. Odražený paprsek se odráží podle zákona o odrazu (úhel dopadu se rovná úhlu odrazu). Tento paprsek je dále sledován k nejbližšímu průsečíku s jiným tělesem, které se bude následně zrcadlit v původním objektu. Zalomený paprsek se láme dle „Snellova zákona o lomu“ a může se zlomit až dvakrát – na vstupu a výstupu z tělesa. Paprsek je sledován podobně, jako v případě odrazu. Posledním typem paprsku je paprsek stínový. Tento paprsek je využit k zjištění, zda objekt není zastíněn nějakým jiným tělesem. Vytvoří se nové paprsky z průsečíku ke světlům, a pokud neprotnou žádné jiné těleso, objekt je daným světlem nasvícen.

Metoda se během 80. a 90. let stále vyvíjela a díky stále vyššímu výpočetnímu výkonu bylo možno počítat stále rozsáhlejší scény ve stále lepší kvalitě. Ray tracery byly doplňovány o různé funkce a také spojovány s jinými technikami pro vykreslování scény, jako například radiozitou. Toto má za následek kombinaci typických vlastností obou metod. Z ray tracingu jsou to zejména odrazy na lesklém povrchu a z radiozity to jsou měkké stíny a globální osvětlení. Výsledek je možný vidět na obrázku.

Obr. 2-2: a) ray tracing b) radiozita c) kombinace ray tracingu a radiozity

V současnosti v souvislosti s mohutným nástupem grafických akcelerátorů a jejich ohromného výpočetního výkonu dochází k pokusům o vytvoření real-time ray traceru pomocí GPU. Nedávno také společnost Intel představila novou architekturu grafických procesorů, jenž by měla do budoucna umožnit přechod od vykreslování scény pomocí rasterizace na vykreslování pomocí raytracingu.

2.1 Osvětlovací modely

2.1.1 Phongův osvětlovací model

Pro výpočet barvy objektů se nejčastěji využívá Phongův osvětlovací model. Tento model se snaží co nejvíce zjednodušit fyzikální a optické vlastnosti reálného světla tak, aby byl výpočet co nejrychlejší a výsledný obraz co nejrealističtější. V tomto modelu se dopadající (a odražené) světlo rozkládá na tři složky.

Tyto složky jsou – ambientní, difúzní a odlesk. Ambientní složka vyjadřuje okolní světlo scény, které není vyzařováno žádným konkrétním světleným zdrojem. Vzniká tak, že se světlo ve scéně několikrát odrazí (například v uzavřené místnosti od stěn), a tudíž není patrný směr, odkud přichází. V Phongově modelu považujeme toto světlo za všesměrové – osvětluje objekty ze všech stran stejně. Pokud je těleso osvětleno pouze touto složkou, ztrácíme pocit prostorovosti.

Druhou složkou je složka difúzní, která vyjadřuje světlo dopadající z určitého světelného zdroje a odrážející se stejnou měrou do všech směrů. Pokud objekt neobsahuje odlesky, ale pouze difúzní složku, jeví se pozorovateli, jako by bylo vytvořeno z matného materiálu. Při výpočtu nezáleží na pozici kamery, ale pouze na vzájemné poloze světla a objektu.

Poslední složkou jsou odlesky. Odlesky vznikají při dopadu paprsku ze světelného zdroje a jeho následném odrazu dle zákona odrazu (úhel dopadu se rovná úhlu odrazu). Ve skutečnosti se paprsek nikdy neodrazí ideálně, vždy dochází k rozptýlení. Phongův osvětlovací model toto modeluje pomocí koeficientu změny intenzity odraženého světla podle úhlu mezi paprskem odraženým a paprskem ideálně odraženým. Tato složka je závislá na poloze kamery, tělesa a světla.

Výsledné barva a osvětlení se spočítá podle následujícího vztahu:

$$\mathbf{I} = \mathbf{c_a}\,\mathbf{I_a} + \mathbf{c_d}\,\mathbf{I_d}\,(\mathbf{NL}) + \mathbf{c_s}\,\mathbf{I_s}\,(\mathbf{VR})^{\mathbf{n}}$$

Význam jednotlivých proměnných:

I – Celková výsledná intenzita světla. Je počítána pro každou RGB složku zvlášť.

$\mathbf{I_a}$ – Intenzita ambientní složky světla. Tato intenzita je pro všechny objekty ve scéně shodná.

$\mathbf{I_d}$ – Intenzita difúzní složky světla. Tato složka je závislá na vzájemné poloze normálového vektoru **N** v bodě dopadu a vektoru dopadu světla **L**.

$\mathbf{I_s}$ – Intenzita odlesku. Hodnota této složky závisí na vzájemné poloze světelného zdroje, povrchu tělesa a pozici pozorovatele.

$\mathbf{c_a}$ – Koeficient určující, v jaké míře objekt odráží ambientní světlo. Udává se v rozmezí 0–1. 0 znamená, že povrch ambientní světlo neodráží vůbec.

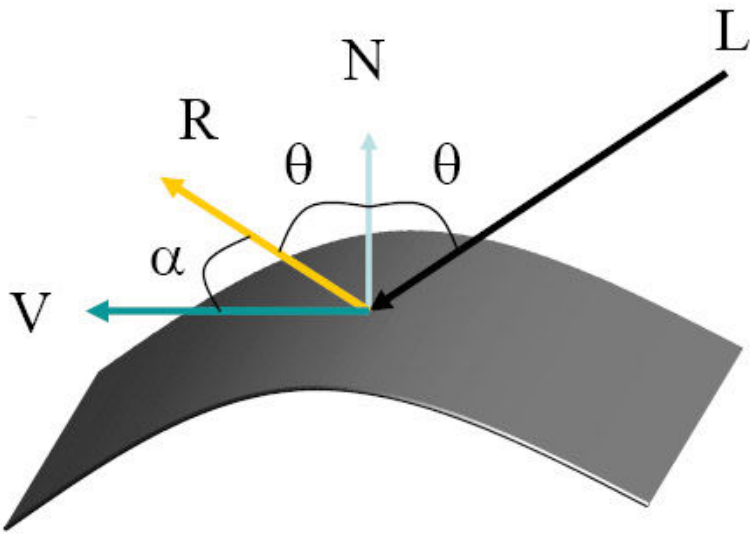
c_d - Koeficient určující, v jaké míře objekt odráží difúzní světlo. Udává se v rozmezí 0–1. 0 znamená, že povrch difúzní světlo neodráží vůbec.

c_s - Koeficient odlesků. Nastavuje se opět v rozmezí 0–1.

NL – Tento součin určuje intenzitu dopadajícího světla. Vektor **N** vyjadřuje normálu v bodě dopadu a vektor **L** směr dopadajícího světla. Pokud jsou tyto vektory lineárně závislé (rovnoběžné) – světelný zdroj je umístěn kolmo nad tělesem, je tento součin maximální, a tudíž i osvětlení v tomto bodě má difúzní složku nejintenzivnější. Před výpočtem tohoto skalárního součinu je potřeba oba vektory normalizovat.

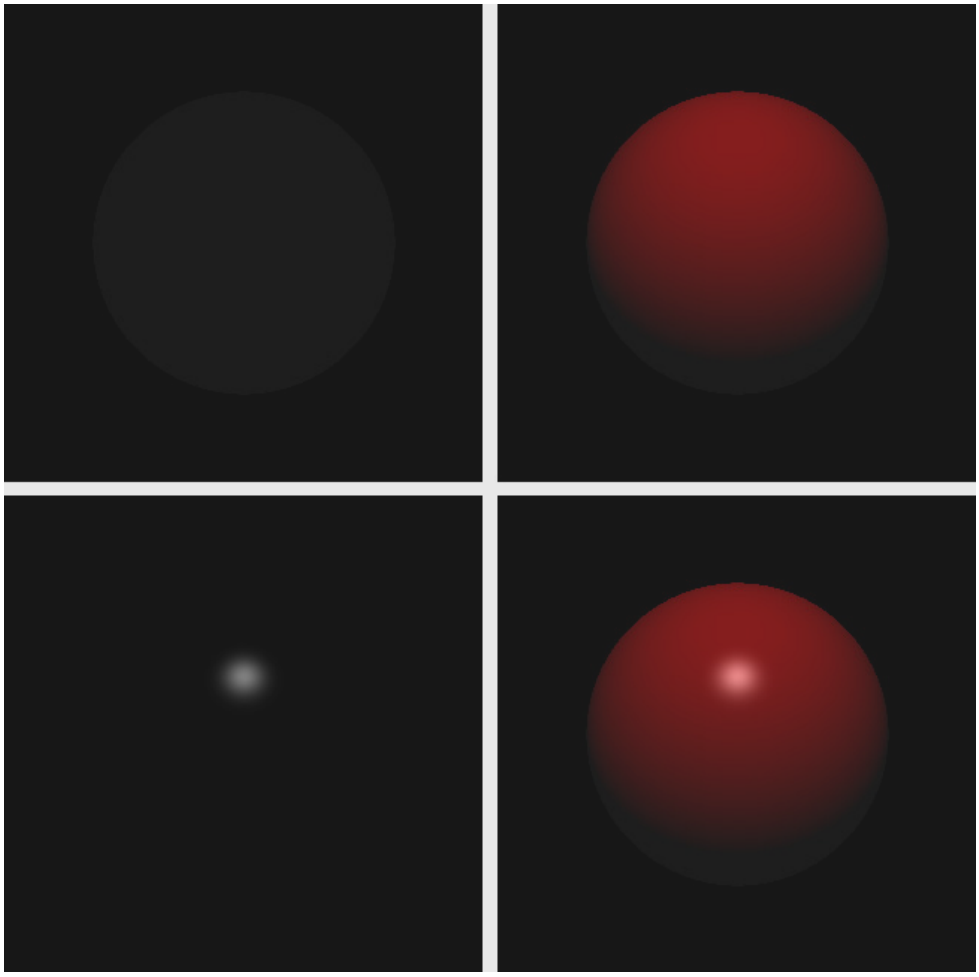
VR – Vektor **V** určuje směr ke kameře a vektor **R** je směr ideálně odraženého paprsku. Odražený paprsek lze spočítat ze vztahu: **R = 2 * (N.L) * N – L**. Stejně jako v předchozím případě oba vektory musí být normalizovány.

^{n} – exponent u výrazu **(VR)^n**, udávající míru lesklosti tělesa. Matné těleso bude mít exponent roven nule. Vysoce lesklé těleso například 64. Čím vyšší exponent, tím jsou odlesky ostřejší a intenzivnější.



Obr. 2-3: Phongův osvětlovací model – vektory

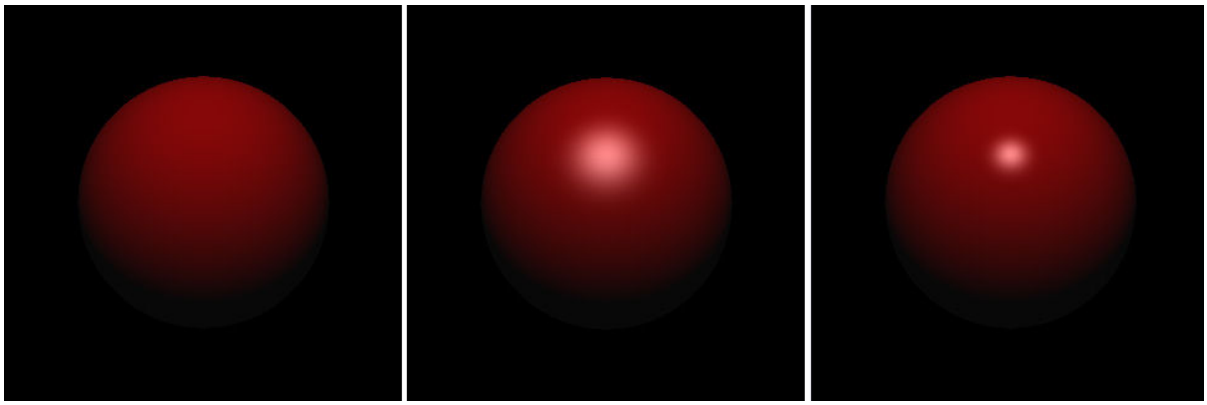
Zdroj: [5]



Obr. 2-4: Ukázka jednotlivých složek světla

Ambientní, difúzní

Spekulární, kombinace všech tří

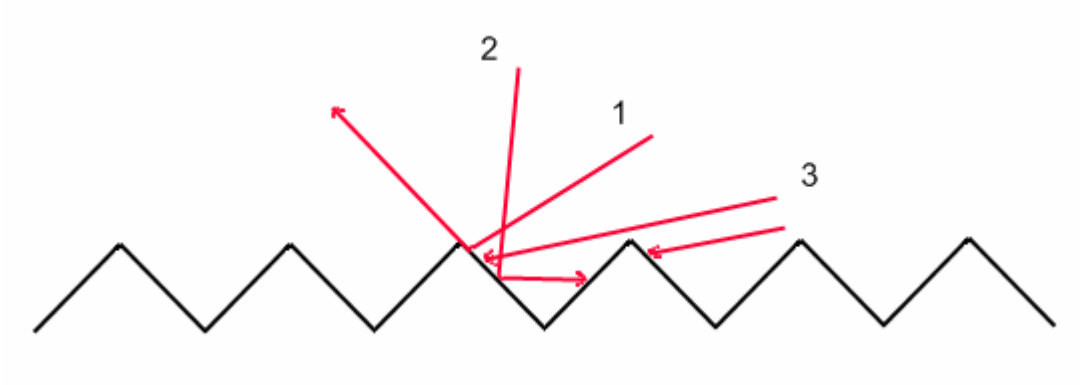


Obr. 2-5: Ukázka vlivu exponentu ‚n‘ na „ostrost“ odraženého světla,

exponenty jsou zleva, bez specularu, 8, 32

2.1.2 Torrance-Sparrow osvětlovací model

Povrch tělesa vykresleného za použití phongova modelu budí dojem, že je vyroben z plastu. Ovšem pro modelování povrchu například z kovu je vhodnější jiný model a to Torrance-Sparrow. Podstatou modelu je, že povrch je složen z velkého množství miniaturních plošek, které jsou různě orientované a tudíž v různém směru odráží rozdílné množství světla. Tímto modelem je simulována nepravidelná fyzická struktura povrchu tělesa.



Obr. 2-6: Paprsky dopadající na mikroplošky
1) paprsek se odrazí 2) maskování 3) stíněni

Orientace mikroplošek je popsána statistickým(např. Gaussovým) rozdělením. Vyhodnocení m tohoto rozdělení pro daný úhel získáme počet plošek, které mohou přispívat k intenzitě odrazu v daném směru.

Přiblíží-li se směrový vektor pozorovatele, či světla k rovině průměrné plochy, dochází k interferenčním efektům.

1) **maskování** – nastává, přiblíží-li se k rovině plochy směrový vektor pozorovatele. Část odraženého světla je zachyceno protilehlou ploškou.
2) **stínění** - nastává, přiblíží-li se k rovině plochy směrový vektor dopadajícího světla. Část dopadajícího světla je zastíněna.

Výsledné barva a osvětlení se spočítá podle následujícího vztahu:

$$\mathbf{I = c_a I_a + [c_d (NL) + (DGF/NV)]}$$

Význam jednotlivých proměnných:

I – Celková výsledná intenzita světla. Je počítána pro každou RGB složku zvlášť.

$\mathbf{I_a}$ – Intenzita ambientní složky světla. Tato intenzita je pro všechny objekty ve scéně shodná.

NL – Tento součin určuje intenzitu dopadajícího světla. Vektor **N** vyjadřuje normálu v bodě dopadu a vektor **L** směr dopadajícího světla. Pokud jsou tyto vektory lineárně závislé (rovnoběžné) – světelný zdroj je umístěn kolmo nad tělesem, je tento součin maximální, a tudíž i osvětlení v tomto bodě má difúzní složku nejintenzivnější. Před výpočtem tohoto skalárního součinu je potřeba oba vektory normalizovat.

D – distribuční funkce orientace mikroplošek

G – množství mikroplošek, které se stíní, či maskují navzájem

F – Fresnelův zákon odrazu

$\mathbf{c_a}$ – Koeficient určující, v jaké míře objekt odráží ambientní světlo. Udává se v rozmezí 0–1. 0 znamená, že povrch ambientní světlo neodráží vůbec.

$\mathbf{c_d}$ - Koeficient určující, v jaké míře objekt odráží difúzní světlo. Udává se v rozmezí 0–1. 0 znamená, že povrch difúzní světlo neodráží vůbec.

Obr. 2-7: Výsledek stínovaní pomocí modelu Torrance-Sparrow

2.2 Popis algoritmu

Základní algoritmus sledování paprsku je cyklus, který projde všemi body promítací roviny a vrhne paprsek z pozice pozorovatele směrem do scény. Tyto paprsky jsou sledovány, a pokud jsou zaznamenány jejich interakce s objekty ve scéně, jsou vytvořeny nové, tzv. sekundární paprsky, na základě vlastností objektů a proces sledování paprsku probíhá znovu. Algoritmus skončí po určitém počtu odrazů či lomů. Výsledná barva pixelu v daném bodě na promítací rovině je dána součtem všech odražených a zalomených paprsků vzniklých z primárního paprsku interakcí s tělesy.

Typy paprsků vznikajících ve scéně:

Primární paprsek – paprsek vyslaný od pozorovatele směrem do scény

Sekundární paprsek – paprsek vzniklý odrazem nebo lomem při interakci s tělesy

Stínový paprsek – paprsek vyslaný z místa dopadu ke světelným zdrojům, pro zjištění, leží-li ve stínu. Neleží-li objekt ve stínu, je pro něj vyhodnocen osvětlovací model

2.2.1 Slovní popis

Při vržení paprsku směrem do scény testujeme tento paprsek se všemi objekty ve scéně, což je značně časově náročné (zde je možné urychlit výpočty pomocí určitého rozdělení scény, např. pomocí obalových těles či oct-tree). Každý průsečík paprsku s objektem generuje stínový paprsek a případně dle vlastností materiálu i paprsky odražené, či lomené. V každém takovém průsečíku se provádí ty samé výpočty, je tedy vhodné pro implementaci ray tracingu využít rekurzi.

Pro ukončení rekurze je možné použít následující podmínky:

1. paprsek narazí na difúzní povrch
2. je dosažena předem stanovená hloubka rekurze
3. energie paprsku klesne pod určitý práh

V každém průsečíku paprsku **R** platí:

$$\mathbf{I(R) = I_{local}(R) + I_{global}(R) = I_l(R) + c_r I(R_r) + c_t I(R_t)}$$

Význam jednotlivých proměnných:

I – intenzita v místě průsečíku
R – průsečík paprsku a objektu
$\mathbf{R_r}$ – průsečík odraženého paprsku
$\mathbf{R_t}$ – průsečík zalomeného paprsku
$\mathbf{c_r}$ – koeficient odrazivosti objektu
$\mathbf{c_t}$ – koeficient propustnosti objektu

Z rekurzivního charakteru této rovnice je zřejmá vhodnost volby tohoto přístupu pro implementaci ray tracingu.

2.2.2 Algoritmizace

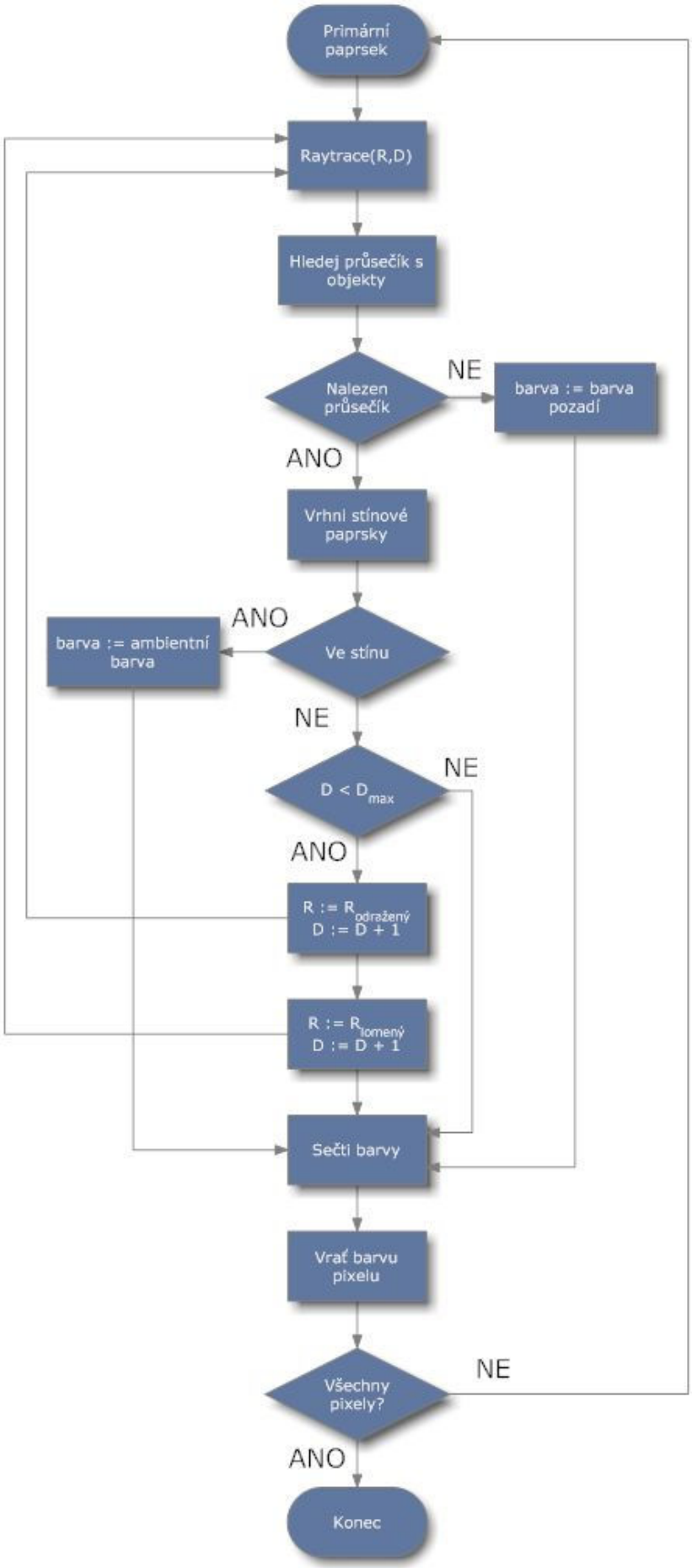
Rekurzivní funkce **Raytrace (R, D)** s parametry **R** paprsek a **D** hloubka rekurze

1. Najdi průsečík **I** mezi **R** a objektem
2. Jestliže průsečík **I** neexistuje, paprsek jde mimo scénu a má barvu pozadí
3. Z **I** vyšli ke světlům stínové paprsky
4. Vyhodnoť součet osvětlovacích modelů v bodě **I** pro nezastíněná světla
5. Pokud není překročena hloubka rekurze **D**, vyšli z **I**:
    a. Odražený sekundární paprsek voláním **Raytrace ($\mathbf{R_r}$, D+1)**
    b. Lomený paprsek **Raytrace ($\mathbf{R_t}$, D+1)**
6. Paprsek má barvu danou součtem barvy osvětlením od odraženého paprsku $\mathbf{R_r}$ a lomeného paprsku $\mathbf{R_t}$

Počet volání funkce **Raytrace** je řízen parametrem **D**. Pokud je nastaven na 1, jedná se o ray casting – vyhodnocují se pouze primární paprsky. Pro zobrazení odrazů musí být hodnota minimálně 2 a pro práci s průhlednými tělesy minimálně 3.

2.2.3 Vývojový diagram

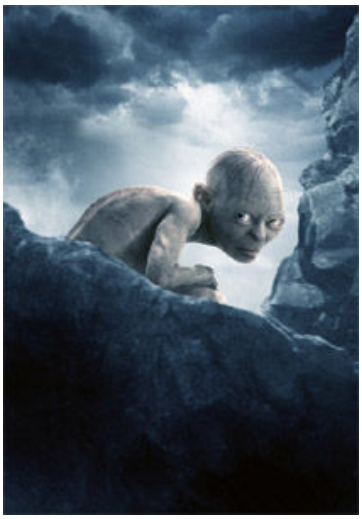
Algoritmus ray tracingu se dá také zapsat vývojovým diagramem


Primární paprsek
Raytrace(R,D)
Hledej průsečík s objekty
Nalezen průsečík
NE
barva := barva pozadí
ANO
Vrhni stínové paprsky
ANO
barva := ambientní barva
Ve stínu
NE
D < Dmax
NE
ANO
R := Rodražený
D := D + 1
R := Rlomený
D := D + 1
Sečti barvy
Vrať barvu pixelu
Všechny pixely?
NE
ANO
Konec

2.3 Využití

Možnosti využití ray tracingu jsou široké. Využívá se v nespočetném množství komerčních aplikací, u kterých je vyžadována vysoká kvalita výsledného obrazu a není požadováno vykreslování v reálném čase. Další výhodou je vysoká kvalita stínů a odrazů světla na lesklém povrchu, kde je možno vykreslit několikanásobný odraz světla, což je při klasických metodách vykreslování téměř nemožné. Hlavní využití je v CAD aplikacích při zobrazování složitých nákresů a plánů. Ať už se jedná o plán několikapatrového mrakodrapu nebo o tištěný spoj, je vždy výhodné využít ray tracing pro zobrazení.

Dalším okruhem aplikací, kde se využívá ray tracingu, je filmový průmysl. Mnoho moderních filmů, jako například Matrix, Pán prstenů apod. využívá pro vykreslení realisticky vypadajících postav aplikaci Maya či 3D Studio MAX, jež využívají vrhání paprsků pro finální vyrenderování scény či postav. Tyto jsou následně skombinovány s filmovým materiálem, a vznikají tak prostředí, která nikdy neexistovala nebo by bylo příliš nákladné vytvořit obdobné kulisy.



Obr. 2-8: Postava Gluma z filmu Pán Prstenů byla kompletně vytvořena na počítači
zdroj: [4]

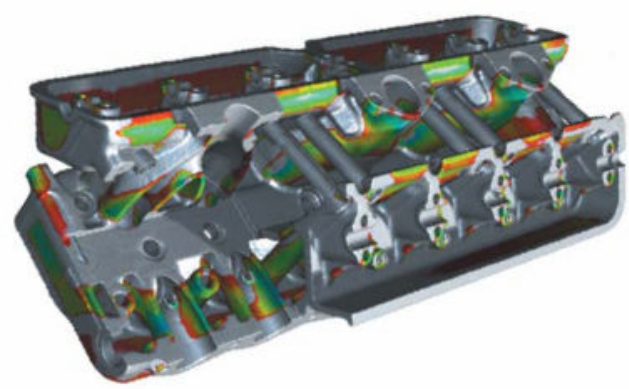
Obr. 2-9: Ray tracovaný obrázek vily
zdroj: [3]

Velmi významné odvětví, kde se využije možností ray tracingu je vizualizace medicínských dat. Jde o velmi objemná a komplexní data, která jsou získávána pomocí scannerů, roentgenů či jiných přístrojů schopných snímat povrch, či vnitřní strukturu lidského těla. Pokud si lékaři mohou prohlédnout reálnou stavbu lidského těla, umožní jim to snadnější studium. Ovšem hlavní přínos je možnost zobrazit detail orgánu určitého pacienta ještě před zákrokem a tudíž operaci důkladněji naplánovat a vše připravit. Také je možné otestovat případné implantáty ještě předtím, než budou voperovány tak, aby byly poté tělem co nejlépe přijaty.

Obr. 2-10: Zobrazení pacientových ledvin

Zdroj: [9]

Podobně jako v medicíně se využívá ray tracingu k vizualizaci nasnímaných dat lidského těla, tak podobně je využíván i ve strojírenství při navrhování nových výrobků, či analyzování výrobků existujících. Je takto možno například zobrazit tloušťku stěn a kontrolovat tak kvalitu výrobního procesu pomocí vizualizace chyb a nepřesností.



Obr. 2-11: Vizualizace tloušťky materiálu bloku motoru

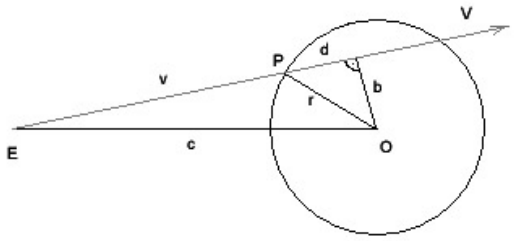
Zdroj: [9]

Množství dat a jejich složitost se neustále zvyšuje a proto dnes pro vykreslování nestačí obyčejné pracovní stanice, ale využívá se serverových farem a není výjimkou, když jeden snímek počítá 1024 procesorů a množství dat dosahuje několika 10GB.

2.4 Výpočet průsečíků

Základem ray tracingu je protínání paprsků s objekty ve scéně. Základním objektem je koule. Také výpočet průsečíků s ní je jeden z nejjednodušších, proto zde bude vysvětlen.

2.4.1 Průsečík paprsku s koulí



Obr. 2-12: Průsečík paprsku a koule

Obrázek ukazuje paprsek **R** (s počátkem **E** a směrem **V**) protínající kouli se středem **O** a poloměrem **r**. Dle obrázku:

$\mathbf{v}^2 + \mathbf{b}^2 = \mathbf{c}^2$ a $\mathbf{d}^2 + \mathbf{b}^2 = \mathbf{r}^2$

tedy

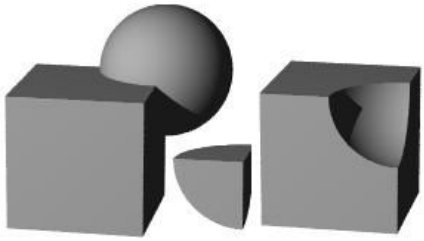
$\mathbf{d} = \text{sqrt}(\mathbf{r}^2 - (\mathbf{c}^2 - \mathbf{v}^2))$

Pro určení, zda došlo k protnutí koule paprskem nám stačí vypočítat **d**. Pokud je **d**>=0, potom došlo k protnutí. Pokud je **d** menší než 0, průsečík nebyl nalezen. Poté vzdálenost od počátečního bodu **E** je **v** – **d**, neboli vektorově:

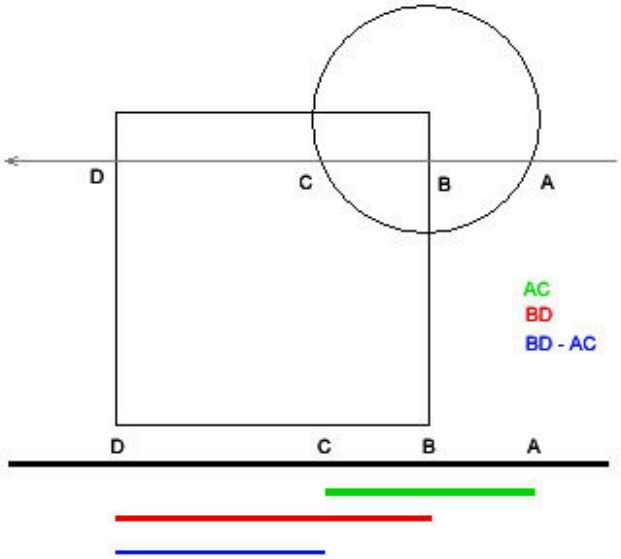
$\mathbf{P} = \mathbf{E} + (\mathbf{v} - \mathbf{d}) * \mathbf{V}$

Výpočet průsečíku s jinými geometrickými útvary se provádí podobně. Většina výpočtů je mnohem složitější.

2.4.2 Průsečík s CSG tělesem



Obr. 2-13: Konstrukce pomocí CSG



Obr. 2-14: Průsečíky s paprskem

V CSG (Constructive Solid Geometry) se používá matematických operací a jednoduchých těles ke konstrukci těles složitějších. Viz Obr. 2-13, pomocí odečtení koule od krychle dostaneme nové těleso. Zde je již nové těleso vystínováno a osvětleno. K tomu, abychom to mohli provést je nutno najít správný průsečík.

Nejprve nalezneme průsečíky s koulí, jsou to body **A** a **C**, zobrazeny zeleně. Poté hledáme průsečíky s krychlí – body **B** a **D**. Jelikož je výsledné těleso určeno rozdílem, dostaneme rovnici:

X = BD – AC

X = CD

Nyní určíme, který z těchto svou průsečíků je bližší, a tím získáme průsečík paprsku se složitým tělesem.

Výpočet CSG operací je možné dělat jednak nalezením všech průsečíku jednoho tělesa a poté druhého. Následně tyto průsečíky seřadit dle vzdálenosti a dle požadované operace je setřídit, až nám zůstanou pouze ty, které jsou výsledkem.

Nebo druhou možností je nalézt nejbližší průsečík obou těles a poté dle zvolené operace mírně posunout počátek paprsku a vypočítat další průsečík. Tuto metodu je možné najít implementovánu ve většině ray tracerů, které podporují CSG grafiku.

Sjednocení

Protože nám jde o nalezení nejbližšího průsečíku, tak nám stačí nalézt průsečík s jedním i druhým tělesem, porovnat jejich vzdálenosti a vrátit jako nejbližší ten, který má kratší vzdálenost od počátku paprsku.

Průnik

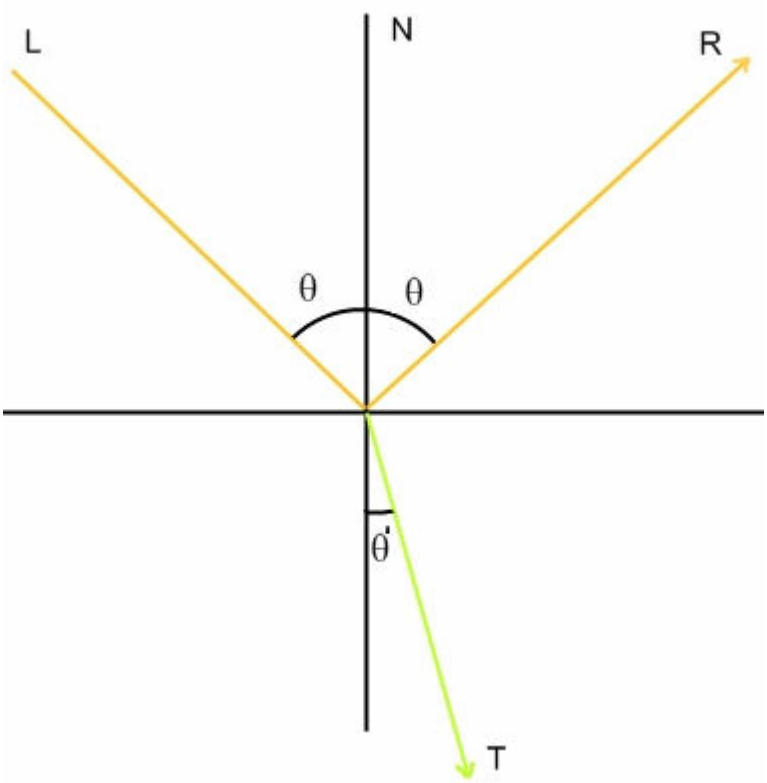
Najdeme nejbližší průsečík v obou tělesech. Pokud jsou nalezeny průsečíky u obou těles, tak je možnost, že se tělesa protínají. Vzdálenosti průsečíků porovnáme a jako počátek nového průsečíku vezmeme vzdálenější bod. Vytvoříme nový paprsek a opět hledáme průsečíky. Pokud jsou opět nalezeny dva průsečíky porovnáme je a nyní vezmeme jako konec průniku bod bližší.

Rozdíl

Rozdíl je obdobný. Najdeme první průsečík s tělesem A. To je to od kterého odčítáme těleso B. Najdeme si další průsečík s tělesem A. Nyní hledáme průsečík s tělesem B. Pokud je tento průsečík blíže, než druhý průsečík A, tak vrátíme jako výsledný průsečík ten s tělesem B.

2.5 Odrazy a lomy

Při vyslání primárního paprsku do scény dochází k jeho interakci s objekty a v bodě průsečíku mohou vzniknout sekundární paprsky. Tyto paprsky jsou paprsky odražené nebo lomené, dle vlastností tělesa.



Obr. 2-15: Odražené a lomené paprsky a úhly

2.5.1 Odraz

Úhel odrazu se rovná úhlu dopadu. Známé pravidlo z fyziky. V ray tracingu se počítá s paprsky, tudíž s vektory a je potřeba tento vztah vyjádřit matematicky:

R = 2 * (N.L) * N – L

R – odražený paprsek
L – dopadající paprsek
N – normálový vektor

2.5.2 Lom

Při přechodu z jednoho prostředí do druhého se paprsek láme. Úhel lomu závisí na hustotě prostředí, že kterého paprsek přichází a prostředí, do kterého paprsek vstupuje. Úhel lomu je dán Smelkovým zákonem:

$$\eta_1 \sin \theta = \eta_2 \sin \theta'$$

neboli:

$$\sin \theta' = (\eta_1 / \eta_2) * \sin \theta$$

Postupnými úpravami a převedením do vektorů dostaneme tento vzorec:

$$n = \eta 1 / \eta 2$$

$$\cos \theta = -N.L$$

$$\cos 2\ \theta' = 1 - n^2 * (1 - \cos^2 \theta)$$

$$T = n * L + n * \cos \theta * \mathrm{sqrt}(\cos^2 \theta') * N$$

n – poměr indexů lomu

η1 – index lomu prvního prostředí, různé materiály mají různý index lomu.

Vzduch	– 1
Voda(20°C)	– 1,33
Sklo	– 1,5

η2 – index lomu druhého prostředí

N – normálový vektor

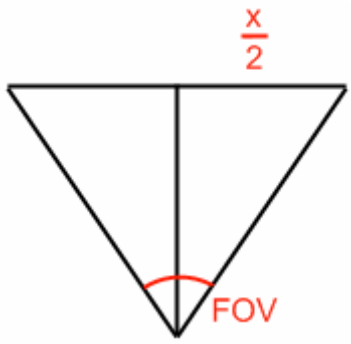
L – dopadající paprsek

T – výsledný lomený paprsek

2.6 Generování paprsků

Abychom mohli určit směr primárních paprsků, musíme znát několik parametrů kamery. Jednak je to zorný úhel (FOV – field of view), dále vzdálenost promítací roviny a také rozlišení výsledného obrazu. Zorný úhel vyjadřuje šíři obrazu, kterou je možno pozorovat v jeden okamžik. U každého živého tvora je to různé. U lidí je to přibližně 180°, ale protože člověk sleduje obraz oběma očima současně je reálný úhel 140° protože část obrazu se překrývá a zbylých 40° vidí člověk pouze jedním okem.

Pokud známe vzdálenost promítací roviny a zorný úhel, můžeme vypočítat rozměry, které budeme následně potřebovat při výpočtu směru primárního paprsku.



Obr. 2-16: Zorný úhel

xsize = tan((FOV / 360) * π) * nearplane

obdobně pro výšku **ysize**

Potom primární paprsek na pozici x,y, která je zadána relativně v rozmezí 0-1 z důvodu nezávislosti na rozlišení, bude generován následovně:

xpos = (x-0.5) * xsize
ypos = (y-0.5) * ysize
zpos = nearplane

Počátek paprsku bude na pozici kamery a směr bude z tohoto počátku do bodu určeného souřadnicemi z výpočtu výše.

3 Návrh tříd

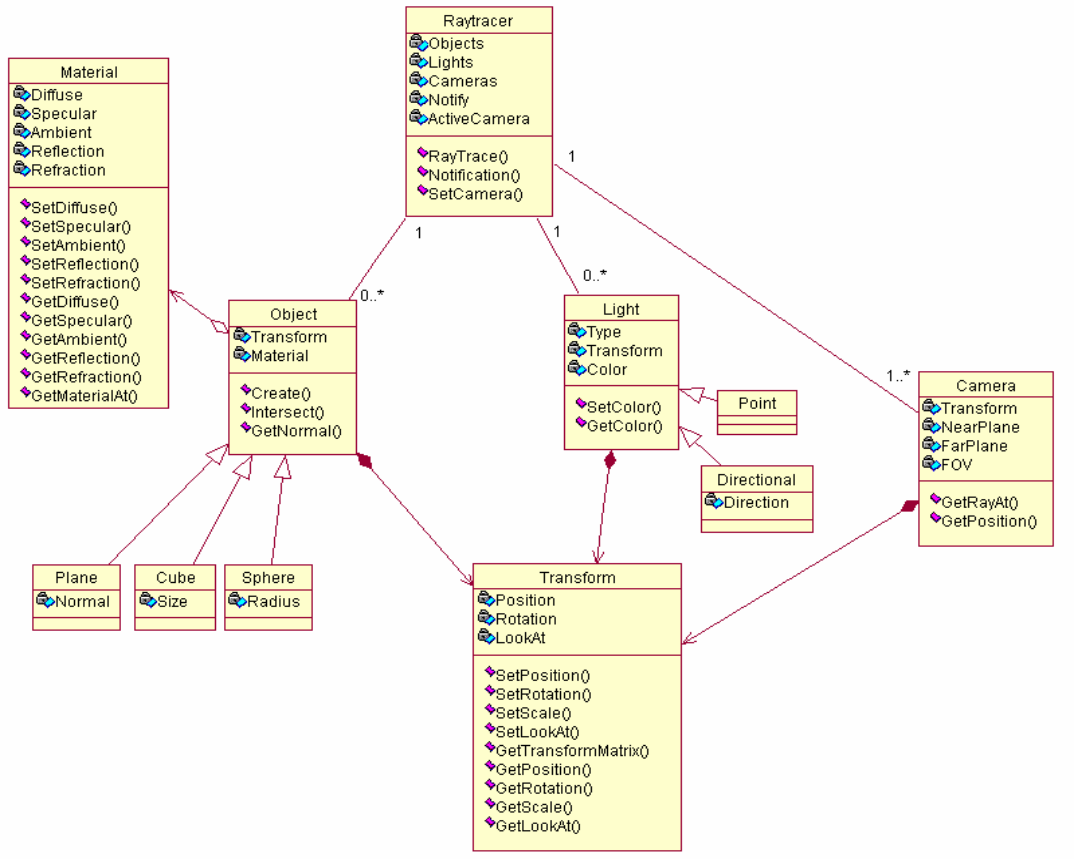
V rámci diplomové práce byl navržen minimalistický objektově orientovaný ray tracer. Tento ray tracer by měl umět vykreslit scénu složenou z jednoduchých objektů – koule, krychle, rovina a dále potom složitější objekty složené z předešlých pomocí CSG. Dále by měl zvládnout tyto objekty umístit kdekoliv ve scéně a případně je pootočit. Další nedílnou součástí jsou materiály. Tyto materiály popisují chování světla po dopadu na povrch tělesa a jeho optické vlastnosti. Vzhledem k tomu, že výsledná barva je vždy počítána v bodě dopadu a nemusí být implicitně pro celý objekt konstantní, je možné vytvářet i komplexnější materiály typu dřevo, mramor apod. Všechny objekty a světla jsou umísťovány do scény. Takže je možné vytvořit několik scén a pak tu která má být vykreslena přiřadit ray traceru. Součástí scény je i kamera, která na základě vlastností „čočky“ bude produkovat primární paprsky.

V rámci návrhu bylo také nutno vytvořit několik matematických tříd a to pro práci s vektory, maticemi a paprsky. Bez těchto tříd by nebylo možné provádět matematické operace, popisovat pozice objektů, jejich transformace apod.

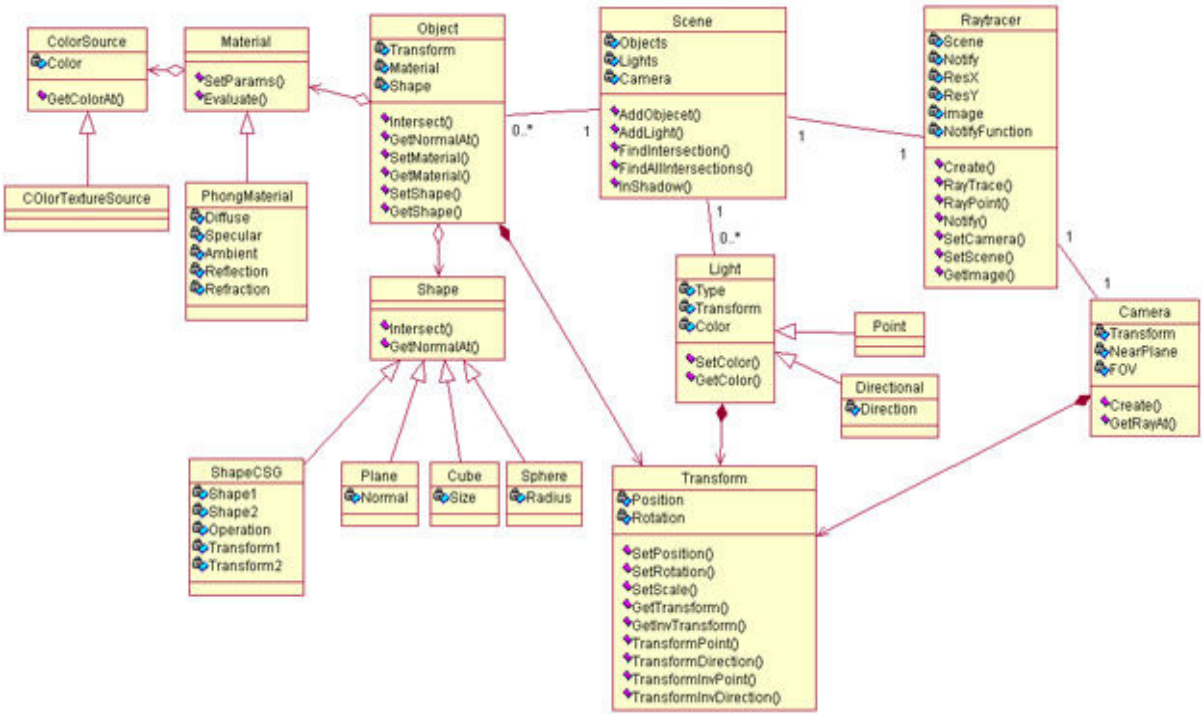
Při návrhu modelu jsem vycházel z mých předešlých zkušeností s programováním 3D grafického enginu a 3D grafického rozhraní Direct3D. Ačkoliv je určeno pro real-time grafiku, je toto rozhraní objektově orientované. Dále jsem studoval některé implementace raytracingu, které jsou volně dostupné na internetu. Jejich komplexnost byla velmi rozdílná. Od primitivních, kde veškeré vykreslování bylo prováděno v několika málo funkcích, až po velmi komplexní POV-Ray, jeden z nejznámějších a nejpoužívanějších open source raytracerů, využívající mnoho tříd a složitý objektový model.

Mnou navrhnutý model jsem v průběhu času rozšiřoval a upravoval, dle nových poznatků získaných studiem textových materiálů, psaním práce, nebo implementací ukázkové aplikace. Pro srovnání uvedu model, který byl prezentován jako součást semestrální práce a pak také model finální, ke kterému jsem dospěl postupnými úpravami. Následně popíšu, proč jsem dané úpravy provedl a jaké z toho plynou výhody.

Hned na první pohled je vidět zvýšení komplexnosti celého modelu. Ovšem tato komplexnost má za následek velmi snadné použití při implementaci a hlavně velmi dobrou rozšiřitelnost. Celý model je připraven tak, aby mohla být libovolná třída zděděna a upravena, dle požadavků a přitom zůstala funkčnost zachována.



Obr. 3-1: Původní diagram tříd, součást semestrální práce



Obr. 3-2: Výsledný diagram tříd

V původním návrhu byl záměr, aby byly objekty, světla i kamery součástí třídy Raytracer. Tato třída měla projít všechny body výsledného obrazu, počítat průsečíky, a vyhodnotit na základě materiálu objektu jeho výslednou barvu.

Jak je z návrhu patrné, původní model počítal pouze s materiálem, který by odpovídal Phongově osvětlovacímu modelu. Tedy materiálu obsahujícímu ambientní, difuzní a spekulární složku. Ovšem z důvodu rozšiřitelnosti a možnosti vytvoření komplexnějších materiálů a jejich popisu bylo nutno tuto třídu nahradit třídou univerzální, od které budou poté specializované třídy zděděny. Také zde došlo k přesunutí výpočtu osvětlení a výsledné barvy pixelu z Raytraceru do Materialu. A to opět z důvodu možného rozšíření.

V souvislosti s vytvořením univerzálního materiálu bylo nutné tomuto materiálu předat informace o prostředí ve kterém se nachází, o okolních objektech a světlech. Toto vedlo k vytvoření třídy Scene, která tyto informace obsahuje. Zde byly vytvořeny tři metody, které se starají o výpočty průsečíků a zastínění objektů.

Třída objekt, která původně představovala přímo určitý tvar – kouli, krychli, rovinu. Byla nahrazena třídou, jenž reprezentuje určitý objekt ve scéně, ale tvar tohoto objektu je určen novou třídou Shape. Pozice objektu je určena transformací a jeho barevné vlastnosti materiálem.

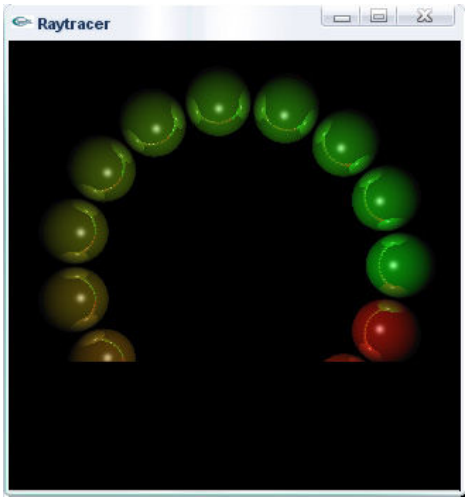
Shape, neboli tvar objektu byl do modelu přidán kvůli podpoře CSG grafiky a modelů. A také kvůli nezávislosti objektu a jeho tvaru. Další neméně podstatnou výhodou tohoto je i veliká paměťová náročnost komplexních modelů. Nyní je možné vytvořit jedinou instanci třídy odvozené od Shape, jenž bude načítat složitý polygonální model o mnoha tisíci trojúhelnících a přiřadit ji k třeba sto objektům různě rozmístěným po scéně. Ovšem takto načtený model bude v paměti uložen pouze jednou a tudíž dojde k obrovské úspoře paměťového prostoru. Obdobně je to možné provést i s materiály. Je možné vytvořit jednu instanci materiálu a tu přiřadit několika objektům.

Jako nezměněné z původního modelu zůstaly pouze dvě třídy a to třída Camera a Light.

3.1 Raytracer

Základní třída celého ray traceru. Po vytvoření je instanci této třídy přiřazen ukazatel na scénu, kterou bude ray tracer vykreslovat. Scéna obsahuje informace o jednotlivých objektech, světlech a kameře. Hlavní metoda RayTrace projde v cyklu průchodu přes všechny body promítací roviny a každým bodem vrhne primární paprsek směrem do scény. Tímto paprskem se testují průsečíky s jednotlivými objekty ve scéně. Poté na základně vlastností materiálu daného objektu se můžou vrhat sekundární paprsky, pokud v bodě průsečíku vzniknou, a nakonec je vrácena výsledná barva pixelu v daném bodě.

Důležitou funkcí je ‚Notify'. Ta umožňuje nastavit interval, ve kterém se zavolá funkce NotifyFunction, jež vrátí dosud vykreslený obraz. Toto je možno nastavit na pixel, řádek nebo celý obraz. Využitelnost takové funkce přichází v úvahu třeba v případě, že vykreslování trvá příliš dlouho a my chceme zobrazit i částečný obraz.



Obr. 3-3: Ukázka z aplikace – vykreslování po řádcích

3.2 Transform

Transformace slouží pro změnu pozice, otočení či měřítka objektu. Tyto transformace jsou prováděny pomocí matic a jejich postupným násobením. Při výpočtu transformačni matice záleží na pořadí, v jakém jednotlivé transformace skládáme. Pokud objekt nejprve otočíme a poté posuneme, tak bude otočený na pozici, kam jsme ho posunuli. Ovšem, pokud provedeme operaci opačně, tak bude objekt otočený okolo počátku ve vzdálenosti, kam jsme ho posunuli.

Pro transformace ve 3D prostoru je potřeba použít matici 4x4. Ukázka matic:

$$\mathbf{M_s} = \begin{matrix} \mathbf{S_x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{S_y} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S_z} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{matrix} \quad \mathbf{M_t} = \begin{matrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{T_x} & \mathbf{T_y} & \mathbf{T_z} & \mathbf{1} \end{matrix} \quad \mathbf{M_{rx}} = \begin{matrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{\cos(R_x)} & \mathbf{\sin(R_x)} & \mathbf{0} & \mathbf{0} \\ \mathbf{-\sin(R_x)} & \mathbf{\cos(R_x)} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{matrix}$$

$\mathbf{M_s}$ – matice pro změnu měřítka v jednotlivých osách

$\mathbf{M_t}$ – matice pro posunutí

$\mathbf{M_{rx}}$ – matice pro otočení okolo osy X o úhel R_x

Ve zde popisovaném modelu je prováděn výpočet výsledné transformační matice v pořadí:

$$\mathbf{T = M_s * M_{rx} * M_{ry} * M_{rz} * M_t}$$

Tímto je získána transformační matice Dále je potřeba vypočítat inverzní transformační matici.

Nyní těchto matic využijeme pro transformaci bodu a směrového vektoru. Je potřeba rozlišovat, jaký typ vektoru transformujeme. Při transformaci směrového vektoru se při výpočtu nevyužívá poslední řádek(posun) transformační matice, kdežto při transformaci bodu v prostoru ano.

Využití je zejména pokud je potřeba transformovat paprsek z prostoru scény do prostoru objektu, aby bylo možné nalézt jeho průsečík s objektem. Tato transformace se provádí pomocí inverzní matice. Po vypočtení průsečíku je tento pomocí normální matice transformace převeden zpět do prostoru scény.

3.3 Material

Touto třídou jsou popisovány optické vlastnosti tělesa. Jeho barva, propustnost pro světlo, odrazivost apod. Při návrhu této třídy, jejích metod a jejího zařazení do modelu bylo prozkoumáno několik variant raytracerů a jejich objektového návrhu. Většina raytracerů předpokládá standardně využití phongova osvětlovacího modelu a základních barevných složek(difuzní, spekulární, ambientní). Materiál v takovémto návrhu plní pouze funkci parametrů, ale výslednou barvu vyhodnocuje ray tracer. Kdežto kvůli dosažení co největší univerzálnosti a možnosti rozšíření je třída Material ta, která vyhodnotí hodnotu materiálu v daném bodě a vrátí jeho výslednou barvu.

Po nalezení průsečíku ray tracerem je zavolána metoda Evaluate, která vyhodnotí všechny parametry materiálu v daném bodě s ohledem na celou scénu a pozici průsečíku. Takto je možno jednoduše díky dědičnosti vytvořit novou třídu, která dokáže popsat přesně materiál a jeho vlastnosti dle požadavků. Tyto vlastnosti totiž nemusí mít pouze diskrétní hodnoty(po celém povrchu stejnou barvu), ale mohou nabývat hodnot spojitých(generovaných funkcí). Také není problém nahradit phongův osvětlovací model modelem jiným, který pro dané využití bude poskytovat lepší výsledky.

Základní třída Material vrací pouze bílou barvu. Je vhodná například na ověření pozice objektu, před použitím složitějšího materiálu.

Třída PhongMaterial využívá k vyhodnocení barvy objektu phongův osvětlovací model. Počítá s difuzní, spekulární a ambientní složkou. Dále pokud je nastavena u materiálu průhlednost, vyšle sekundární paprsky skrz objekt a vypočte lomený obraz. Podobně, pokud má materiál lesklý povrch, tak jsou vyslány odražené paprsky a výsledná barva je spočítána kombinací všech těchto složek dohromady.

Jednotlivé barevné složky, se kterými je počítáno jsou ambientní – toto je složka, která reprezentuje okolní světlo scény, které není vyzařováno žádným konkrétním světleným zdrojem. Například jiná barva prostředí je při západu slunce a jiná v pravé poledne. Difúzní složka je typická barva objektu. Spekulární barva vyjadřuje odlesk na tělese, odrazivost určuje, jaké množství světla se odrazí od povrchu tělesa, a lámavost, kolik světla tělesem projde. Toto umožňuje definovat vlastnosti každého objektu, zda se v něm světlo láme, zda je lesklý či matný atd.

Výpočet výsledné barvy z jednotlivých složek:

$$\mathbf{R = 2 * (N.L) * N - L}$$

$$\mathbf{I = Ambient + Diffuse * N.L + (R.V)^n}$$

N – normálový vektor, **L** – vektor ke světlu, **V** – vektor ke kameře, **R** – vektor odraženého paprsku

Díky dědičnosti je možné odvodit složitější materiály, které budou popisovat strukturu například mramoru a nebo dřevo díky procedurálním texturám. Tato třída umožní vrátit barvu kdekoliv v tělese. Toto je velmi dobře použitelné v CSG grafice, kde může dojít k „vyříznutí“ jednoho objektu ze druhého, potom i v tomto výřezu si objekt stále zachovává svojí strukturu.

SetParams – tato metoda slouží k základnímu nastavení parametrů daného materiálu. Při dědění může dojít k úpravám této třídy a ke změně počtu a typů parametrů

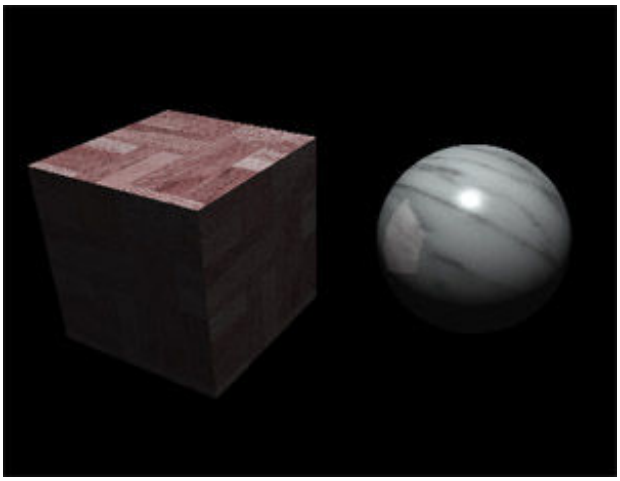
Evaluace – nejdůležitější funkce, která dle vlastností materiálu vrátí výslednou barvu

3.4 ColorSource

Tato třída je zdrojem barvy. Tato třída vznikla na základě potřeby začlenit do modelu textury. Textura(základní difuzní) je bitmapa popisující barvu v daném bodě. Instance této třídy je použita v PhongMaterial pro určení barvy objektu. Při vytváření takovéhoto materiálu jsou zadávány jeho parametry a to jednotlivé barvy. Difuzní barva je typu ColorSource. Pokud chceme, aby celý objekt měl jednolitou barvu, použijeme rodičovskou třídu ColorSource, která pouze předá barvu.

Pokud ovšem chceme využít možností, které tímto získáme, tak použijeme ColorSourceTexture, této třídě je při vytváření předán název souboru, ze kterého má načíst obrazek, jenž bude použit jako textura. Dále je potřeba nastavit způsob mapování textury na výsledný objekt. Můžeme si zvolit mezi mapováním na krychli, na kouli, nebo na rovinu. Metoda GetColorAt nejprve převede souřadnice bodu do prostoru objektu, poté vypočítá dle způsobu mapování, pozici v textuře a vrátí výslednou barvu v daném bodě.

Možnosti využití této třídy nekončí pouze u barevných textur, ale dají se u složitějších materiálu použít také normálové mapy. Jako barevný zdroj se použije normálová mapa načítaná ze souboru a poté při výpočtu normálového vektoru může být tento upraven dle souřadnic získaných z normálové mapy.



Obr. 3-4: Ukázka mapování textur na kouli a krychli

3.5 Scene

Třída Scene sdružuje objekty, světla a kameru v prostoru. Je to velmi důležitá třída, protože poskytuje metody pro výpočet průsečíků a také kontrolu, zda je objekt zastíněn. Tato třída byla navržena, protože umožňuje logické oddělení ray traceru od dat, která vykresluje. Proto je například možné použít různé scény v jedné aplikaci a pouze se mezi nimi přepínat. Například pro porovnání nějakých detailů. Součástí scény je i kamera. Do scény je možní přidat libovolný počet světel a objektů.

Mimo metod pro přidání objektu a světla poskytuje třída dvě metody na výpočet průsečíků. První metoda vrátí nejbližší průsečík paprsku s objekty. Druhá metoda vrátí všechny nalezené průsečíky(omezené maximem), seřazené od nejbližšího po nejvzdálenější. Množství těchto průsečíků je omezeno maximálním počtem, protože se může stát, že těchto průsečíku je nekonečně mnoho. A to například pokud paprsek rovnoběžně protíná rovinu. Další metodou je zjištění, zda je průsečík zastíněn, či zda je osvětlen. Výpočet je podobný, jako v případě zjišťování průsečíku, ovšem s tím rozdílem, že není potřeba hledat všechny průsečíky, či nejbližší, ale stačí nám pouze jeden jediný k tomu, aby jsme určili, že je objekt zastíněn.

3.6 Camera

Úkolem této třídy je vrhat primární paprsky směrem do scény a vracet jejich pozici a směr v daném bodě. Z důvodu snadné rozšiřitelnosti ray traceru a možnosti nastavit různé módy kamery, jako například rybí oko nebo ortogonální kamera, byla tato třída navržena tak, že při vytváření je jí zadána základní transformace – pozice, otočení. Dále pak parametry jako nearplane rovina, na kterou se bude promítat. Dále je zadán FOV, neboli Field Of View – zorný úhel kamery.

GetRayAt(x,y) – parametry x a y určují relativní souřadnici na nearplane v rozmezí 0–1. Z důvodu možných implementací rozdílných typů kamer a možného nastavení libovolného rozlišení je pozice zadávána relativně. Výstupem této funkce je počáteční bod a směr paprsku z kamery na dané pozici.

3.7 Light

Tato třída reprezentuje světlo ve scéně. Světlo je určeno svým typem. Tyto typy mohou být dva základní. Směrové světlo, kde všechny paprsky míří jedním směrem, v přírodě zhruba odpovídá Slunci. Paprsky se nám jeví, jako by byly všechny rovnoběžné. Další druh je světlo bodové. Toto světlo vrhá paprsky ze své pozice všemi směry. Typicky například žárovka.

Světlo má svoji barvu. Dále má také pozici a může využívat funkcí třídy Transform.

3.8 Object

Třída object prošla při práci na modelu tříd asi největšími změnami. V původním modelu tato třída obsahovala geometrii tělesa, která byla přesunuta do třídy Shape. A to hlavně z důvodu snadné rozšiřitelnosti a nezávislosti, takže objekt může mít tvar koule stejně jako složitého modelu z mnoha tisíc trojúhelníků. Také výpočet průsečíků byl přesunut do třídy Shape. Další součástí je Material. Tento byl pouze popisem barvy objektu. Ovšem nyní je to komplexní třída, počítající dle pozice průsečíku a parametrů scény výslednou barvu v daném bodě. Využitím těchto dvou tříd je možné oddělit popis materiálu od tvaru objektu a obě dvě součásti mohou být libovolně složité. Poslední součástí je transformace umožňující umístit objekt libovolně v prostoru, či ho libovolně otočit.

Objekt je základní geometrický útvar ve scéně. Jeho základní funkcí je vrácení nejbližšího průsečíku geometrie objektu a paprsku. Toto záleží vždy na implementaci daného objektu, jinak se bude počítat průsečík s rovinou a jinak například s koulí. Dále každý objekt využívá transformaci, takže je s ním možno libovolně v prostoru pohybovat a otáčet. Dále je možno nastavit vlastnosti materiálu. Jeho barvu, průhlednost, odrazivost atd.

Create() – vytvoří objekt, podle typu objektu jsou vyžadovány parametry. Např. u koule poloměr, u roviny normála, u krychle velikost hrany.

Intersect(ray) – vrátí třísložkový vektor určující nejbližší průsečík s paprskem ray

GetNormalAt(x,y,z) – vrátí třísložkový vektor určující normálu v daném bodě. Nejčastěji to bývá v bodě průsečíku s paprskem. Tato normála je následně používána při výpočtu osvětlení.

3.9 Shape

Tvar objektu byl původně součástí třídy Object. Ovšem postupnými úpravami modelu z něj vznikla samostatná třída. A to zejména proto, aby tvar objektu byl nezávislý na objektu, ke kterému náleží. Další výhodou je úspora paměti při použití komplexních modelů. Tyto modely mohou být načteny do paměti pouze jednou a poté k objektům přiřazeny pouze pomocí ukazatele na tento tvar. Třída Shape je sama o sobě virtuální, má pouze dvě metody a to Intersect a GetNormalAt. Kde první vrací nejbližší průsečík a druhá vrací normálu v bodě průsečíku. Obě dvě tyto funkce jsou implementovány ve zděděných třídách podle tvaru daného tělesa.

V základu jsou implementovány tři geometrické objekty a to koule, krychle a rovina. Pomocí těchto tvarů se dají udělat zajímavě vypadající scény.

Dále byla implementována třída ShapeCSG, jenž dokáže za pomoci operací průniku, rozdílu a sloučení vytvořit složité geometrické tvary z jednoduchých.

3.9.1 ShapeCSG

Tato třída byla do návrhu přidána kvůli podpoře CSG grafiky. Třída umožňuje nastavit dvě geometrie, mezi kterými se provede požadovaná operace. Vzhledem k tomu, že obě dvě jsou odvozeny od rodičovské třídy Shape, tak touto geometrií může být primitivní tvar typu koule, nebo také složité těleso, které vzniklo několika CSG operacemi.Těmito operacemi může být:

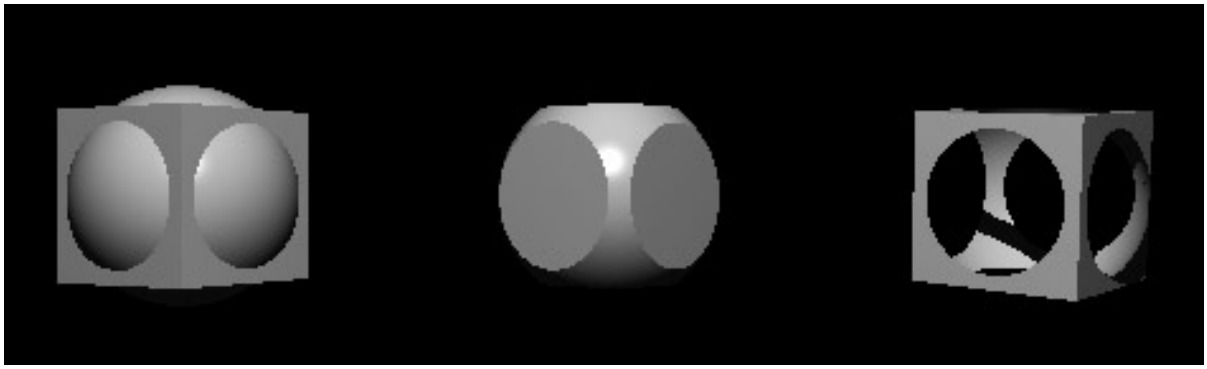
Sloučení – ze dvou těles vznikne nové, které obsahuje geometrii obou původních
Rozdíl – od geometrie prvního je odečtena geometrie druhého objektu
Průnik – geometrie, která je společná oběma objektům

Po provedení požadované operace je výsledkem nový tvar a tudíž i nové průsečíky. Ve výsledku poté Object přiřadí celému nově vzniklému objektu transformaci a tím ho někam umístí a také materiál, který bude pro celou novou geometrii shodný.

Create(shape1,shape2,op) – Vytvoří skupinu objektů. Jako parametry jsou předány dvě skupiny objektů a booleovská operace, která se s nimi má provést. Pokud nebude předána druhá skupina objektů, pracuje se s touto třídou podobně jako s obyčejným objektem. Využít je to možné, pokud potřebujeme provést operaci se třemi objekty.

Intersect(ray) – vrátí třísložkový vektor určující průsečík skupiny s paprskem ray. Výpočet tohoto průsečíku viz kapitola 2.4.2.

GetNormalAt(x,y,z) – vrátí třísložkový vektor určující normálu v daném bodě. Nejčastěji to bývá v bodě průsečíku s paprskem. Tato normála je následně používána při výpočtu osvětlení. Výpočet tohoto průsečíku viz kapitola 2.4.2.



Obr. 3-5: CSG operace

4 Rozšíření

4.1 Textury

Detailní popis struktury povrchu objektu pomocí geometrie by byl velice náročný na množství dat a výpočet průsečíku by trval velmi dlouho. Proto se pro popis vlastností povrch(a nejen povrhu) objektu zavádí textury. V praxi dělíme textury na dva typy:

Obrazové textury

Jedná se většinou o bitmapu obsahující čtyři barevné kanály – červenou, modrou, zelenou a alfa kanál. Záleží na rozlišení obrázku a jeho detailech.

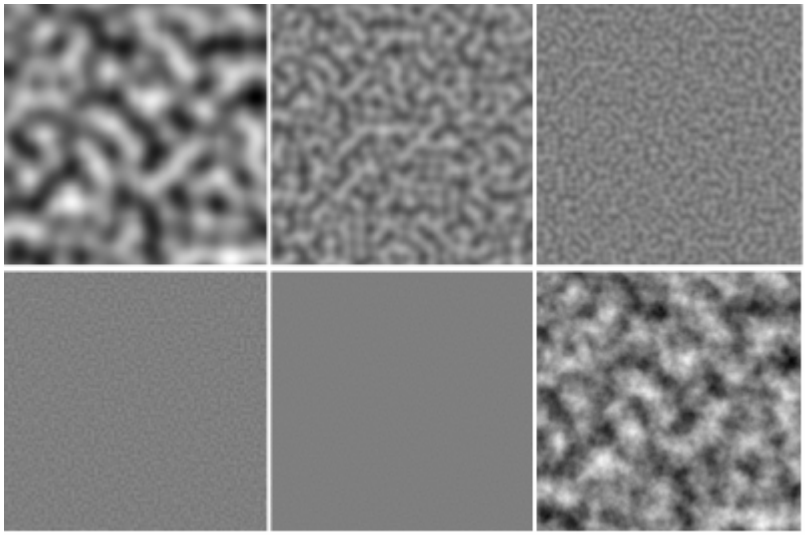
Speciální typy textur

- **Difuzní textura** - Kanály RGBA - obsahují základní obrazovou informaci při rovnoměrném nasvícení povrchu. Kanál A bývá používán jako alfa kanál určující hodnotu průhlednosti. Obvykle nižší hodnota znamená vyšší průhlednost.
- **Normálová textura** - Kanály RGB značí XYZ hodnotu normálového vektoru v tangent space.
- **Spekulární textura** - Kanály RGB označují barvu a intenzitu odlesku pixelu po nasvícení.

Procedurální textury

Na rozdíl od předchozí varianty je tento typ textur generován určitou funkcí. Proto nezáleží na rozlišení, barvu je možné vypočítat v libovolném bodě. Další výhodou je prostorová nenáročnost, protože nemusí být v paměti uchovávána velká obrazová data. Nevýhodou je ovšem delší doba výpočtu.

Většina funkcí při generování textury vychází z tzv. Perlinova šumu. Tuto funkci navrhl v roce 1983 Ken Perlin a v roce 1985 ji veřejně prezentoval na konferenci SIGGRAPH. Výsledky této funkce mají na první pohled náhodný vzhled, ovšem na základě vstupních parametrů může být vzhled upravován. Při generování výsledné textury dojde většinou ke složení několika vzorků o různých amplitudách a frekvencích.



Obr. 4-1: generování Perlinova šumu

Mapování textur

Mapováním textury rozumíme nanášení textury na objekt. Na každý objekt musí být textura nanášena jinak. Proto rozlišujeme základní typy mapování dle geometrie objektu – koule, rovina, kvádr, válec.

Mapování textury na kouli

Pokud známe bod na povrchu koule, můžeme vypočítat jakému bodu v textuře odpovídá.

U souřadnici vypočítáme z X souřadnice normálového vektoru z rozmezí -1 … 1

U = asin(N.x) / π + 0.5

V souřadnici vypočítáme z Y souřadnice normálového vektoru z rozmezí -1 … 1

V = asin(N.y) / π + 0.5

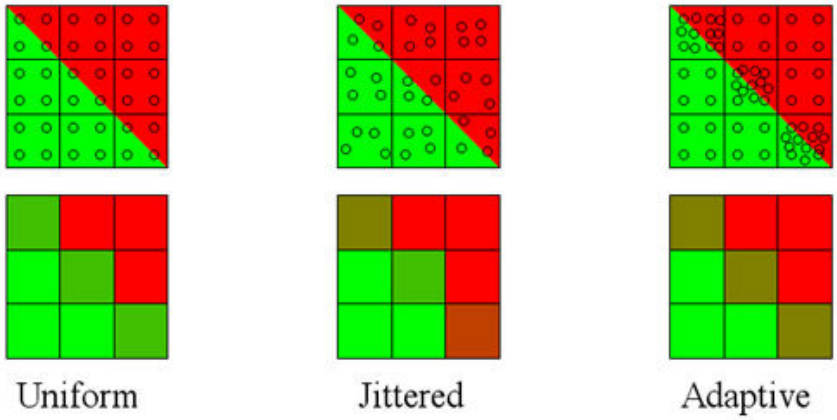
Tímto dostaneme u,v souřadnice v textuře, která bude namapována na bod na kouli.

4.2 Anti-aliasing

„**Aliasing** je jev, ke kterému může docházet v situacích, kdy se spojitá informace převádí na diskrétní (nespojitou). Takový převod se nazývá vzorkování, a aby nedocházelo k aliasingu, musí být vzorkovací frekvence rovna minimálně dvojnásobku nejvyšší frekvence obsažené ve vzorkovaném signálu - tzv. Nyquistův teorém. Pokud tuto podmínku nesplňuje, dochází k překrytí frekvenčních spekter vzorkovaného signálu a tedy ke ztrátě informace.“ [1]

Aby ke vzniku takových artefaktů nedocházelo využívá se různých metod anti-aliasingu. Všechny metody vychází z toho, že je potřeba zvýšit počet paprsků vrhaných do scény pro výpočet barvy jednoho pixelu. Většinou tento počet bývá čtyři, ovšem například u adaptivního supersamplingu je počet paprsků vyšší. Z toho plyne i mnohem vyšší časová náročnost výpočtu scény, protože počet vržených paprsků se násobí.

Supersampling je metoda, při níž dojde ke zvýšení počtu paprsků které využijeme pro výpočet barvy jednoho pixelu. Výsledné barvy každého z nich zprůměrňujeme a tím dostaneme výslednou barvu pixelu.



Obr. 4-2: Typy supersamplingu

[1] cs.wikipedia.org, *Aliasing* [Online] [20.4.2008], Dostupný z WWW: http://cs.wikipedia.org/wiki/Aliasing

4.2.1 Uniform supersampling

Každým bodem výsledného obrazu je vržen stejný počet paprsků. Výsledný pixel je pak vypočítán jako průměr barev vzniklých z těchto paprsků. V našem případě jsou použity paprsky čtyři a to téměř v „rozích“ výsledného pixelu.

Obr. 4-3: Uniform supersampling

4.2.2 Jittered supersampling

Tato metoda je velmi podobná metodě předchozí, opět jsou vrhány čtyři paprsky, které jsou následně zprůměrňovány, ovšem jejich pozice není pravidelná, ale jsou náhodně rozmístěny po celém pixelu. Ačkoliv může výsledek vypadat hůře, než v předchozím případě, tak tato metoda má využití zejména při odstraňování aliasingu na rozsáhlé ploše, která má pravidelnou texturu, jako třeba šachovnice.

Obr. 4-4: Jittered supersampling

4.2.3 Adaptive supersampling

Adaptivní supersampling podává nejlepší výsledky. Není použito konstantního počtu paprsků, ale tento počet se mění podle toho, zda mezi sousedními body je velký rozdíl v barvě. Pokud ano, tak se provede rozdělení této části a vrhnou se nové paprsky. Takto je možno pokračovat až dokud není rozdíl mezi barvami menší než nějaká úroveň.

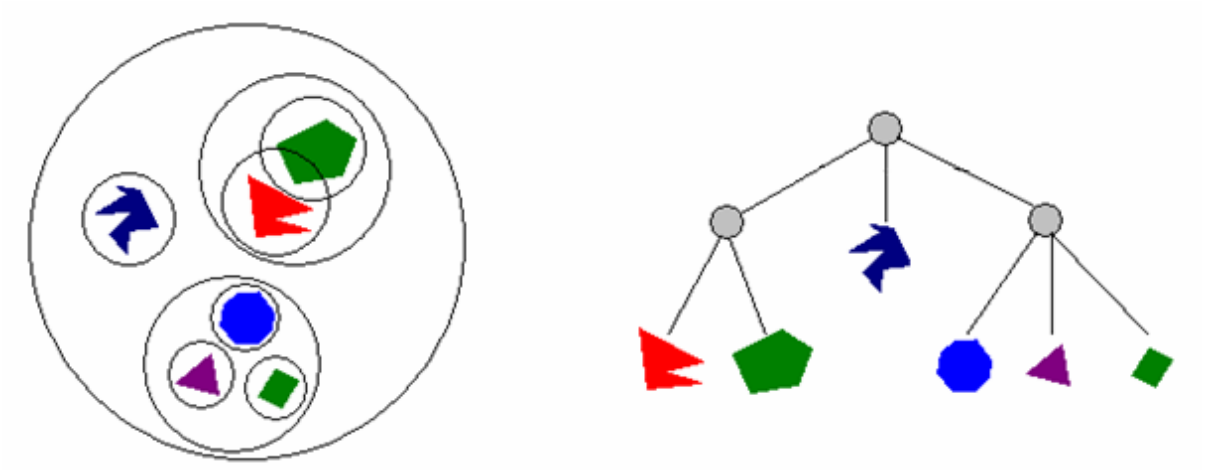
Obr. 4-5: Adaptive supersampling

4.3 Metody pro urychlení nalezení průsečíku

Při výpočtu obrázku o rozměrech 640x480 pixelů je potřeba do scény vrhnout 307 200 primárních paprsků. Každý tento paprsek je potřeba testovat na protnutí s objektem. Pokud se ve scéně nachází 100 objektů, tak už dojde k 30 miliónům testů na protnutí(a to pouze s primárními paprsky, ovšem ve scéně mohou vzniknou i paprsky sekundární a množství testů se zvyšuje). Ovšem objekty jsou ve scéně rozloženy různě a tedy není třeba testovat ty objekty, které leží za kamerou, nebo mimo zorný úhel kamery. Mezi další způsoby urychlení patří obalová tělesa a jejich hierarchie, nebo prostorové rozdělení scény.

4.3.1 Obalová tělesa

Obalové těleso je jednoduchý geometrický objekt(např. koule z důvodu rychlého výpočtu, obalující geometrii celého původního modelu. Vždy je dobré využít co nejtěsnější obal, protože bude sloužit k testům, zda jej paprsek protíná. Pokud ano, tak provedeme test se složitým modelem, pokud ne, tak pokračujeme testem jiného tělesa.



Obr. 4-6: Hierarchie obalových těles

Zdroj: [10]

Pro objekty složené z velkého počtu trojúhelníků je výhodné využít takzvaných hierarchie obalových těles. Tato hierarchie je sestavována tak, že nejprve je vytvořeno obalové těleso pro všechny trojúhelníky a pak postupně pro menší a menší skupiny trojúhelníků, až do určitého počtu, nebo do určité hloubky stromu. Při testování na průsečík se nejprve testuje, zda paprsek protne kořen stromu, pokud ano, tak se pokračuje větvemi, které jsou protnuté až k samotným trojúhelníkům.

4.3.2 Prostorové rozdělení scény

Při prostorovém rozdělení scény dojde k dělení scény postupně tak, že v daném vymezeném prostoru se nachází, či ho protíná ideálně pouze jediný objekt.

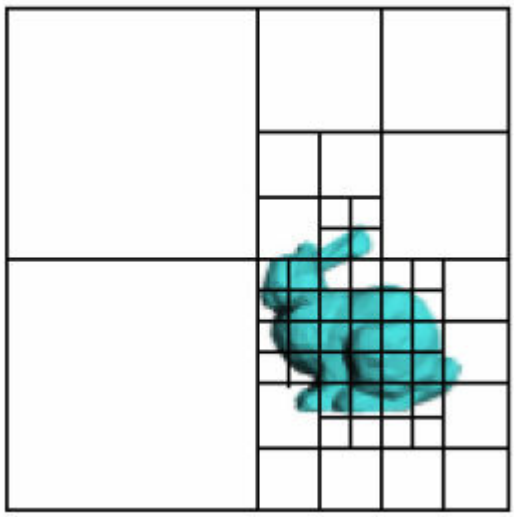
Oct-tree

Datová struktura, jejíž základem je krychle. Tato krychle je následně rozdělena na osm krychlí s hranou poloviční délky.

Při dělení scény se dělí pouze ty části krychle, které obsahují nějaké objekty. Tudíž dochází k úspoře potřebných testů s paprskem, pokud se v daném směru nenachází žádný objekt. Dělení může pokračovat až do doby, než je splněna nějaká podmínka. Tou může být pouze jeden objekt v krychli, nebo minimální velikost krychle.

BSP-tree

Zkratka BSP znamená Binary Space Partitioning, neboli binární dělení prostoru. To je přesně, co tento algoritmus dělá. Postupně vytváří strom tak, že daný prostor rozdělí na dvě části a přiřadí je danému uzlu. Postupně rozdělí uzly až do určité hloubky stromu, nebo počtu objektů v uzlu. Ideální je, aby byl strom vyvážený, tudíž v každém uzlu byl v levé i pravé větvi stejný počet objektů z důvodu stejného času procházení oběma větvemi. Dělící rovina může být dle základních os, nebo dle nějakého objektu, který je vybrán náhodně, či analyticky dle počtu objektů vlevo a vpravo od něj.



Obr. 4-7: Ukázka dělení prostoru v 2D

POV-Ray

Při studování metod pro urychlení výpočtu jsem narazil na další možný způsob optimalizace. Jedná se o způsob využitý v aplikaci POV-Ray. Tento ray tracer vytvoří obalová tělesa pro každý objekt zarovnaná dle os. Poté tyto obalová tělesa promítne do promítací roviny. Následkem toho při vrhání paprsku testuje, zda paprsek leží v 2D regionu. Pokud ano, testuje průsečík s tělesem, kterému tento region patří, pokud ne, pokračuje dále. Tato optimalizace má velmi velký vliv na rychlost vykreslování tohoto ray traceru.

5 Demonstrační aplikace

Pro ověření navrženého modelu v praxi byla vytvořena demonstrační aplikace, která měla ukázat že navržený model je snadno použitelný, přehledný a výsledný obraz, jenž je generován ray tracerem je korektní. První verze byla vytvořena podle prvotního návrhu za účelem vygenerování obrázků do Semestrálního projektu. Při psaní této první verze bylo zjištěno několik poznatků, které byly dále zapracovány do modelu. Především šlo o rozšíření objektu o materiály tak, aby mohly být univerzální. Následkem toho byly přidány další třídy a model se rozšiřoval. Stejně tak se rozšiřovala i aplikace. Jako programovací jazyk jsem si zvolil C++, jednak kvůli zkušenostem s vývojem v tomto jazyce, ale také kvůli jeho rychlosti a také výbornému vývojovému prostředí Visual Studio.

5.1 Výsledná aplikace

Výsledkem mé práce je demonstrační aplikace, která načte scénu popsanou ve zdrojovém kódu a tuto scénu zobrazí. Do scény je možné přidávat geometrická tělesa – kouli, kvádr, rovinu. Dále lze přidat světla. Počet světel, ani objektů není nijak omezen. Dále je nutné nastavit kameru. Kamera je ve scéně vždy jedna. Všechny objekty ve scéně je možné libovolně posouvat a otáčet a tím sestavit scénu dle představ. Po takovéto inicializaci scény je tato přiřazena ray traceru a může být spuštěn výpočet. Po ukončení výpočtu je scéna vykreslena do okna aplikace.

Objektům je možné přiřadit materiál. Byl vytvořen materiál počítající pomocí phongova osvětlovacího modelu výslednou barvu povrchu tělesa. Dále tento materiál počítá i s odaženými paprsky a také s paprsky lomenými.

Program je napsán v programovacím jazyce C++ s využitím grafické knihovny OpenGL pro zobrazení výsledného obrázku. Jedná se o okenní aplikaci, která v hlavním okně vykresluje výsledný obrázek a ve druhém okně vypisuje čas, který zabralo vykreslování. Tento údaj bude dále využit při měření výkonu aplikace. Případně je zde možné vypsat i další údaje potřebné pro ladění a testování aplikace.

5.2 Výkon

Při navrhování modelu i aplikace byl hlavní důraz kladen na snadné použití a rozšiřitelnost, kdežto výkon byl až na druhém místě. Ovšem i tak byla snaha udělat ray tracer co nejrychlejší. Myslím, že se to po optimalizacích podařilo

Testy byly prováděný na notebooku s procesorem Celeron 1.6GHz, 1GB RAM a Windows XP. Naměřený čas je čas, který zabere vykreslení jednoho snímku. Do tohoto času není počítán start aplikace, ani její inicializace.

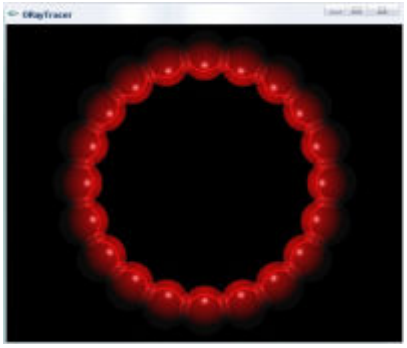
Byly vytvořeny 3 testovací scény na kterých jsem měřil různé části ray traceru a jejich vliv na celkový výkon a dobu vykreslování. V prvním testu je rozmístěno různé množství koulí v kruhu, všechny koule mají stejný materiál a to materiál, který je maximálně odrazivý a neprůhledný. V testu jsem měřil vliv počtu objektů na dobu vykreslování. Dále jsem chtěl vyzkoušet, zda s vyšší mírou rekurze roste počet paprsků a o kolik. V dalším testu je vykreslena známá scéna nazývaná Cornell Box. Jedná se o krychli, ve které jsou dva pootočené kvádry. Většinou je tato scéna využívaná pro výpočet osvětlení pomocí radiozity, ovšem je možné ji využít i v ray tracingu. V originální scéně je použito plošné světlo na „stropě“ krychle. V ray traceru nejsou plošné světelné zdroje implementovány, proto je použit rozdílný počet světel. A to postupně 1, 2, 4, 8 světel. Posledním testem je mřížka postupně 1, 3, 5 krychlí, u kterých je nastaven lom světla a průhlednost. Výsledky všech testů byly postupně zaznamenány do tabulek a byly následně zhodnoceny a okomentovány. Výsledné obrázky vzniklé ray tracingem jsou v obrazové příloze.

Testy byly prováděny v rozlišení 400x400 pixelů, s nastavenou mírou rekurze na 3. Vykreslení prázdné scény trvá 1,2s a je vrženo 160 000 primárních paprsků.

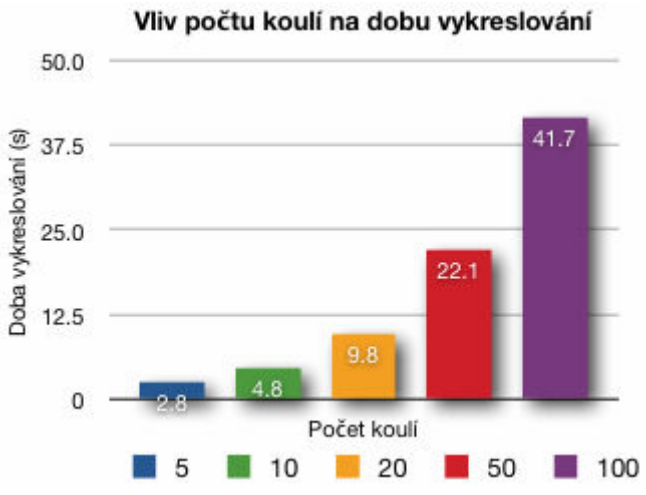
5.2.1 Testovací scéna 1

Počet koulí	Doba vykreslování (s)	Počet paprsků
5	2,81	179190
10	4,81	199522
20	9,75	236578
50	22,10	241368
100	41,73	242544

Tab. 5-1: Vliv počtu koulí na dobu vykreslování



Obr. 5-1: Testovací scéna 1

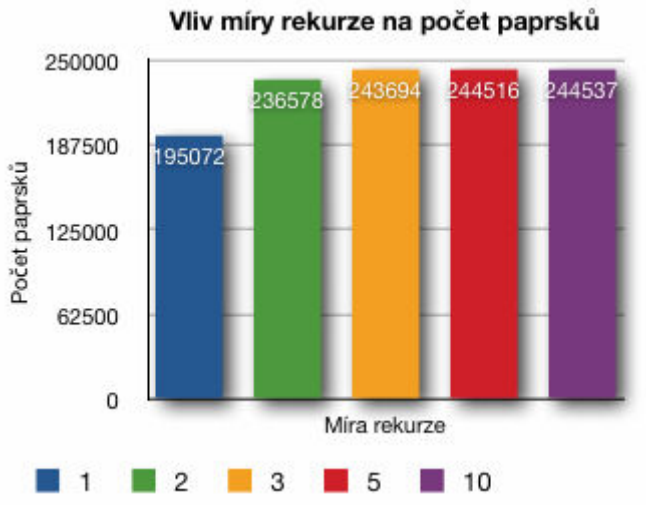


Graf 5-1: Graf vlivu počtu koulí na dobu vykreslování

Bylo proveden test na scéně, ve které se nacházel různý počet koulí v kruhu. Byla měřena doba vykreslení této scény a počtu paprsků vzniklých při výpočtu. Z naměřených výsledků plyne, že doba výpočtu záleží na počtu objektů a to lineárně a počet nových paprsků, které ve scéně vznikají s vyšším počtem koulí se stále snižuje, protože koule se vzájemně překrývají a tudíž vytváří jakoby stále stejný tvar.

Míra rekurze	Doba vykreslování (s)	Počet paprsků
1	7,92	195072
2	9,516	236578
3	9,78	243694
5	9,95	244516
10	10,02	244537

Tab. 5-2: Vliv míry rekurze na počet paprsků



Graf 5-2: Graf vlivu míry rekurze na počet paprsků

Druhým testem na podobné scéně bylo na 20 koulích umístěných do kruhu otestovat, jaký má vliv míra rekurze na dobu vykreslování a počet vzniklých paprsků. Výsledkem tohoto testu je, že pro objekty volně rozložené v prostoru je zbytečné používat hloubku větší než 3, to odpovídá dvěma odrazům paprsku. Už při této míře je většina paprsků dražena mimo ostatní objekty a proto zanikají. Rozdíl mezi hloubkou 5 a 10 je pouhých 21 nově vzniklých paprsků a na výsledný obraz nemají žádný vliv.

5.2.2 Testovací scéna Cornell Box

počet světel	Doba vykreslování (s)	Počet paprsků
1	9,23	326130
2	13,41	489194
4	23,03	815322
8	41,00	1467555

Tab. 5-3: Vliv počtu světel na dobu vykreslování

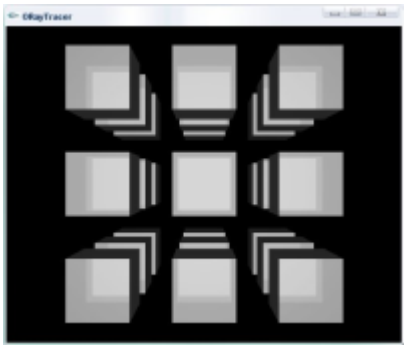
Obr. 5-2: Testovací scéna Cornell Box

V tomto testu byl měřen vliv počtu světel na dobu potřebnou pro vykreslení scény. Je zde vidět obrovský počet paprsků které vznikají. Je to dáno tím, že pro každý průsečík se počítá osvětlení s každým světlem a vrhají se stínové paprsky pro každé světlo. Materiály byly nastaveny na neodrazivé a neprůhledné. Čas potřebný pro vykreslení roste lineárně s množstvím světel do scény přidaných.

5.2.3 Testovací scéna 2

počet krychlí na hraně / celkem	Doba vykreslování (s)	Počet paprsků
1 / 1	1,57	165202
3 / 27	26,83	329805
5 / 125	196,13	696839

Tab. 5-4: Vliv počtu lomených paprsků na výkon



Obr. 5-3: Testovací scéna 2

Tento test byl zaměřen na výpočet lomených paprsků. Výpočet lomu paprsku je časově náročnější, než výpočet jeho odrazu. Krychle byly zobrazeny v pravidelné mřížce. Jejich množství bylo zadáno počtem krychlí na hraně.

5.3 Možné rozšíření

Demonstrační aplikace byla psána za účelem otestování správnosti navrženého modelu. Toto se podařilo a aplikace je napsána přesně modelu a vše funguje správně tak jak má. Jako základní rozšíření, které bych viděl pro vyšší kvalitu objektů je načítání polygonálních modelů ze souborů. Ať už půjde o 3DS soubory, nebo jiný formát, myslím, že to by zvedlo zajímavost výsledných obrazů o mnoho nahoru. Tato funkce nebyla implementována, protože se jedná pouze o rozšíření třídy shape a tudíž ve vztahu k modelu by neměla žádný přínos. V souvislosti se složitými objekty se nabízí také možnost implementace optimalizace pomocí obalových těles, nebo v souvislosti s polygonálními modely spíše pomocí rozdělení prostoru. Toto opět nebylo implementováno vzhledem k použití základních geometrických útvarů, se kterými je výpočet průsečíku velmi rychlý. A je zbytečné testovat, zda paprsek protne kouli, která je obalovým tělesem krychle a pak teprve testovat na průsečík s krychlí.

Další možné rozšíření se nabízí v podobě dalších osvětlovacích modelů, či například nefotorealistického vykreslování(ala komiks, nebo malba tužkou). Model se dá na tyto metody snadno rozšířit.

5.4 Popis nastavitelných parametrů

Veškerá nastavení základních parametrů se nachází v souboru `const.h`. Uvádím zde přehled konstant i s popisem pro přehlednost a srozumitelnost.

MAX_DIST – maximální vzdálenost od kamery, do které se paprsek vyhodnocuje. Pokud je nalezen průsečík ve větší vzdálenosti, není brán v úvahu.

MIN_DIST - minimální vzdálenost od kamery, od které se paprsek vyhodnocuje. Pokud je nalezen průsečík v menší vzdálenosti, není brán v úvahu. Tato vzdálenost je důležitá pro odstranění artefaktů, které mohou vzniknout při výpočtu. Bez této vzdálenost je například při výpočtu odražených paprsků dost často nalezen průsečík tělesa sama se sebou.

MAX_RAY_DEPTH – určuje hloubku rekurze sledování paprsku. Pokud je nastaven na 1, jsou počítány pouze primární paprsky. Pokud je nastaven na 2 a také materiál objektů je nastaven na odraz světla, tak jsou vyhodnocovány odrazy. Pro výpočet lomu paprsku je potřeba mít nastaveno minimálně 3.

EPSILON – minimální konstanta. Při výpočtech může dojít k zaokrouhlovacím chybám a proto je zavedena tato konstanta, která má za úkol tyto chyby odstranit. Dále je použita v některých případech pro posunutí paprsku ve směru normály, aby nedocházelo k nalezení průsečíku paprsku od objektu opět s tím samým objektem.

XRES – určuje šířku výsledného obrázku
YRES – určuje výšku výsledného obrázku

Notifikace určují po výpočtu jakého množství obrazu bude zavolána funkce NotifyFunction.
PER_PIXEL – po vykreslení každého bodu
PER_ROW – po vykreslení jednoho řádku
PER_IMAGE – po vykreslení celého obrázku

Způsoby mapování textury
MAP_SPHERE – sférické mapování na kouli
MAP_PLANE – mapování na rovinu
MAP_BOX – mapování na krychli

CSG operace
CSG_UNION – sloučení dvou těles
CSG_INTERSECTION – průnik dvou těles
CSG_DIFFERNCE – odečtení druhého tělesa od prvního

6 Závěr

V Diplomové práci jsem navázal na předchozí zkušenosti s programováním 3D grafických aplikací a ray traceru. Zamyslel jsem se nad snadno rozšiřitelným minimalistickým objektově orientovaným modelem takového ray traceru. Prostudoval jsem různé aplikace a jejich návrhy, každá implementace se liší i přes na první pohled shodný přístup a algoritmus. Každá aplikace je specializována na něco jiného, a tedy volí i jiný přístup a návrh. Pokusil jsem se vzít z každého to nejlepší a vytvořit model, který funguje a je velmi snadno pochopitelný a rozšiřitelný. Pro ilustrační obrázky a pro doplnění této práce jsem vytvořil jednoduchou aplikaci, která umí vykreslit scénu s jednoduchými objekty.

Výsledkem mé práce je návrh tříd pro minimalistický objektově orientovaný ray tracer a jeho implementace. Model tříd vznikal postupně současně s rozšiřováním mých poznatků studiem zdrojových kódu jiných aplikací a na základě zkušeností s vlastní implementací. Aplikace, která v průběhu psaní práce vznikla koresponduje s navrženým modelem a je praktickou ukázkou funkčnosti tohoto modelu v praxi. Tato aplikace vykresluje základní geometrické útvary jako kouli, kvádr a rovinu, umí jim přiřadit libovolný materiál. Dále je do scény možné umístit světla a všechny objekty libovolně posouvat a otáčet. Výsledný obraz, který vznikne vykreslením scény odpovídá obrazům z jiných ray tracerů při stejném nastavení parametrů. Na rozdíl od mnoha jednoduchých implementací, na které je možné narazit, umí zde implementovaný i CSG grafiku, díky které se za vhodného složení objektů dá vytvořit velmi zajímavých tvarů.

Jako případné další rozšíření do budoucna vidím implementaci polygonálních modelů a v souvislosti s nimi také rozšíření o prostorové rozdělení scény.

Literatura

[1] M. Page-Jones, *Základy objektově orientovaného návrhu v UML*, GRADA 2001

[2] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Návrh programů pomocí vzorů*. GRADA 2003

[3] *Raytracing - Buildings* [online][20.4.2008]. Dostupný z WWW: http://www.helbing-online.de/Homepage/img/Raytracing/Gebaeude/index2.html

[4] IMDB.com, *Gollum (Character)* [online]. [20.4.2008]. Dostupný z WWW: http://www.imdb.com/character/ch0000152/photogallery-1

[5] Kadi Bouatouch, *Ray Tracing* [Online] [20.4.2008]. Dostupný z WWW: http://www.irisa.fr/siames/Kadi.Bouatouch/enseignement/Courses/RayTracing.pdf

[6] wikipedia.org, *Ray Tracing* [Online] [20.4.2008], Dostupný z WWW: http://en.wikipedia.org/wiki/Raytracing

[7] Benjamin Schnaidt, *Raytracing* [Online] [20.4.2008], Dostupný z WWW: http://www.scimacros.de/Files/Samples/SampleRaytracing.pdf

[8] cs.wikipedia.org, *Aliasing* [Online] [20.4.2008], Dostupný z WWW: http://en.wikipedia.org/wiki/Aliasing

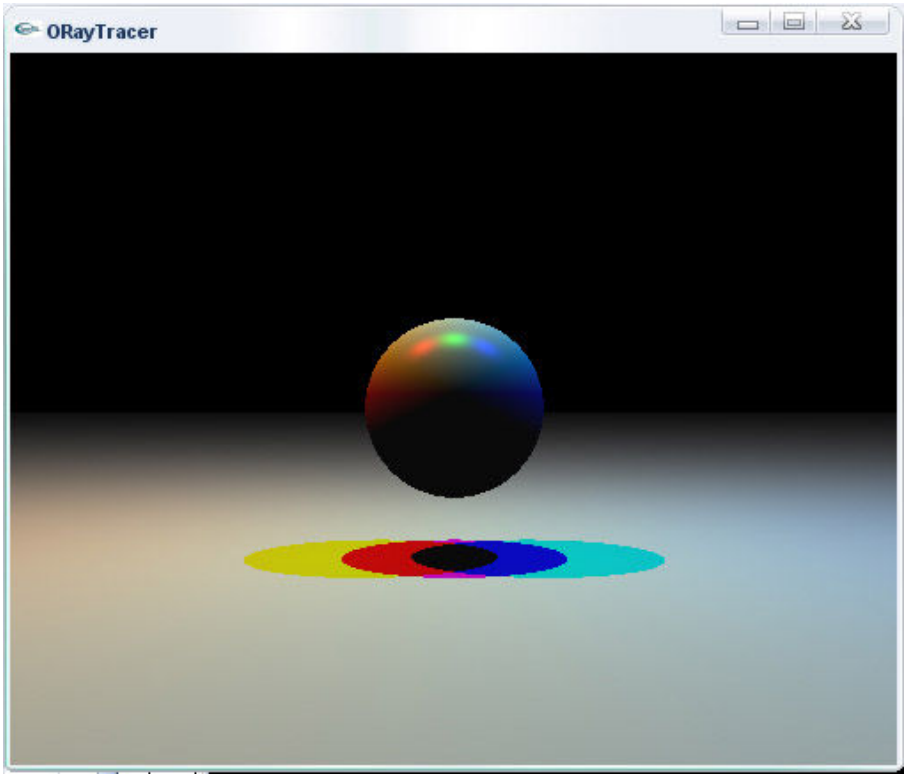
[9] www.sgi.com, *Tackling Large Volume Visualization Challenges with Real-Time Ray Tracing* [Online] [20.4.2008], Dostupný z WWW: http://www.sgi.com/pdfs/3883.pdf

[10] Markus Trenkwalder, *Erweiterung des Raytracer,* [Online] [20.4.2008], Dostupný z WWW: http://www.trenki.net/files/Raytracing2.pdf

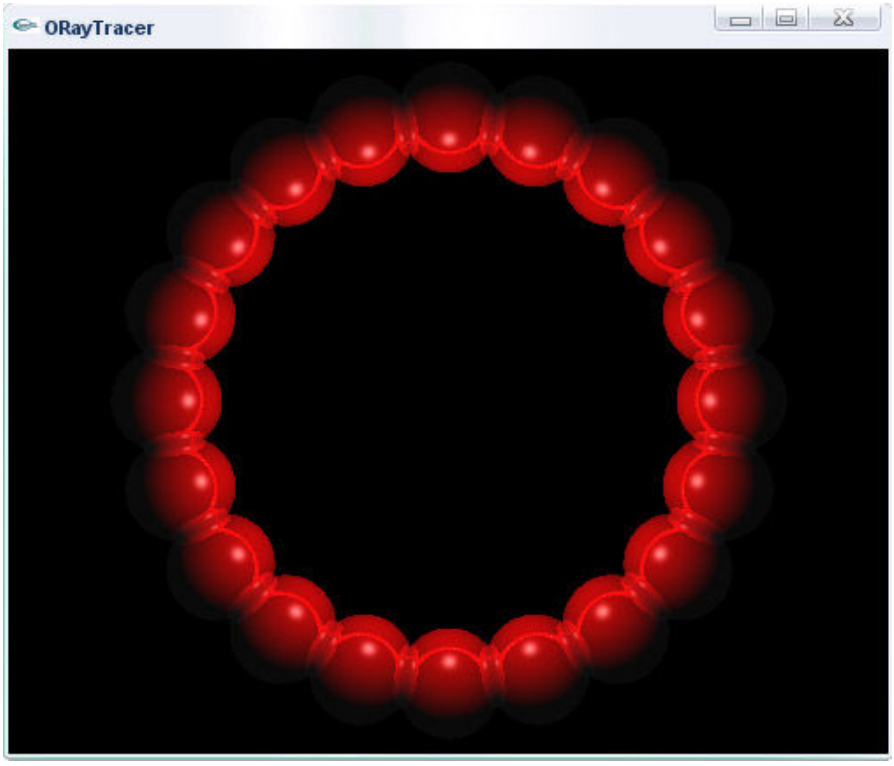
Příloha 1: Zdrojový kód – Testovací scéna 1

```
void LoadScene1(RayTracer *RT, int numspheres)
{
	Scene *scene = new Scene;
	RT->SetScene(scene);

	Camera *cam = new Camera;
	cam->Create(1,90,RT->ResX*1.0/RT->ResY);
	Scene->SetCamera(cam);

	Light *light = new Light;
	light->SetColor(Color(255,0,0));
	light->Trans.SetPosition(Vector(0,0,40));
	scene->AddLight(light);

	PhongMaterial *pm = new PhongMaterial;
	pm->SetParams(Color(255,0,0));
	pm->SetSpecular(32);
	pm->SetReflection(0.5);
	pm->SetRefraction(0,1);

	Sphere *sp = new Sphere;
	sp->Create(3);

	for(int i=0;i<numspheres;i++)
	{
		Object *obj = new Object;
		obj->SetMaterial(pm);
		obj->SetShape(sp);
		obj->Trans.SetPosition(Vector(
				16*cos(i*(6.283852)/numspheres),
				16*sin(i*(6.283852)/numspheres),50));
		scene->AddObject(obj);
	}
}
```

Příloha 2: Obrazová příloha

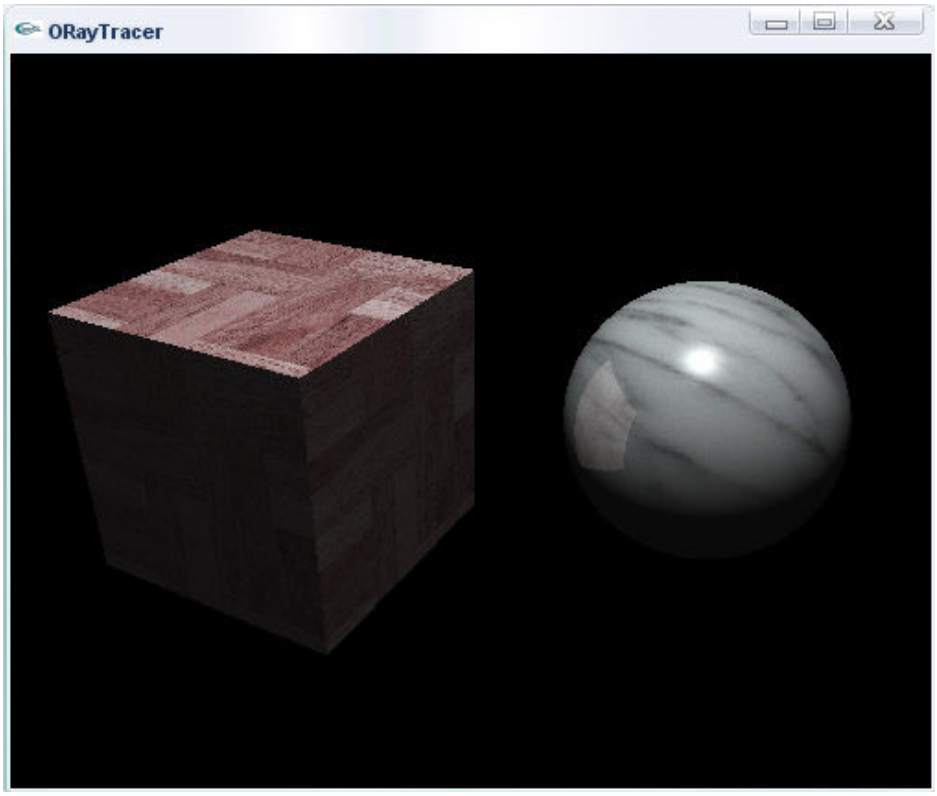


Tři barevná světla

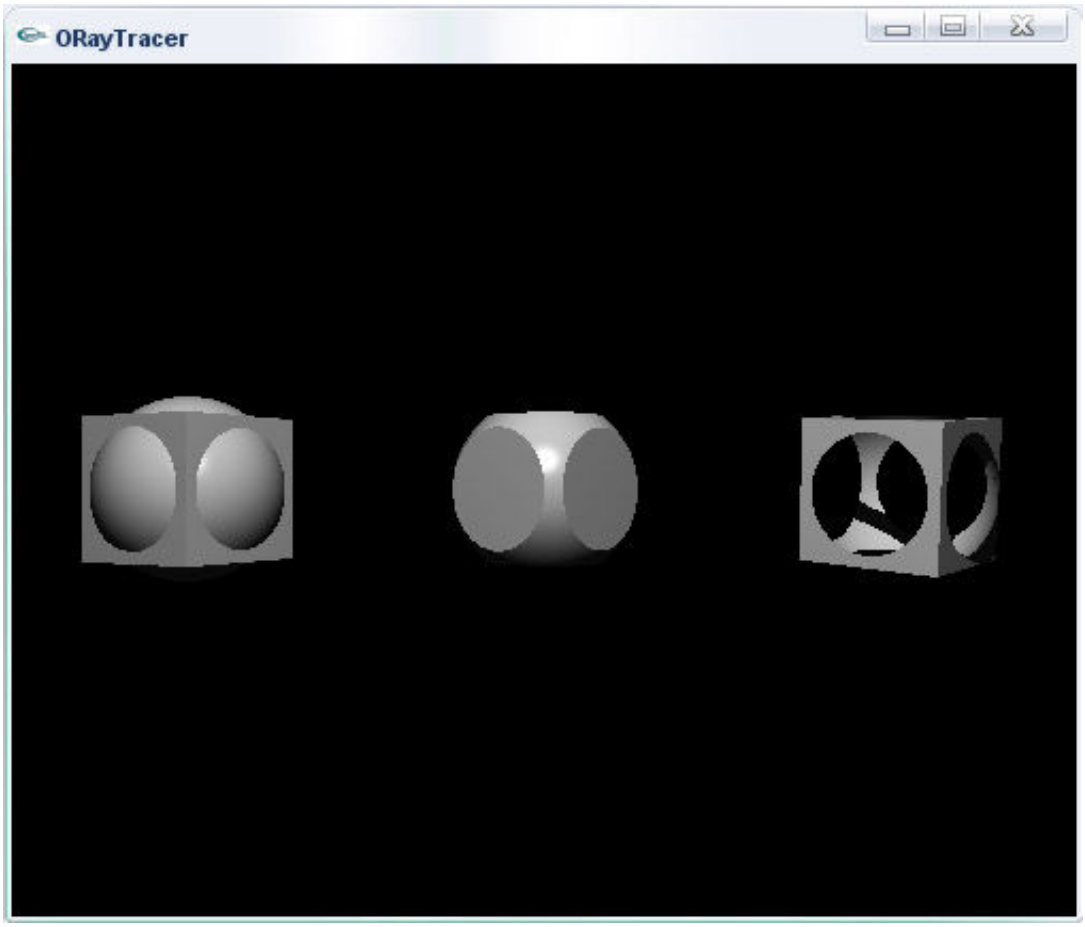


Koule v kruhu

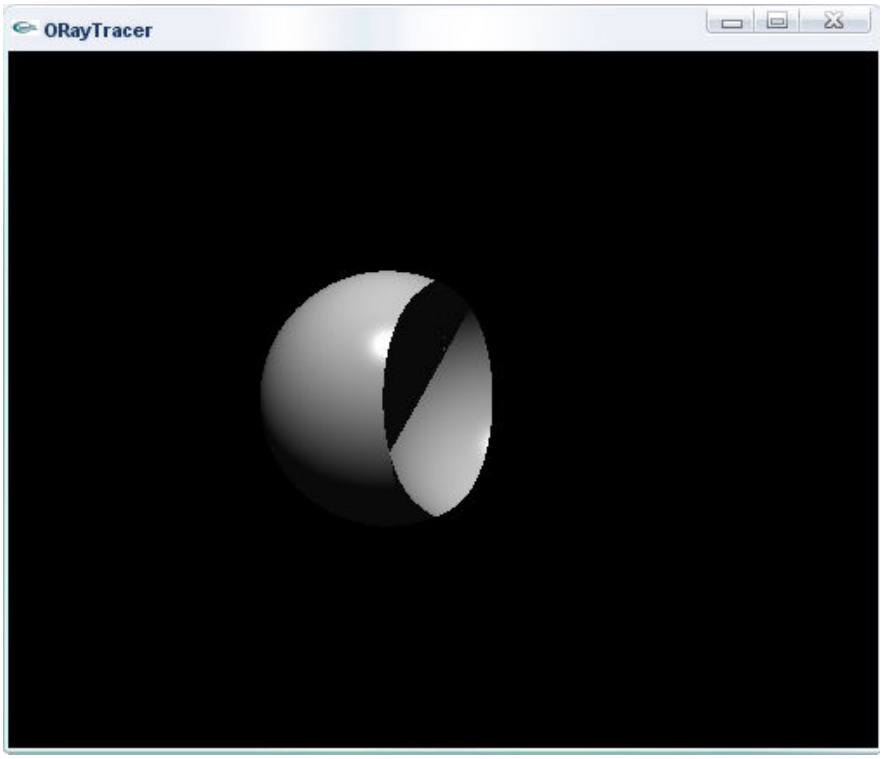
Cornell Box



Texturování koule a krychle



CSG operace



CSG operace odečtení dvou koulí

Příloha 3: CD se zdrojovými kódy a dokumentací