FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Globální osvětlovací modely
Student:	Matěj Hoffmann
Vedoucí:	Ing. Radek Richtr, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

1) Proveďte širokou rešerši lokálních a globálních osvětlovacích modelů. Soustřeďte se na srozumitelnost předkládaných informací.
2) Krátce analyzujte optimalizaci renderovacího procesu pomocí datových struktur.
3) Vyberte kombinaci zobrazovací metody a datové struktury a navrhněte váš renderer.
4) Implementujte navržený prototyp.
5) Otestujte implementovaný renderer kvalitativně i kvantitativně. Diskutujte výsledky.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 3. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Globální osvětlovací modely

Matěj Hoffmann

Katedra softwarového inženýrství
Vedoucí práce: Ing. Radek Richtr, Ph.D.

13. května 2021

Poděkování

Chtěl bych poděkovat svému vedoucímu bakalářské práce Ing. Radku Richtrovi, PhD. za vedení a pomoc při psaní bakalářské práce. Velké díky také patří všem, kteří se mnou práci diskutovali a rodině za ohromnou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Hoffmann, Matěj. *Globální osvětlovací modely*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá teorií přenosu světla a metodami řešícími globální osvětlování a následné implementaci jedné z těchto metod. V první části se práce nejprve věnuje teoretickým podkladům z oblasti počítačové grafiky, matematiky a fyziky. Pomocí těchto poznatků je následně vysvětlena problematika umělé syntézy obrazu a jsou ukázány metody pro její řešení. Důraz je kladen především na řešení zobrazovací rovnice pomocí metody Monte Carlo a různých optimalizačních prvků. V závěrečné části se práce věnuje implementaci vlastního renderovacího systému založeném na algoritmu path tracing.

Klíčová slova globální osvětlovací modely, ray tracing, path tracing, rendering, 3D grafika, simulace světla, realistický rendering, C++, monte carlo

Abstract

This work is focused on the theory of light transportation and the globall illumination methodes and the subsequent implementation of one of them. The first part of the thesis deals with the theoretical background in the field of computer graphics, mathematics and physics. With the help of this knowledge, the issue of artificial image synthesis is then explained and methodes on how to solve it are presented. Emphasis is placed primarily on solving the rendering equation using Monte Carlo integration and various optimization elements. The final part of the thesis deals with the implementation of the own rendering system based on the path tracing algorithm.

Keywords global illumination methodes, ray tracing, path tracing, rendering, 3D graphics, light simulation, realistic rendering, C++, monte carlo

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato práce je věnována úzkému odvětví počítačové grafiky, které se zabývá věrohodnou simulací šíření světla prostorem a vytvářením realisticky vypadajících obrazů pomocí počítačově vytvořených modelů. Záměrem této oblasti počítačové grafiky je vytváření obrazů na první pohled nerozeznatelných od fotografií, čehož hojně využívá například herní či filmový průmysl. Pro ten je toto téma dokonce tak důležité, že se filmová akademie rozhodla udílet za určité objevy slavného Oscara za jejich přínos tomuto průmyslu.

Problematice realistického renderování se vědci věnují již mnoho let, během kterých přišli s mnoha různými nápady a technikami, jak výše zmíněný problém řešit.

V počátcích, kdy byl výkon počítačů stále nedostatečný, se uchylovali k metodám, které sice nevytvářely příliš realistickou grafiku, ale za to byly schopné problém řešit rychle. Takto vzniklo renderování pomocí rasterizace, které se dochovalo dodnes a které se využívá především v počítačových hrách.

To, jak realisticky vykreslit scénu už v té době bylo také známo, ale počítače neměly dostatečný výkon k efektivnímu řešení vymyšlených algoritmů. To dokazují i slova jednoho z nejvlivnějších vědců počítačové grafiky J. Kajiya, který údajně prohlásil: „ray tracing is not slow – computers are“.

S růstem výkonu počítačů, obzvlášť v posledních dvou desetiletích, se původní myšlenky staly použitelnými a nyní se neustále posouvá hranice toho, jak dobře a věrohodně jsme schopni simulovat reálná prostředí.

To vedlo i k volbě tohoto tématu, jelikož se jedná o velice zajímavou oblast počítačové grafiky, která využívá poznatků z téměř každého kouta počítačové vědy.

Cíl práce

Cílem této bakalářské práce je přiblížení problematiky realistického renderování za pomocí metod globálního osvětlení. Dále je cílem tyto teoretické poznatky implementovat a vytvořit tak vlastní renderovací systém. Ten by měl být optimalizován a být lehce rozšířitelný. Cílem je také vyhodnotit výstupy tohoto systému a ukázat rozdíly v optimalizovaném a neoptimalizovaném řešení. Závěrečným cílem je tuto práci vytvořit jako možný výukový materiál.

Struktura

Práce je rozdělena na dvě části – teoretickou a realizační. První část se nejprve věnuje základům počítačové grafiky, matematiky a fyziky, které jsou potřebné k následnému pochopení teorie přenosu světla a globálních iluminačních metod. Druhá část se zabývá implementací vlastního renderovacího systému pomocí poznatků z předchozí části. Implementace je následně testována a jsou ukázány výstupy z tohoto řešení.

Část I

Teoretická část

Kapitola 1

Matematické základy

Napříč touto prací se budeme setkávat se základními pojmy z oblasti matematiky a geometrie, zejména s vektory, body, normálou a paprsky, které je velmi dobré znát. Tato kapitola slouží k seznámení se s těmito pojmy.

1.1 Souřadnicový systém

V praxi se nejčastěji používá tzv. *Kartézský souřadnicový systém*, který můžeme znát z běžné geometrie. Abychom si tento souřadnicový systém mohli popsat, je nutné se seznámit s několika předpoklady uvedenými níže.

Vektorový prostor

Mějme libovolné těleso $\mathbb{T}$, neprázdnou množinu V a dvě zobrazení:

$$\oplus : V \times V \to V, \qquad \odot : \mathbb{T} \times V \to V.$$

Pokud následně platí:

(i) $\forall a, b \in V : a \oplus b = b \oplus a$,

(ii) $\forall a, b, c \in V : (a \oplus b) \oplus c = a \oplus (b \oplus c)$,

(iii) $\forall \alpha, \beta \in \mathbb{T}, \forall a \in V : \alpha \odot (\beta \odot a) = (\alpha \cdot \beta) \odot a$,

(iv) $\forall \alpha \in \mathbb{T}, \forall a, b \in V : \alpha \odot (a \oplus b) = (\alpha \odot a) \oplus (\alpha \odot b)$,

(v) $\forall \alpha, \beta \in \mathbb{T}, \forall a \in V : (\alpha + \beta) \odot a = (\alpha \odot a) \oplus (\beta \odot a)$,

(vi) $\forall a \in V : 1 \odot a = a$,

(vii) $\exists \theta \in V, \forall a \in V : 0 \odot a = \theta$,

řekneme, že V je *vektorový prostor* nad tělesem $\mathbb{T}$ s vektorovými operacemi $\oplus$ a $\odot$ [1].

Prvku $\vec{x}$ vektorového prostoru V říkáme vektor a jedná se o uspořádanou množinu $(x_1, x_2, \ldots, x_n)$, kde $x_i \in \mathbb{T}$ nazveme i-tou souřadnicí vektoru $\vec{x}$. Výše zmíněný symbol θ značí tzv. nulový vektor, který má v případě reálných čísel všechny souřadnice rovné nule [1].

Vektorový prostor V dimenze n můžeme generovat pomocí n lineárně nezávislých vektorů. Tyto vektory tvoří uspořádanou množinu zvanou báze a jsou označovány jako bazické. Každý další vektor $\vec{v}$ prostoru V lze popsat lineární kombinací těchto bazických vektorů $\vec{b}_i$ a existuje právě jedna kombinace skalárů $\alpha_i \in \mathbb{T}$ tak, že platí:

$$\vec{v} = \alpha_1 \vec{b}_1 + \alpha_2 \vec{b}_2 + \ldots + \alpha_n \vec{b}_n \ ,$$

kde α_i reprezentují jednotlivé souřadnice vektoru $\vec{v}$ v bázi $(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n)$ [2].

V praxi se nejčastěji používá tzv. *standardní báze* $\mathcal{E} = (\vec{e_1}, \vec{e_2}, \ldots, \vec{e_n})$. Ta je složena z vektorů $\vec{e}_i$, kde pro každý takový vektor platí, že jeho i-tá souřadnice je rovna jedné a všechny ostatní mají hodnotu nula [1].

Pro potřebu této práce budeme jako těleso $\mathbb{T}$ uvažovat pouze množinu všech reálných čísel $\mathbb{R}$.

Afinní a Euklidovský prostor

Afinním prostorem $A = A(V)$ myslíme trojici $(A, V, +)$, kde A je neprázdná množina, jejíž prvky nazýváme body, V je vektorový prostor, $+$ je operace $A \times V \to A$ a platí axiomy zmíněné níže [3].

(i) $\forall a \in A, \vec{\theta} \in V : a + \vec{\theta} = a$,

(ii) $\forall a \in M, \forall \vec{v}, \vec{w} : a + (\vec{v} + \vec{w}) = (a + \vec{v}) + \vec{w}$,

(iii) $\forall a, b \in A, \exists \vec{v} \in V : a + \vec{v} = b; \quad \vec{v} = a - b$.

Euklidovský prostor $E = E(V)$, je afinní prostor A obohacený o skalární součin nad vektorovým prostorem V. Dimenze euklidovského prostoru je určena dimenzí příslušného vektorového prostoru V [3].

Euklidovský prostor si zde zavádíme z toho důvodu, že společně s afinním prostorem přináší pojem bod a zároveň se jedná o unitární metrický prostor, tj. prostor, ve kterém můžeme měřit vzdálenost mezi dvěma body, nebo-li velikost vektoru.

Kartézský souřadnicový systém

Kartézskou soustavou souřadnic v Euklidovském prostoru $E(V)$ nazveme množinu $S = (p, \vec{b_1}, \vec{b_2}, \ldots, \vec{b_n})$, kde $(\vec{b_1}, \vec{b_2}, \ldots, \vec{b_n})$ je ortonormální báze vektorového prostoru V a bod p je tzv. *počátek soustavy souřadnic* [3].

Jako osy souřadnicového systému považujeme přímky procházející bodem p ve směru jednotlivých bazických vektorů a souřadnice libovolného bodu $a \in E$ lze vyjádřit jako vektor $\overrightarrow{a - p}$.

Často také rozlišujeme mezi pravotočivou a levotočivou soustavou souřadnic ve 3D. Budeme-li předpokládat, že osa y ukazuje směrem vzhůru a osa x ukazuje doprava, máme dvě možnosti jak určit kterým směrem ukazuje osa z. V případě, že ukazuje směrem k nám, mluvíme o pravotočivé soustavě. V opačném případě se jedná o levotočivou soustavu.

Zda-li se jedná o pravotočivou nebo levotočivou soustavu lze určit pomocí pravidla pravé ruky. Nasměřujeme-li prostředníček pravé ruky ve směru osy x, palec ve směru osy y a ukazuje-li ukazovák ve směru osy z, jedná se o pravotočivou soustavu. V opačném případě se jedná o levotočivou.

1.2 Vektor, bod a matice

Nyní si konečně můžeme popsat základní prvky, se kterými se v počítačové grafice setkáme nejčastěji.

Vektor

Jak již bylo řečeno, vektor je uspořádaná n-tice definovaná nad nějakým vektorovým prostorem. Často se značí jako $\vec{x} = (x_1, x_2, \ldots, x_n)$, kde x_i představuje jednotlivé souřadnice a $\vec{x}$ značí samotný vektor. Často se také můžeme setkat s vektorem zapsaným ve sloupcovém tvaru:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} .$$

Při zápisu v řádkovém tvaru se využívá jeho transpozice:

$$\vec{x}^T = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix} .$$

Vektor si v počítačové grafice můžeme představit jako orientovanou úsečku, která má nějaký směr, velikost a může se nacházet kdekoliv v prostoru. Vektory můžeme sčítat a odčítat a získat tím nový vektor. Stejně tak můžeme vektory násobit skalárem, čímž můžeme měnit jejich velikost a orientaci.

Bod

Na rozdíl od vektoru, bod udává pozici v prostoru vzhledem k počátku. Body od sebe lze odečítat, čímž získáme vektor posunu, ale již nelze dva body sčítat, jelikož se jedná o nedefinovanou operaci. Místo toho můžeme k bodu přičítat vektor a tím získat nový bod, který je posunutý ve směru tohoto vektoru o vzdálenost rovné jeho velikosti. Ačkoliv se bod značí stejným způsobem jako vektor, v počítačové grafice se tyto dvě entity odlišují kvůli jejich rozdílným vlastnostem.

Normála

Normálou, nebo také normálovým vektorem, je nazýván vektor, který je kolmý na (*n-1*)-dimenzionální podprostor v *n*-dimenzionálním prostoru. Ve 3D prostoru se pak jedná o vektor, který je kolmý na rovinnou plochu [4]. V počítačové grafice také často předpokládáme, že se jedná o jednotkový vektor.

1.3 Operace nad vektory

Nad vektory existují operace, které jsou z hlediska počítačové grafiky velmi důležité a používají se velmi často. Z tohoto důvodu jim bude věnována tato část textu, kde se s těmito operacemi seznámíme. Budeme předpokládat, že pracujeme v kartézském souřadnicovém systému s reálnými čísly.

Normalizace a jednotkový vektor

Normalizace je operace, která libovolný vektor převede na vektor jednotkový a zároveň zachová jeho směr. Takový vektor, jak už z názvu vypovídá, má velikost rovnu jedné. Toho lze dosáhnout tak, že se každá souřadnice daného vektoru vydělí jeho velikostí. Velikost tří-dimenzionálního vektoru tak vypočítáme následovně [5]:

$$||\vec{v}|| = \sqrt{v_1^2 + v_2^2 + v_3^2} \ ,$$

kde v_i značí jednotlivé souřadnice vektoru a $|\vec{v}|$ značí velikost vektoru. Obecně pak platí, že velikost *n*-dimenzionálního vektoru se spočítá jako [5]:

$$||\vec{v}|| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2} \ .$$

Normalizaci vektoru $\vec{v}$ pak spočítáme následovně:

$$\vec{n} = \left(\frac{\vec{v}}{||\vec{v}||}\right) = \left(\frac{v_1}{||\vec{v}||}, \frac{v_2}{||\vec{v}||}, \ldots, \frac{v_n}{||\vec{v}||}\right) \ ,$$

kde $\vec{n}$ značí výsledný znormalizovaný vektor [6].

Skalární součin

Skalární součin, v angličtině také zvaný jako *dot product*, vyjadřuje vztah mezi velikostí dvou vektorů a úhlem, který svírají. Mějme dva vektory $\vec{u} = (u_1,\ u_2,\ \ldots,\ u_n)$ a $\vec{v} = (v_1,\ v_2,\ \ldots,\ v_n)$ v n-dimenzionálním prostoru, kde $n \geq 2$. Skalární součin mezi těmito vektory značíme $\vec{u} \cdot \vec{v}$ a je definován následovně [7]:

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i v_i = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n \ .$$

Může být také znázorněn jako součin transponovaného vektoru $\vec{u}^T$ a vektoru $\vec{v}$ [7]:

$$\vec{u}^T \vec{v} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 & v_2 & \ldots v_n \end{bmatrix} \ .$$

Geometricky je skalární součin definován jako [7]:

$$\vec{u} \cdot \vec{v} = ||\vec{u}||\,||\vec{v}||\cos\alpha \ ,$$

kde α představuje velikost úhlu mezi těmito vektory [7]. Tato geometrická vlastnost nám dává jednoduchý způsob, jak zjistit úhel, který tyto dva vektory svírají. V případě, že vektory $\vec{u}$ a $\vec{v}$ jsou jednotkové, získáme úhel α rovnou z výpočtu skalárního součinu.

Vektorový součin

Vektorový součin, také známý jako *cross product*, je binární operace nad dvěma vektory v trojrozměrném vektorovém prostoru a jeho výsledkem je vektor, který je kolmý k oběma původním.

Mějme dva vektory $\vec{u} = (u_1,\ u_2,\ u_3)$ a $\vec{v} = (v_1,\ v_2,\ v_3)$. Vektorový součin mezi těmito vektory značíme $\vec{u} \times \vec{v}$ a je definován jako vektor kolmý k vektorům $\vec{u}$ a $\vec{v}$ s velikostí rovnou obsahu rovnoběžníku, který oba vektory určují [8]:

$$\vec{u} \times \vec{v} = \vec{n}\,||\vec{u}||\,||\vec{v}||\sin\alpha \ ,$$

kde $\vec{n}$ značí jednotkový vektor kolmý k rovině, ve které leží vektory $\vec{u}$ a $\vec{v}$ a α je úhel mezi nimi. Vektorový součin lze také vypočítat pouze za pomoci jednotlivých souřadnic vektorů $\vec{u}$ a $\vec{v}$ [8]:

$$\vec{u} \times \vec{v} = (u_2 v_3 - v_2 u_3,\ u_3 v_1 - v_3 u_1,\ u_1 v_2 - v_1 u_2) \ .$$

1.4 Matice a operace s nimi

Matice je další z důležitých pojmů lineární algebry a počítačové grafiky. Mnoho kalkulací a operací lze provádět právě díky maticím, kterým bude věnována tato část textu. Jako jednu z nejdůležitějších operací můžeme označit transformace, které dokáží manipulovat s vektory nebo převádět mezi různými souřadnicovými systémy. Tato část textu čerpá ze studijního textu k předmětu BI-LIN [1] a předpokládá, že zacházíme s prostorem nad reálnými čísly.

Matice

Matice je uspořádaný soubor čísel zapsaných do tabulky, která má určitý počet řádek a sloupců. Matici o mn prvcích nazveme matici typu $m \times n$, kde m je počet řádků a n je počet sloupců. Matice se obvykle značí takto:

$$\mathbb{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} ,$$

kde a_{ij} jsou prvky matice a čísla i a j jsou řádkový, respektive sloupcový index. Množinu všech matic značíme $\mathbb{R}^{m,n}$. $\mathbb{A}_{:j} \in \mathbb{R}^{m,1}$ značí j-tý sloupec matice $\mathbb{A}$:

$$\mathbb{A}_{:j} = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}$$

a $\mathbb{A}_{i:} \in \mathbb{R}^{1,n}$ značí i-tý řádek matice $\mathbb{A}$:

$$\mathbb{A}_{i:} = \begin{pmatrix} a_{i1} & a_{i2} & \dots & a_{in} \end{pmatrix} .$$

Řádky a sloupce matice si tak můžeme představit jako samostatné vektory. Této vlastnosti se například využívá k zápisu báze, kde každý sloupec představuje jeden bazický vektor.

Násobení matice číslem a sčítání matic

Násobení matice číslem probíhá po prvcích. To znamená, že každý prvek a_{ij} matice $\mathbb{A}$ je násoben číslem α zvlášť. Platí tedy:

$$\alpha\mathbb{A} = \begin{pmatrix} \alpha a_{11} & \alpha a_{12} & \dots & \alpha a_{1n} \\ \alpha a_{21} & \alpha a_{22} & \dots & \alpha a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha a_{m1} & \alpha a_{m2} & \dots & \alpha a_{mn} \end{pmatrix} .$$

Součet dvou matic se také provádí po jednotlivých prvcích. Pro matice $\mathbb{A}$ a $\mathbb{B}$ a jejich prvky a_{ij} respektive b_{ij} platí:

$$\mathbb{A} + \mathbb{B} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \dots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn} \end{pmatrix} .$$

Z tohoto plyne, že matice $\mathbb{A}$ a $\mathbb{B}$ musejí mít stejné rozměry a pro obě tak platí, že náleží stejnému prostoru $\mathbb{R}^{m,n}$.

Násobení matic

Násobení matic je o něco složitější operace, než zmíněné předchozí. Mějme matici $\mathbb{A} \in \mathbb{R}^{m,n}$ a $\mathbb{B} \in \mathbb{R}^{n,p}$ s prvky $a_{ij} \in \mathbb{A}$ a $b_{ij} \in \mathbb{B}$. Pro výslednou matici $\mathbb{D} \in \mathbb{R}^{m,p}$ a její prvky d_{ij} platí:

$$d_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \ .$$

Všimněme si, že matice $\mathbb{A}$ a $\mathbb{B}$ mohou být různého typu, ale musejí splňovat podmínku, že počet sloupců matice $\mathbb{A}$ je roven počtu řádků matice $\mathbb{B}$. Z tohoto důvodu není součin matic komutativní a záleží tak na jejich pořadí. V případě prohozeného součinu $\mathbb{BA}$ dokonce může nastat situace, že výsledek není definován.

Transpozice matice

Transpozice je další z častých operací prováděných nad maticemi. Mějme matici $\mathbb{A} \in \mathbb{R}^{m,n}$ s prvky a_{ij}. Transpozicí matice $\mathbb{A}$ je matice z $\mathbb{R}^{n,m}$, jejíž prvek v *jtém* řádku a *itém* sloupci je roven a_{ij}. Tato matice se značí jako $\mathbb{A}^T$.

V praxi to znamená, že vezmeme řádky matice $\mathbb{A}$ a zapíšeme je do sloupců, přičemž zachováme pořadí. Pro příklad můžeme uvést:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} .$$

Inverze matice

K tomu, abychom mohli matici invertovat, musí splňovat že je regulární. To ve zkratce znamená, že daná matice je čtvercového tvaru (typu $n \times n$) a její řádky jsou lineárně nezávislé [9]. Mějme takovou matici $\mathbb{A} \in \mathbb{R}^{n,n}$. Existuje-li matice $\mathbb{B} \in \mathbb{R}^{n,n}$ taková, že platí:

$$\mathbb{AB} = \mathbb{BA} = \mathbb{E} \ ,$$

nazýváme ji maticí inverzní k matic $\mathbb{A}$. Matice $\mathbb{E} \in \mathbb{R}^{n,n}$ zde značí jednotkovou matici. Ta má takové vlastnosti, že všechny prvky mají hodnotu 0, s výjimkou prvků $e_{ij} \in \mathbb{E}$ u kterých se indexy i a j rovnají. Takové prvky mají hodnotu 1.

1.5 Paprsek

Paprsek je polopřímka, která má počátek a směr. V prostoru lze paprsek reprezentovat pomocí počátečního bodu a vektoru a můžeme ho zapsat parametrickou formou:

$$r(t) = o + t\vec{d} \ , \tag{1.1}$$

kde $r(t)$ je paprsek, o je počáteční bod, $\vec{d}$ je vektor udávající směr a parametr t

$$0 \leq t < \infty$$

udává výslednou délku tohoto paprsku [2].

Průsečík paprsku a trojúhelníku

Výpočet průsečíku paprsku s trojúhelníkem je esenciální znalostí této práce a existuje několik různých postupů, jak jej vypočítat. V této sekci si uvedeme výpočet pomocí běžně známých geometrických operací [10].

Pro výpočet průsečíku paprsku s trojúhelníkem potřebujeme provést dva kroky:

(i) vypočítat bod průniku paprsku s rovinou, ve které trojúhelník leží,

(ii) zjistit, jestli vypočítaný bod leží uvnitř trojúhelníku.

Pro začátek si uveďme obecnou rovnici roviny v prostoru:

$$ax + by + cz + d = 0 \ , \tag{1.2}$$

kde a, b, c jsou souřadnice normály $\vec{n} = (a, b, c)$ dané roviny a $x,\ y,\ z$ jsou souřadnice bodu ležícího v ní.

Předchozí rovnici také můžeme přepsat do tvaru využívající skalárního součinu:

$$\vec{n} \cdot \vec{x} = d \ , \tag{1.3}$$

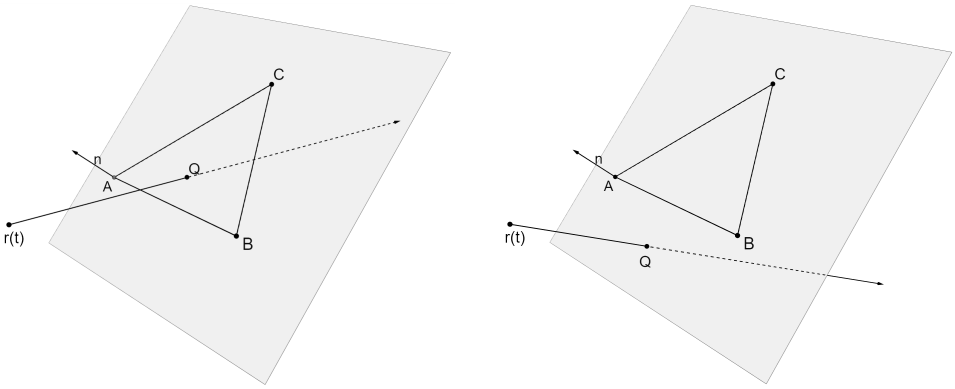
kde $\vec{x} = (x, y, z)$.

Pro výpočet průsečíku nám tak stačí nahradit vektor $\vec{x}$ z rovnice 1.3 za rovnici paprsku 1.1 a vyřešit tuto rovnici pro parametr t:

$$\begin{aligned} \vec{n} \cdot r(t) &= d \\ \vec{n} \cdot (\vec{o} + t\vec{d}) &= d \\ \vec{n} \cdot \vec{o} + t\vec{n} \cdot \vec{d} &= d \\ t &= \frac{d - \vec{n} \cdot \vec{o}}{\vec{n} \cdot \vec{d}} \ . \end{aligned}$$

Pokud je paprsek rovnoběžný s normálou $\vec{n}$, znamená to, že skalární součin $\vec{n} \cdot \vec{d} = 0$. Tím pádem hledaný průsečík neexistuje a rovnice nemá řešení. V případě, že tato rovnice řešení má, stačí vypočtený parametr t dosadit zpět do rovnice paprsku (1.1), čímž získáme hledaný bod průniku, který si označíme jako Q.

Mějme nyní trojúhelník ABC, ležící v rovině, pro kterou jsme vypočítali průsečík Q a chceme zjistit, zda-li tento průsečík leží uvnitř trojúhelníku. Dále mějme normálu $\vec{n}$, která určuje, zda-li je k nám trojúhelník natočen čelem. To můžeme zjistit tak, že prsty pravé ruky nasměrujeme od bodu A do bodu B a C, poté palec ukazuje ve směru normály $\vec{n}$.



(a) Paprsek prochází trojúhelníkem. (b) Paprsek neprochází trojúhelníkem.

Obrázek 1.1: Různé případy průniku paprsku s rovinou a trojúhelníkem ABC.

Aby bod Q ležel uvnitř trojúhelníku ABC, musí ležet nalevo od jednotlivých vektorů, určených hranami AB, BC a CA viz obrázek 1.1.

Ke zjištění, na jaké straně se bod Q nachází, použijeme vektorového součinu a výše zmíněné normály $\vec{n}$. Nejdříve vytvoříme vektor $\overrightarrow{AB} = B - A$ a vektor $\overrightarrow{AQ} = Q - A$.

Aby bod Q ležel na levé straně vektoru $\overrightarrow{AB}$ musí mít výsledný vektor vektorového součinu $\overrightarrow{AB} \times \overrightarrow{AQ}$ stejný směr, jako normála $\vec{n}$.

Z toho plyne, že skalární součin $(\overrightarrow{AB} \times \overrightarrow{AQ}) \cdot \vec{n}$ musí být větší, nebo roven nule. Aby se bod Q nacházel uvnitř trojúhelníku ABC, musí platit následující:

$$\begin{aligned}((\overrightarrow{B-A}) \times (\overrightarrow{Q-A})) \cdot \vec{n} \geq 0 \ , \\ ((\overrightarrow{C-B}) \times (\overrightarrow{Q-B})) \cdot \vec{n} \geq 0 \ , \\ ((\overrightarrow{A-C}) \times (\overrightarrow{Q-C})) \cdot \vec{n} \geq 0 \ .\end{aligned}$$

1.6 Směr a prostorový úhel

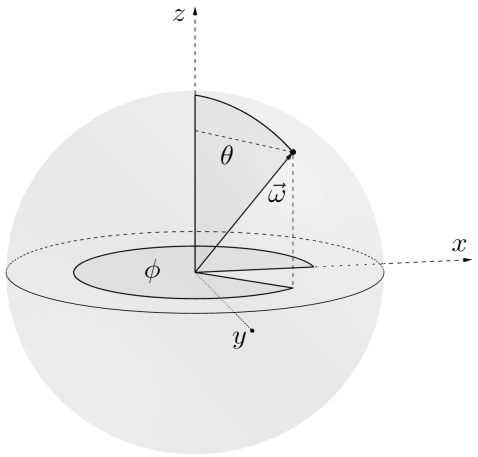
V počítačové grafice a zejména v oblasti renderování často pracujeme s prostorovým úhlem. Ten hraje důležitou roli zejména při výpočtu osvětlení, kde se setkáme se sférickými integrály.

Směr ve 3D

Směr, který často určujeme pomocí jednotkového vektoru, lze ve 3D popsat několika způsoby. Jedním z nich jsou sférické souřadnice, kde lze jednotkový vektor $\vec{\omega}$ vyjádřit jako bod na jednotkové kouli určený dvěma úhly: $\vec{\omega} = (\theta, \phi)$. Úhel $\theta \in [0, \pi]$ určuje odchylku od osy z (výška) a nazývá se *polární úhel*. Úhel $\phi \in [0, 2\pi]$ určuje odchylku od osy x a nazývá se *azimut* [11].

V případě libovolně velké koule o poloměru r lze jednoznačně určit polohu bodu na této kouli pomocí trojice sférických souřadnic (r, θ, ϕ). V případě převodu klasických kartézských souřadnic vektoru ω na tyto sférické a naopak, platí následující vzorce [12]:

$$\begin{aligned}\omega = (x, y, z) = (r, \theta, \phi) \ , \\ \theta = \arccos z \ , \\ \phi = \arctan \frac{y}{z} \ , \\ r = \sqrt{x^2 + y^2 + z^2} \ , \\ x = r \sin\theta \cos\phi \ , \\ y = r \sin\theta \sin\phi \ , \\ z = r \cos\theta \ .\end{aligned}$$

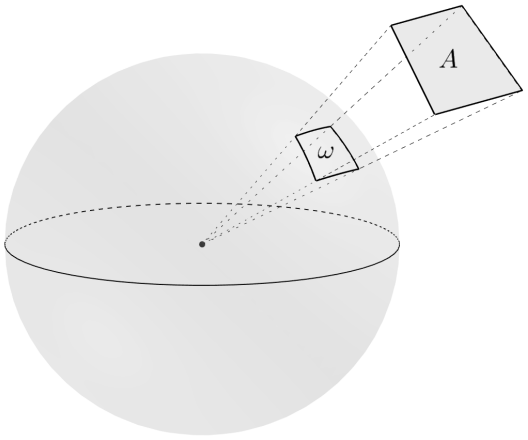


Obrázek 1.2: Ukázka vektoru vyjádřeného sférickými souřadnicemi

Prostorový úhel

Prostorový úhel je podobný koncept jako rovinný úhel, s tím rozdílem, že zatímco rovinný úhel udává délku oblouku na jednotkové kružnici, která má celkem 2π radiánů, prostorový úhel udává velikost plochy na jednotkové kouli. Jednotkou je *steradián* a jednotková koule má celkem 4π steradiánů [12].

Velikost prostorového úhlu ω vymezeného určitou plochou A, lze určit jako velikost promítnuté plochy A na jednotkovou kouli [12].



Obrázek 1.3: Ukázka prostorového úhlu

Často se setkáme s pojmem *diferenciální prostorový úhel*. Ten si můžeme představit jako nekonečně malý prostorový úhel $d\omega$ okolo nějakého směru $\vec{\omega}$.

Velikost úhlu $d\omega$ je dána velikostí diferenciální plošky dA na jednotkové kouli a směr úhlu $d\omega$ je určen jako střed projekce diferenciální plošky na jednotkovou kouli [12].

Pro výpočet prostorového úhlu určité diferenciální plošky dA platí [12]:

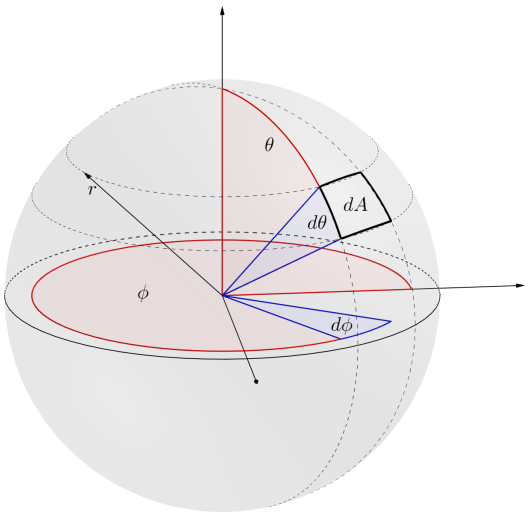
$$d\omega = dA\frac{\cos\theta}{r^2} \ ,$$

kde r značí vzdálenost od středu koule do středu plošky dA a θ je úhel mezi normálou plochy a směrem prostorového úhlu $d\omega$.

Diferenciální prostorový úhel se často používá k vyjádření směru a zaměňuje se za směrový vektor [11]. Jeho výpočet pak lze provést i pomocí následujícího vztahu [11]:

$$d\omega = \sin\theta\, d\theta\, d\phi \ ,$$
$$dA = r^2 \sin\theta\, d\theta\, d\phi \ .$$

Tento výpočet můžeme odvodit z obrázku 1.4 níže.



Obrázek 1.4: Ilustrace výpočtu prostorového úhlu a diferenciální plošky

Častokrát se setkáme s integrálem přes celou hemisféru. Tento integrál se v textu práce bude značit jako $\int_\Omega$, kde Ω je symbol pro hemisféru, neboli prostorový úhel pokrývající celou polokouli.

1.7 Monte Carlo integrování

Monte Carlo integrování je matematický postup pro odhad hodnoty integrálu funkce za použití náhodného vzrokování. Tato metoda je velmi často využívána v počítačové grafice, jelikož jí můžeme nahradit složité výpočty integrálů, které jsou mnohdy nemožné, za mnohem rychlejší a postačující odhad jejich hodnoty. Počet použitých vzorků ovlivňuje přesnost odhadu a pokud se jejich počet limitně blíží k nekonečnu, chyba odhadu konverguje k nule [13].

Odhadovanou hodnotu integrálu I funkce $f : \Omega \to \mathbb{R}$:

$$I = \int_{\Omega} f(x)dx \ ,$$

můžeme pomocí Monte Carlo metody vyjádřit následovně:

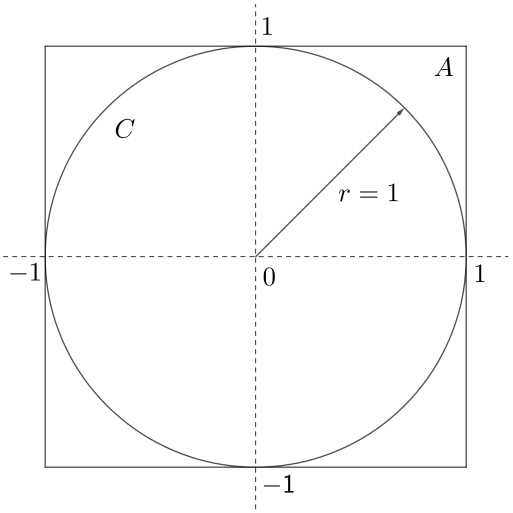
$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)} \ ,$$

kde F_N nazveme jako estimátor integrálu I, X_i je náhodná veličina s hustotou pravděpodobnosti $p(X_i)$ a N udává počet použitých vzorků [14].

Vzorkování podle důležitosti

Použijeme-li non-uniformní rozdělení pravděpodobností a zvolíme hustotu pravděpodobnosti $p(X)$, která nejlépe odpovídá rozptylu hodnot integrandu, dosáhneme tím efektivnější aproximace. Tomuto postupu se říká vzorkování podle důležitostí (*importance sampling*). Ten můžeme chápat tak, že některé náhodně generované vzorky, zejména ty kolem střední hodnoty, mají největší vliv na odhad výsledku. Importance sampling má za úkol náhodně generovat takové hodnoty, které nejvíce ovlivní výsledný odhad, díky čemuž rychleji konverguje ke správnému výsledku za použití menšího počtu vzorků [13].

Pro lepší pochopení Monte Carlo metody si ukážeme, jak pomocí ní odhadnout hodnotu Ludolfova čísla π [13]. Mějme čtverec A s délkou strany $l = 2$ a kruh C s poloměrem $r = 1$ viz obrázek 1.5.



Obrázek 1.5: Ilustrace jednotkové kružnice uvnitř čtverce

Z obrázku výše si můžeme odvodit, že pro obsah kruhu platí $S_C = \pi$ a pro obsah čtverce $S_A = 4$.

Mějme nyní funkci $f(x, y)$, pro kterou platí:

$$f(x,y) = \begin{cases} 1 & \text{pokud } x^2 + y^2 \leq 1 \\ 0 & \text{pokud } x^2 + y^2 > 1 \end{cases}$$

a dále mějme rovnoměrné rozdělení $p(x, y)$ na intervalu $\langle -1, 1 \rangle^2$:

$$p(x,y) = \begin{cases} \frac{1}{4} & \text{pokud } |x|, |y| \leq 1 \\ 0 & \text{pokud } |x|, |y| > 1 \end{cases}$$

Nyní můžeme obsah kruhu S_C vyjádřit následovně:

$$S_C = \pi = \int \int f(x,y)p(x,y)dxdy \ .$$

Po aplikaci metody Monte Carlo na výše uvedený integrál dostáváme:

$$\pi \approx \frac{4}{N} \sum_{i=1}^{N} f(x,y) \ .$$

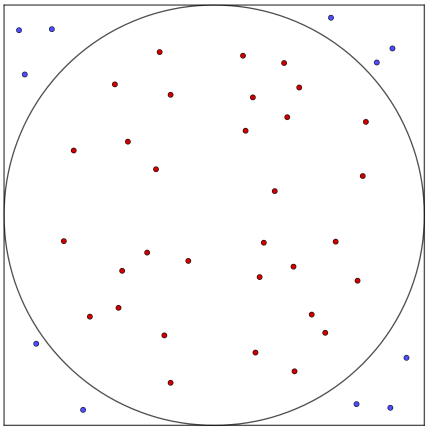
Výše uvedené si můžeme ukázat na příkladu. Budeme-li náhodně generovat body uvnitř uvedeného čtverce A, můžeme vyjádřit pravděpodobnost $P(x_i)$ toho, že tento bod bude vně kruhu C, jako poměr mezi obsahy S_C a S_A [15]:

$$P(x_i) = \frac{S_C}{S_A} = \frac{\pi}{4} \ .$$

z toho plyne, že π můžeme vyjádřit jako:

$$\pi = 4P(x_i) \ .$$

Vygenerujeme-li nyní velký počet bodů, můžeme pravděpodobnost $P(x_i)$ odhadnout jako poměr mezi počtem bodů, které leží uvnitř kruhu a celkovým počtem bodů. Následně stačí tuto hodnotu vynásobit čtyřmi a máme odhad čísla π. Zároveň platí, že čím větší počet bodů vygenerujeme, tím lepší tento odhad bude.



Obrázek 1.6: Ukázka náhodného generování bodů

Kapitola 2

Scéna

Abychom byli schopni vytvářet virtuální světy, musíme nejdříve najít způsob, jak reprezentovat jednotlivé objekty a simulovat tak náš reálný svět. V počítačové grafice se však spíše než „svět“, používá pojem scéna.

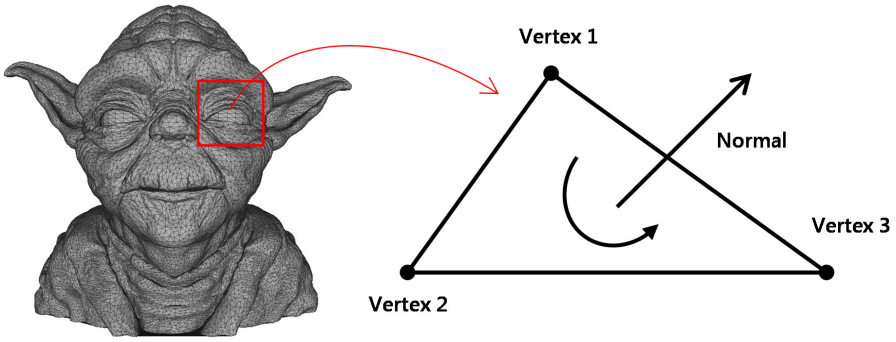
Scéna v sobě uchovává informace o všech objektech, které se v ní nacházejí. Zejména se jedná o jednotlivé modely, které reprezentují hmotu reálného světa, světelné zdroje, které si popíšeme v kapitole 4 a kameru, která simuluje pohled člověka na všechny tyto objekty. Scéna může dále obsahovat určité logické struktury usnadňující práci s objekty a informace o transformacích těchto objektů.

Některé objekty se ve scéně nezobrazují a slouží pouze jako logická reprezentace, jedná se zejména o kamery a některé typy světelných zdrojů. Kromě toho může scéna obsahovat i mnoho dalších entit, které však v rámci této práce nebudeme uvažovat.

2.1 Model

Modelem je v počítačové grafice myšlena nějaká matematická reprezentace 3D objektu. Nejčastěji se pak používá reprezentace pomocí sítě trojúhelníkových polygonů tzn. *hraniční reprezentace* [11], která tento objekt popisuje viz. obrázek 2.1. Každý bod tohoto trojúhelníku je nazýván vertex a nese informace o jeho umístění v souřadnicovém systému samotného modelu. Kromě polohy může vertex obsahovat i jiné informace, nejčastěji se jedná o normálu vzhledem k ploše trojúhelníku, jehož je součástí, nebo texturovací souřadnice.

Model může nést dodatečné informace. Nejčastěji se jedná o informace o jeho materiálu, který se následně používá při aplikaci lokálního osvětlovacího modelu, nebo o jeho transformaci vzhledem k souřadnicovému systému scény.

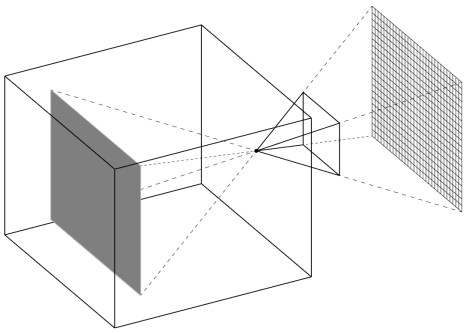


Obrázek 2.1: Ukázka 3D modelu reprezentovaného polygonovou sítí a trojúhelníku sestaveného z vertexů. [16]

2.2 Kamera

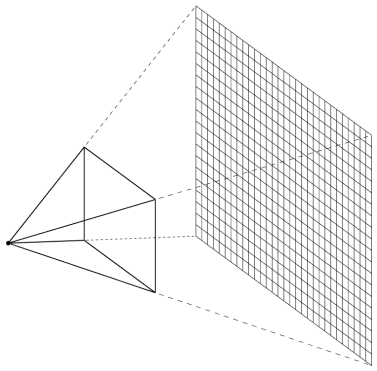
Stejně tak jako v reálném světě používáme kameru k zachycení obrazu, využíváme obdobného fenoménu i v počítačové grafice. V tomto případě je ale kamera uměle simulována a princip jejího fungování je značně zjednodušen. Tato „umělá“ kamera vychází z principu dírkové kamery, známé z angličtiny jako *pinhole camera* [2].

Dírková kamera se skládá ze světelně nepropustné schránky ve tvaru kvádru s malou dírkou ve středu jedné z jejích stěn. Pokud je tato dírka odkryta, světlo skrze ni proniká a dopadá na fotografický papír(film), umístěný na protější stěně, čímž po nějaké době vzniká obraz. Této době se odborně říká expoziční čas a v případě dírkové kamery je potřeba, aby byl velmi dlouhý pro zachycení adekvátního množství světla dopadajícího na film [2]. Ačkoliv je většina kamer v dnešní době mnohem komplexnějších, je tento model dostatečný pro použití v počítačové grafice.



Obrázek 2.2: Dírková kamera

Pro zjednodušení můžeme uvažovat, že film umístíme před dírku ve stejné vzdálenosti, ve které byl umístěna za ní. Tímto si zjednodušíme práci, jelikož nebudeme muset obraz zrcadlit a zároveň nám tato abstrakce postačí. V takovémto případě dírka představuje pozici, ze které je pohlíženo na scénu a místo termínu dírka se používá oko a nebo kamera [2].



Obrázek 2.3: Simulace dírkové kamery v počítačové grafice

Některé softwary dokáží simulovat i klasické moderní kamery spolu s celým optickým aparátem. Díky tomu umožňují vytvářet jevy, se kterými se setkáme při běžném fotografování, jako je například hloubka ostrosti.

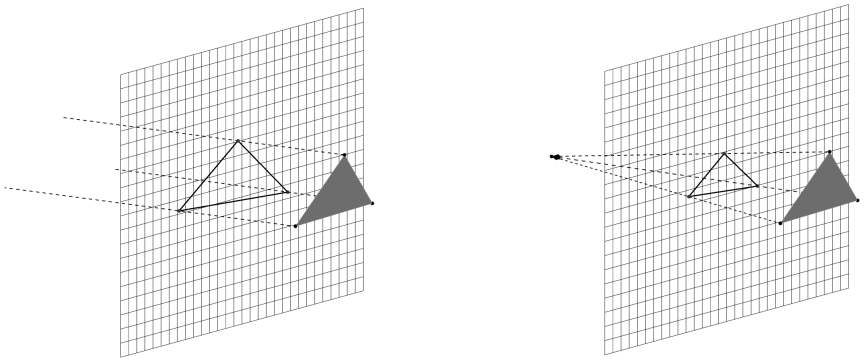
2.3 Projekce

Jak J.Žára vysvětluje ve své knize *Moderní počítačová grafika* [11], projekce, nebo v některých zdrojích promítání, je způsob, jak transformovat trojrozměrný objekt nebo scénu do její výsledné dvourozměrné reprezentace. Nejprve si uvedeme dva základní pojmy [11]:

- *promítací paprsek* – polopřímka vycházející z bodu v prostoru,
- *průmětna* – plocha v prostoru, na kterou dopadají promítací paprsky a tím vytvářejí obraz.

V počítačové grafice se používá hlavně promítání do rovinné průmětny, které se dělí na dva základní typy [11]:

- *ortografické (rovnoběžné)*– Všechny body v prostoru se na průmětnu přenášejí pomocí přímek, které jsou kolmé na rovinu průmětny. Všechny tyto přímky jsou vzájemně rovnoběžné. Tohoto promítání se využívá především v CAD systémech, neboť zachovává původní rozměry.
- *perspektivní (středové)* – zde se body do průmětny přenášejí také pomocí přímek, ale všechny tyto přímky procházejí jedním bodem, kterému se říká středový. Tento druh promítání se používá právě při renderování, neboť dokáže vytvořit perspektivu, která je běžná i v reálném světě.



(a) Ortografické promítání (b) Perspektivní promítání

Obrázek 2.4: Ukázka ortografického a perspektivního promítání

2.4 Datové struktury pro reprezentaci scény

Často se setkáme s tím, že potřebujeme provádět určité operace s objekty nacházejícími se ve scéně. Jednou z těchto operací je například hledání průniků mezi objektem a paprskem. Jelikož ale předem nevíme, které objekty se nacházejí v cestě paprsku, musíme hledat průsečík pro každý objekt. To je ale velmi neefektivní a časově náročné, zejména pokud scéna obsahuje velké množství objektů.

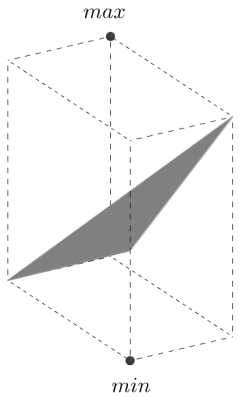
Z tohoto důvodu se používají datové struktury, které si kladou za cíl vhodně uspořádat informace o prostoru scény a snížit tak časovou náročnost při jejím procházení. Tyto struktury mají velmi široké použití a často se s nimi setkáme v metodách sledování paprsku, nebo při detekci kolizí [11].

Tyto struktury dělíme do dvou skupin, které nazýváme jako *hierarchie obálek (BVH – Bouding Volume Hieararchies)* a *hierarchie dělení prostoru (Space Subdivision Hieararchies)*. Jedná se o stromové struktury, kde kořen stromu představuje celou scénu a listy ve většině případů představují jednotlivé objekty [11].

2.4.1 Hierarchie obálek

Hierarchie obálek je metoda, při níž je objekt ohraničen tělesem s velmi jednoduchou geometrií. Tato metoda je jeden z nejzákladnějších způsobů, jak zrychlit vyhledávání průsečíků s objekty.

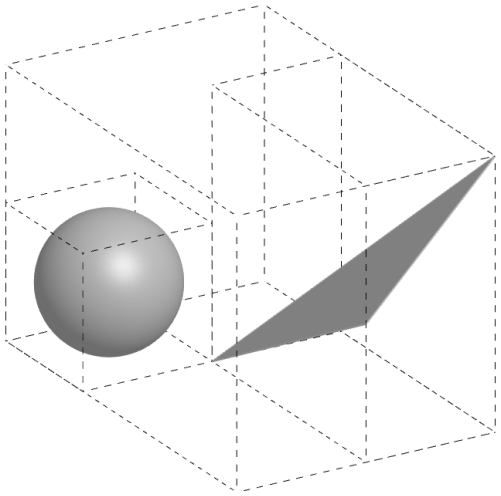
Existuje několik způsobů, jak tělesa ohraničit, kde mezi nejzákladnější patří koule, osově zarovnaný kvádr, nebo orientovaný kvádr. Pro nás důležitou obálkou je především osově orientovaný kvádr. S tím se můžeme často setkat pod názvem *AABB (Axis-Aligned Bounding Box)* a má tu vlastnost, že všechny jeho stěny jsou kolmé na souřadnicové osy [11]. Díky tomu je lze reprezentovat pouze dvěma body, často nazývané jako *min* a *max*. Příklad této obálky můžeme vidět na obrázku 2.5.



Obrázek 2.5: Ukázka obálky typu AABB

Z takovýchto obálek je možné sestavit binární strom zvaný *BVH*, kde kořen představuje obálku nad všemi objekty ve scéně. Jeho potomci následně obsahují tuto obálku rozdělenou na dvě části, kde rozdělování probíhá podle určité heuristiky. Často se můžeme setkat například s dělením podle osy, na které je obálka největší.

Pokud takto rozdělené obálky obsahují pouze jeden objekt, nebo je již nelze dále dělit, dělení končí a vzniká nový list stromu. V opačném případě se v dělení pokračuje. Z toho plyne, že vnitřní uzel stromu obsahuje vždy pouze obálku ohraničující všechny objekty obsažené v jeho podstromech a objekty samotné jsou uloženy v listech stromu [11]. Ohraničení kolem více objektů pak může vypadat například jako na obrázku 2.6.



Obrázek 2.6: Ukázka AABB obálky kolem více objektů

2.4.2 Dělení prostoru

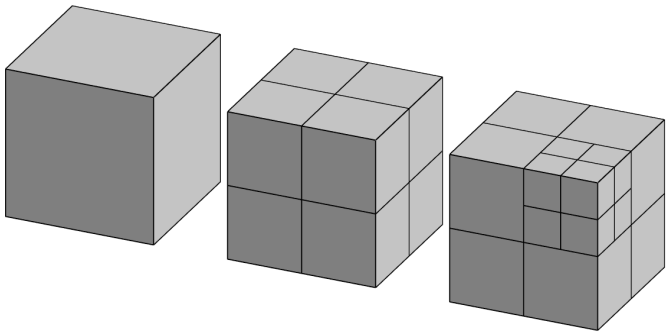
Dělením prostoru scény je myšlena reprezentace scény pomocí datové struktury, která rekurzivně rozděluje scénu do disjunktních oblastí pomocí rovinných řezů. Tyto datové struktury jsou reprezentovány pomocí stromů, kde do vnitřních uzlů zaznamenáváme informace o dělení prostoru (orientaci a polohu řezu) a listy těchto stromů odkazují na objekty, které se v nich alespoň z části nacházejí. Existuje mnoho různých řešení, z nichž si můžeme například jmenovat oktalový strom (*octree*), BSP strom (*Binary Space Partitioning tree*) a kD-strom (*kD-tree*) [11].

Oktalový strom

Jedná se o strom, ve kterém má každý uzel přesně 8 potomků, vyjma listů. Využívá se k rekurzivnímu dělení 3D prostoru na 8 menších podprostorů o stejné velikosti, kde je řez prostoru proveden ve všech osách a střed řezu se nachází uprostřed daného prostoru [11].

Dělení prostoru je ideálně prováděno, dokud každý bod neleží ve vlastním podprostoru, ale může být určeno ukončovacími podmínkami, jako například maximální hloubka rekurze, počet bodů v podprostoru, nebo minimální velikost podprostoru. Tato kritéria ale mohou zpomalit následné vyhledávání v takto vytvořeném stromu.

S oktalovými stromy a jejich využitím ve 3D grafice poprvé přišel D. Meagher v 80. letech 20. století [17], přičemž se inspiroval strukturou zvanou *quadtree*, která funguje na velmi podobném principu, ale ve 2D.



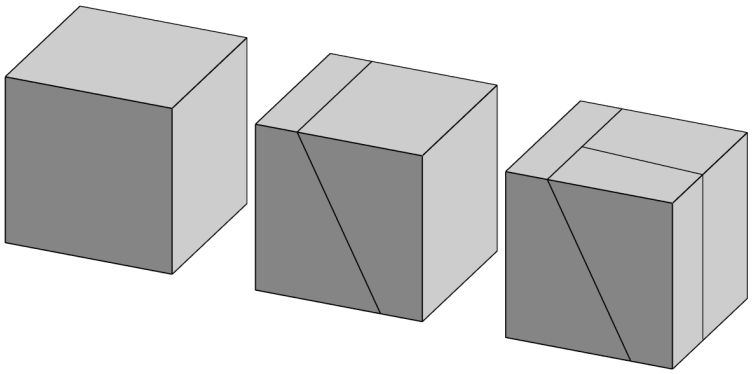
Obrázek 2.7: Ukázka dělení prostoru podle oktalového stromu

BSP strom

Jedná se o strukturu reprezentovanou binárním stromem, kde se prostor vždy rozděluje na dva podprostory (poloprostory). Tato struktura se vyznačuje tím, že lze použít pro libovolný n-dimenzionální prostor.

Výběr místa řezu se může řídit různými pravidly, jako například dělení prostoru na poloviny, nebo dělení podle počtu bodů, kde je nejprve vybrán medián podle souřadnic a následně se s pomocí něj dělí prostor na dvě podobně velké skupiny [11].

Stejně jako u oktalových stromů, i zde se využívá stejných podmínek pro ukončení dělení prostoru. Oproti oktalovým stromům mají BSP stromy tu výhodu, že jsou častěji lépe vybalancované a nemusejí vznikat prázdné buňky. Tím se snižuje čas nutný k jeho průchodu [11].

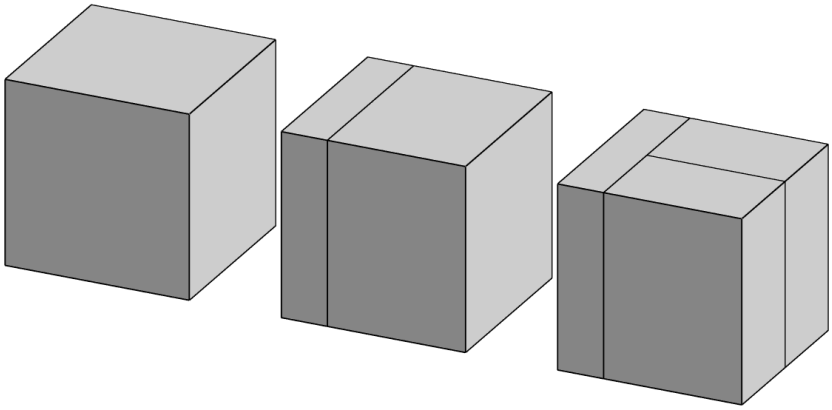


Obrázek 2.8: Ukázka dělení prostoru podle BSP stromu

kD-strom

Další strukturou je kD-strom. Ten je speciálním případem BSP stromu a vyznačuje se způsobem, jakým dělí prostor.

Bod dělení je obvykle určen mediánem souřadnic bodů v určité ose, kde se následně provede řez za pomocí roviny, která je rovnoběžná s touto osou. Osy, podle kterých se provádí řez prostoru, se pravidelně střídají. Tímto způsobem se dosahuje vyváženého stromu [11].



Obrázek 2.9: Ukázka dělení prostoru podle kD stromu

Kapitola 3

Renderování

Proces generování obrazu z 2D nebo 3D modelu nazýváme z hlediska počítačové grafiky jako renderování. V praxi existují dva přístupy k němu. První z nich se pokouší o vytváření fotorealistických obrazů za pomoci simulace reálných fyzikálních fenoménů týkajících se optiky. Algoritmy spadající do této kategorie se často hromadně nazývají jako metody globálního osvětlení (*global illumination methodes*) a vyznačují se především komplexitou a náročností na výpočet.

Druhý přístup pak využívá jen velmi malou část optiky a není sám o sobě schopný vytvářet složitější efekty jako například stíny objektů, či vliv objektů na okolí. Tento postup je standardně označován jako rasterizace, nebo realtime renderování a pro rychlost používaných zobrazovacích metod a speciálně určenému hardwaru se využívá především (zejména) v segmentu počítačových her.

Tato práce se zabývá první z těchto dvou metod a následující kapitoly slouží k lepšímu porozumění teorie použité v algoritmech řešících problémy globálního osvětlení.

3.1 Problém viditelnosti

Výše popsané postupy se neliší pouze v tom, do jaké míry využívají fenomény popsané optikou, ale i tím, jak řeší problém viditelnosti. Algoritmy zabývající se problémem viditelnosti mají za úkol zjistit, které objekty ve scéně jsou pozorovatelné z určitého místa.

Obě metody shodně používají rastrovou reprezentaci výsledného obrazu tzn. že obraz je složen z jednotlivých pixelů popisujících barvu odpovídajícího objektu, který je viditelný z místa pozorování.

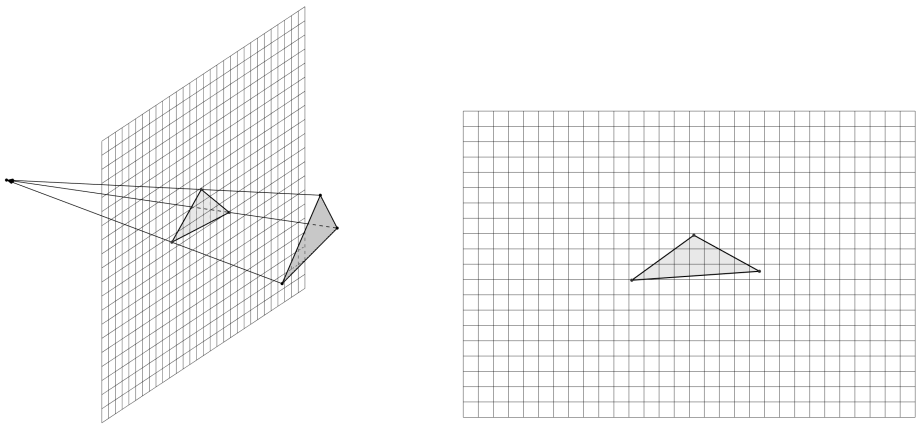
Rasterizace

Zobrazování scény pomocí rasterizace trojúhelníků je nejpoužívanější metoda pro real-time renderování (renderování v reálném čase) a využívá se převážně v herních enginech. Tato metoda se vyznačuje svojí rychlostí a tím, že ji lze akcelerovat pomocí grafické karty, která je uzpůsobena k řešení podobných problémů. Její rychlost ale přináší jistá negativa, zejména z hlediska věrohodné simulace reality, jelikož například generování stínů nebo odrazů je poměrně složité.

Geometrická data objektu, tedy jednotlivé vertexy, jsou nejdříve nahrány do pole, které je uloženo v paměti grafické karty. Algoritmus následně iteruje přes jednotlivé vertexy v tomto poli a pomocí transformací a projekce převádí souřadnice těchto bodů v lokálním souřadnicovém systému až do výsledného souřadnicového systému samotného obrazu (*screen space*). Takto transformované vertexy jsou následně převedeny do jednotlivých pixelů. V této fázi lze ovlivnit výslednou barvu těchto pixelů, a co je pro nás důležitější, určuje se, který objekt je viditelný a bude zobrazen ve výsledném renderu [18].

K tomu slouží tzv. *z-buffer*. Ten si pro každý pixel pamatuje, jak daleko od místa pozorování se nachází bod, který byl transformován do souřadnic daného pixelu. Může se totiž stát, že jednomu pixelu odpovídá více objektů, to však znamená, že jsou objekty v zákrytu. Pokud k tomuto dojde, do z-bufferu se vždy ukládá nejmenší z těchto hodnot a výsledný pixel tak náleží objektu nacházejícímu se nejblíže místu pozorování. Tím je tak určena viditelnost jednotlivých objektů a podoba výsledného obrazu [18].

Tento popis je velmi stručný a slouží jen k představě toho, jak tento postup funguje. Pro lepší pochopení těchto algoritmů a pro více informací doporučuji vyhledat jiné zdroje.



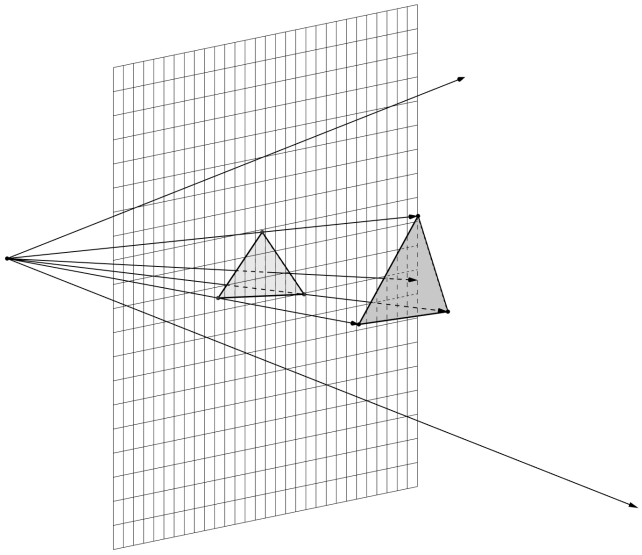
(a) Projekce (b) Promítnutý objekt

Obrázek 3.1: Rasterizace

Ray casting

Ray casting je pro změnu nejpoužívanější metodou pro zjišťování viditelnosti u algoritmů globálního osvětlení se kterou přišel na počátku 80. let 20. století Scott D. Roth [19].

Ray casting funguje na principu vystřelování paprsků z místa pozorování skrze jednotlivé pixely obrazu. Tyto paprsky jsou následně sledovány a je hledán jejich průsečík s objekty ve scéně. Pokud je nějaký průsečík s objektem nalezen a jedná se o první průsečík paprsku při jeho cestě skrze scénu, tzn. jeho vzdálenost od místa pozorování je ze všech ostatních průsečíků daného paprsku nejmenší, znamená to, že pro pixel, skrze který byl paprsek vystřelen, bude tento objekt viditelný a jeho barva bude odpovídat barvě objektu v místě průniku.



Obrázek 3.2: Ray casting

3.2 Shading

V případě, že jsme zjistili viditelnost objektů, ještě musíme vyřešit další otázku, a to sice jak vybarvit jednotlivé pixely obrazu. Této fázi se nejčastěji říká shading a jedná se o velmi rozsáhlé téma a existuje mnoho možných řešení. Zjednodušeně řečeno, shading má na starost výsledný vzhled viditelných objektů. Tento vzhled záleží na mnoha faktorech, jako je například osvětlení scény, úhel, ze kterého jsou pozorovány, nebo jejich materiál. Co dává výslednému obrazu jeho realističnost, je schopnost správně simulovat světlo a jeho vliv na celou scénu (globální osvětlení) společně se simulací materiálů. Právě tomuto se věnují následující kapitoly.

Kapitola 4

Světlo a světelné zdroje

Značná část počítačové grafiky se věnuje tomu, jak co možná nejvěrněji simulovat světlo a jeho putování prostorem, ve kterém interaguje s povrchy objektů a opticky aktivním prostředím. Pro tvorbu virtuálních scén a realistickou syntézu obrazu je pochopení těchto jevů základním předpokladem. Většina algoritmů, která se snaží chování světla napodobit, je založena na fyzikálních jevech, se kterými se seznámíme v této části textu.

Jelikož má světlo dualistickou povahu, tzn. že se chová jako vlna, ale i jako částice, se nauka o světle (Optika) rozděluje do několika podoblastí [11]:

- *Geometrická optika* – pracuje se světlem jako s nezávislými paprsky, které putují prostorem a lze je popsat pomocí geometrie.
- *Vlnová optika* – respektuje vlnové vlastnosti světla a umožňuje popsat jevy, které nelze popsat pomocí geometrické optiky. zejména se jedná o difrakci a interferenci.
- *Elektromagnetická optika* – uplatňuje Maxwellovy rovnice. Tím oproti vlnové optice dokáže popsat polarizaci světla a jeho disperzi na hranách objektů. Jedná se o nejvěrnější fyzikální model světla v rámci klasické fyziky.
- *Kvantová optika* – zohledňuje jevy, které nelze popsat klasickou fyzikou a používá přístup kvantové fyziky. Je základem pro vysvětlení interakce světla s materiálem.

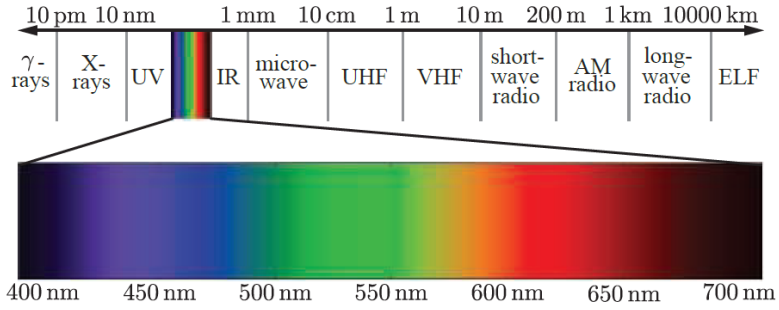
Vlnový popis světla je vhodný pro složitější jevy jako disperze, difrakce nebo interference, zatímco částicovým popisem můžeme vysvětlit odraz světla, nebo interakci s drsnými povrchy. V počítačové grafice se v převážné většině používá geometrická optika [11].

Při simulaci světla se používají určitá zjednodušení [11]:

- Světlo se šíří přímočaře.
- Rychlost šíření světla je nekonečná.

4.1 Fyzikální vlastnosti

Světlo je elektromagnetické záření, přesněji řečeno jeho viditelná část, která má vlnovou délku mezi 390-760 nm [18].



Obrázek 4.1: Elektromagnetické spektrum podle vlnové délky [20]

V souvislosti se světlem se také často zmiňuje pojem *foton*. Foton je elementární částice světla, která vyjadřuje nejmenší kvantum (množství) záření. Můžeme si ho představit jako nejmenší možný „balíček energie“ elektromagnetického záření o určité vlnové délce. Foton se nejčastěji značí jako γ, jednotkou je joule J a jeho energii lze vyjádřit jako:

$$E = \frac{hc}{\lambda} \ ,$$

kde $h \approx 6,6 \cdot 10^{-34} J \cdot s$ značí Planckovu konstantu, $c \approx 3 \cdot 10^{8} m \cdot s^{-1}$ je rychlost světla a λ je vlnová délka elektromagnetického záření [11].

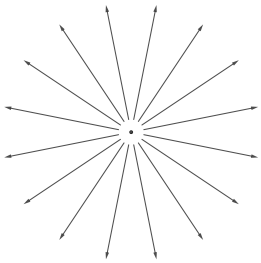
Einstein byl první, kdo označil světlo, jako tok fotonů a vysvětlil jeho dualistickou povahu [21].

4.2 Typy světelných zdrojů

Abychom ve scéně něco viděli, musí v ní být přítomen světelný zdroj tak, jako tomu je i v reálném světě. V počítačové grafice simulujeme světelné zdroje několika různými způsoby a s těmi nejpoužívanějšími se seznámíme v této části.

Bodový zdroj světla

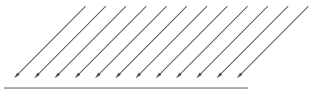
Asi nejběžnějším zdrojem světla je v počítačové grafice tzv. *bodový zdroj světla*. Jak název napovídá, jedná se o bod, který rovnoměrně a se stejnou intenzitou vyzařuje světlo do všech směrů. Tento zdroj je určen intenzitou a polohou v souřadnicovém systému scény [18].



Obrázek 4.2: Bodový zdroj světla

Směrový světelný zdroj

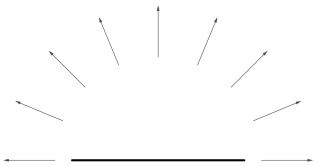
Někdy také nazýván jako rovnoběžný světelný zdroj. Tento zdroj světla můžeme chápat jako nekonečně velký rovinný zdroj umístěný v konečné vzdálenosti od objektů, nebo jako bodový zdroj umístěný nekonečně daleko. Tyto dvě představy však mají stejný význam a to ten, že všechny světelné paprsky vyzářené tímto zdrojem dopadají pod stejným úhlem, tzn. cestují ze stejného směru. Tento světelný zdroj nejčastěji simuluje velmi vzdálené globální světelné zdroje, například slunce [11].



Obrázek 4.3: Směrový zdroj světla

Plošný světelný zdroj

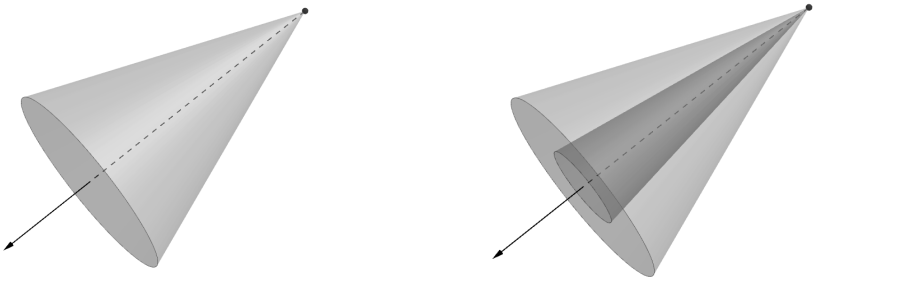
Plošný světelný zdroj je obyčejná konečná plocha, která vyzařuje světelné paprsky z každého bodu na ploše a každým směrem, omezeným polokoulí nad touto plochou. Tento zdroj se vyznačuje tím, že vytváří tzv. polostín. To znamená, že stín nemá ostrou hranici. Toho se často využívá například u produktových renderů a simulací interiérových světelných zdrojů.



Obrázek 4.4: Plošný zdroj světla

Reflektor

Reflektor můžeme přirovnat k bodovému světelnému zdroji s tím rozdílem, že vyzařuje světlo pouze určitým směrem. Tento světelný zdroj vyzařuje nejvíce světla ve směru osy vyzařování a intenzita světla klesá exponenciálně vzhledem ke vzdálenosti kolmé na tento směr. Reflektory však mohou mít oblast kolem osy záření, kde intenzita klesá pouze se vzdáleností od zdroje. Ve zbylé oblasti pak klesá intenzita i vůči vzdálenosti od osy vyzařování [11].



(a) Obyčejný reflektor (b) Reflektor s jasným středovým svazkem

Obrázek 4.5: Reflektor

4.3 Radiometrie

Tato část nás seznámí s radiometrií, která zavádí veličiny kvantifikující světlo. Narozdíl od *fotometrie*, která popisuje vnímání světla vzhledem k lidskému zraku, je radiometrie na člověku nezávislá a tím je objektivnější. Fotometrické veličiny lze odvodit z těch radiometrických při použití viditelného spektra elektromagnetického záření[12].

Definice radiometrických veličin využívají základního diferenciálního počtu, jehož alespoň částečná znalost je potřebná pro lepší pochopení výrazů, se kterými se zanedlouho setkáme. Připomeneme si, že derivace se značí následovně:

$$\frac{df}{dx} \ ,$$

kde f značí funkci, kterou derivujeme podle proměnné x. Aneb, jak nekonečně malá změna v hodnotě x změní funkční hodnotu $f(x)$.

V této sekci se setkáme s diferenciálním počtem ve vztahu k prostorovému úhlu, ploše a času. Tyto veličiny $d\theta$, dA a dt tak vyjadřují nekonečně malé kvantity těchto hodnot.

Tato část textu vychází z knížky *Moderní počítačová grafika*[11] a z knížky *Computer Graphics: Principles and Practice* [18].

Zářivá energie

Zářivá energie (*radiant energy*) udává vyzářenou, nebo pohlcenou energii elektromagnetického záření za celou dobu pozorování. Nejčastěji se značí Q_e a její jednotkou je joul J. Můžeme o ní hovořit jako o součtu energií všech fotonů a lze ji vyjádřit jako integrální součet:

$$Q_e = \int_0^\infty n_\lambda e_\lambda d_\lambda \ ,$$

kde n_λ vyjadřuje počet fotonů vlnové délky λ a e_λ vyjadřuje energii fotonu o vlnové délce λ.

Pro lepší pochopení si můžeme představit senzor, na který dopadají jednotlivé fotony. Zářivá energie vyjadřuje součet energií všech těchto fotonů za celou dobu pozorování.

Zářivý tok

Zářivý tok, také zvaný jako zářivý výkon (*flux*), značený Φ_e, je zářivá energie, která je vyzářena nebo pohlcena za jednotku času t. Její jednotkou je $J \cdot s^{-1}$, neboli watt W a je definována následovně:

$$\Phi_e = \frac{dQ_e}{dt} \ .$$

Navážeme-li na předchozí příklad, zářivý tok vyjadřuje energii, která v průměru za daný časový úsek t dopadla na tento senzor. Z tohoto důvodu lze energii také vyjádřit pomocí integrálu:

$$Q_e = \int_t \Phi_e(t) dt \ .$$

Ozáření a Radiozita

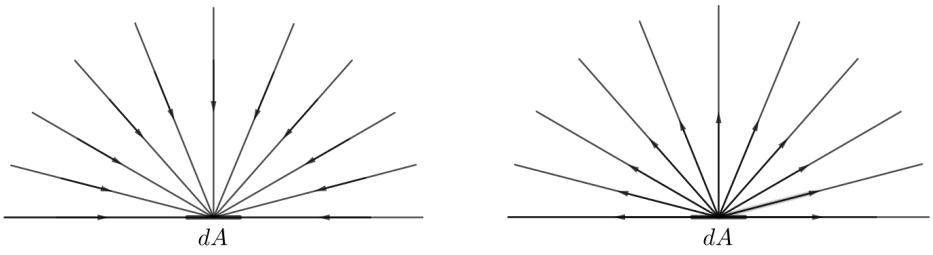
Ozáření (*irradiance*) a radiozita (*radiosity*), udává množství zářivého toku dopadajícího, respektive vyzářeného na jednotku plochy a značí se E_e a M_e. Jednotkou je $W \cdot m^{-2}$ a pro ozáření platí:

$$E_e = \frac{d\Phi_e}{dA} \ , \tag{4.1}$$

kde A je ozářená plocha a pro radiozitu platí:

$$M_e = \frac{d\Phi_e}{dA} \ , \tag{4.2}$$

kde A je plocha vyzařující.



(a) Ozáření (b) Radiozita

Obrázek 4.6: Ozáření a radiozita

Budeme-li pokračovat v našem příkladu, můžeme si ozáření představit jako průměrný zářivý tok přes celou plochu tohoto senzoru, nebo-li hustotu zářivého toku dopadajícího na tento senzor. Zářivý tok tak lze také vyjádřit jako integrál:

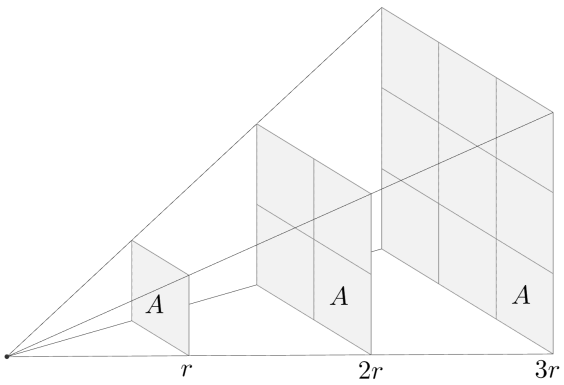
$$\Phi_e = \int_A E_e(p)dt \ .$$

Zákon převrácených čtverců

Další vlastnost ozáření a radiozity si ukážeme na příkladu. Řekněme, že nás zajímá, jakou hodnotu ozáření má koule, v jejímž středu se nachází bodový zdroj světla. Víme, že plocha koule se určí jako $4\pi r^2$ a po dosazení dostaneme:

$$E_e = \frac{\Phi_e}{4\pi r^2} \ .$$

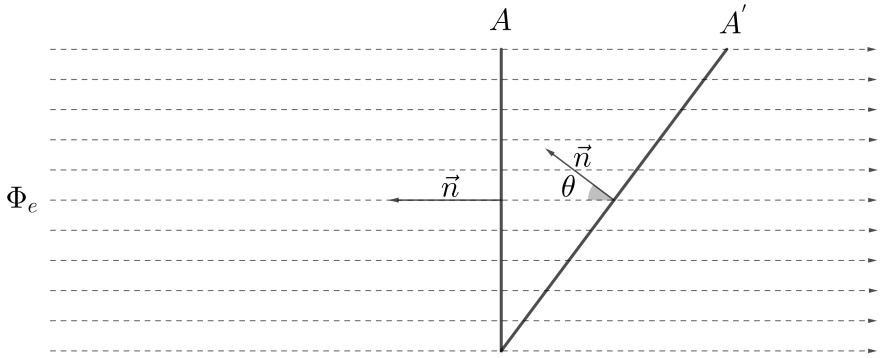
Z toho plyne, že čím větší je průměr této koule, tím menší je hodnota ozáření. Obecněji řečeno, množství ozáření určité plochy A klesá s druhou mocninou vzdálenosti r od světelného zdroje. Tomuto faktu se říká zákon převrácených čtverců (*inverse square law*).



Obrázek 4.7: Zákon převrácených čtverců

Lambertův kosínový zákon

Již jsme si řekli, že množství ozáření klesá s druhou mocninou vzdálenosti od světelného zdroje. Nyní si představme, že máme svazek světelných paprsků čtvercového průřezu, viz obrázek 4.8.



Obrázek 4.8: Lambertův kosínový zákon

Můžeme vidět, že na obrázku máme dvě různě velké plochy A a $A^{'}$, přičemž na obě dopadá stejné množství záření. Plocha A je pak orientována kolmo ke směru paprsků, zatímco plocha $A^{'}$ svírá úhel θ mezi normálou plochy a směrem paprsků. Jelikož je plocha $A^{'}$ větší, ale dopadá na ni stejné množství záření, má ozáření (hustotu záření) menší, než plocha A. Velikost plochy $A^{'}$ můžeme získat pomocí vzorce:

$$A^{'} = \frac{A}{\cos\theta} \ .$$

Tomuto jevu se říká *Lambertův kosínový zákon*. Ten nám říká, že ozáření, respektive radiance, záleží na úhlu, pod kterým záření dopadá nebo je vyzářeno. Pro plochu $A^{'}$ pak platí:

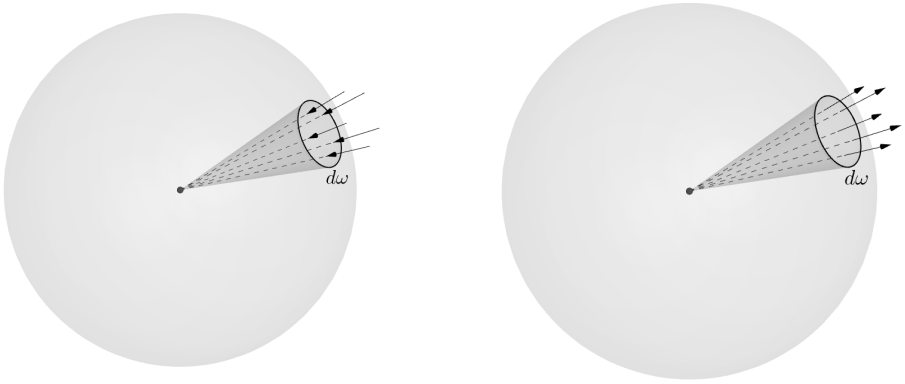
$$E_e = \frac{d\Phi_e}{dA^{'}} = \frac{d\Phi_e}{dA}\cos\theta \ .$$

Zářivost

Zářivost (*radiant intensity*) je světelný tok procházející prostorovým úhlem. Značí se I_e, její jednotka je $W \cdot sr^{-1}$ a platí:

$$I_e = \frac{d\Phi_e}{d\omega} \ ,$$

kde dw značí diferenciální prostorový úhel.



(a) Zářivost příchozí (b) Zářivost odchozí

Obrázek 4.9: Zářivost

Toto schéma si také můžeme představit jako bodový světelný zdroj, přičemž nás zajímá výkon tohoto zdroje omezený výsečí určenou prostorovým úhlem.

Máme-li plochu A ozářenu bodovým zdrojem a ležící kolmo k ose kužele, který ji osvětluje, lze převést tuto plochu na prostorový úhel pomocí vztahu [22]:

$$d\omega = \frac{dA}{r^2} \ ,$$

kde r je vzdálenost plošky od zdroje. Pokud je ploška pod úhlem θ vůči ose kuželu, platí následující vztah [22]:

$$d\omega = \frac{dA\cos\theta}{r^2} \ .$$

Po dosazení do vzorce dostáváme:

$$\begin{aligned} I_e &= \frac{d\Phi_e}{d\omega} \ , \\ I_e &= \frac{d\Phi_e r^2}{dA\cos\theta} \ , \\ I_e &= E_e\frac{r^2}{\cos\theta} \ , \\ E_e &= \frac{I_e}{r^2}\cos\theta \ . \end{aligned}$$

Z toho plyne, že intenzita ozáření plochy bodovým zdrojem je závislá na zářivosti zdroje, vzdálenosti plochy od zdroje a úhlu dopadu [22]. Tímto si tak potvrzujeme vlastnosti zmíněné výše.

Radiance

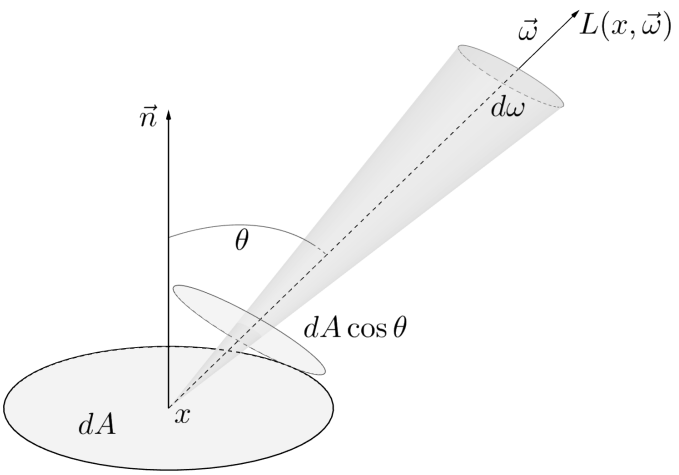
Jedná se o nejdůležitější radiometrickou veličinu, která se používá v algoritmech globálního osvětlování. Zatímco ozáření nám udává zářivý výkon dopadající na plochu A ze všech směrů, radiance se omezuje na zářivý výkon dopadající pouze v určitém směru a procházející skrze prostorový úhel $d\omega$ kolem směru $\vec{\omega}$.

Radiance závisí na poloze bodu x a směru $\vec{\omega}$. Značí se $L(x, \vec{\omega})$, jednotka je $W \cdot m^{-2} \cdot sr^{-1}$ a platí:

$$L(x, \vec{\omega}) = \frac{d^2\Phi}{d\omega \, dA \, \cos\theta} \ .$$

Je důležité upozornit na to, že se radiance měří vzhledem k ploše kolmé ke směru $\vec{\omega}$. Je tak potřeba plochu nejdříve promítnout do správné polohy. K tomu ve vzorci slouží $\cos\theta$, kde θ značí úhel mezi normálou povrchu a směrem $\vec{\omega}$. Formálně pak radiance udává výkon na jednotkovou plochu dA promítnutou kolmo ke směru $\vec{\omega}$ na jednotkový prostorový úhel $d\omega$.

Jelikož pracujeme s diferenciálními hodnotami, můžeme jinými slovy říct, že radiance je energie paprsku, určeného bodem x a směrem $\vec{\omega}$. V počítačové grafice interpretujeme tuto veličinu jako barvu.



Obrázek 4.10: Radiance

Radiance je navíc oproti ozáření konstantní se vzdáleností r. Stejně tak hodnotu radiance neovlivňuje úhel, který ploška svírá se směrem $\vec{\omega}$ [12].

4.4 Odraz a lom světla

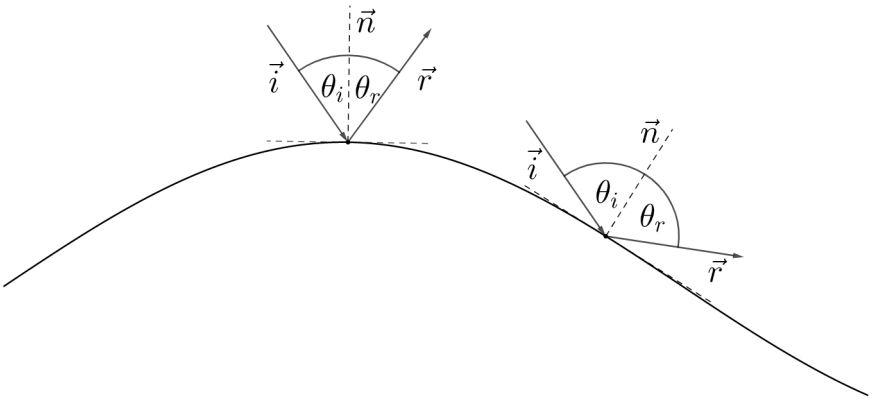
Odraz a lom světla (*reflection* a *refraction*) jsou další důležité vlastnosti světla, které se využívají v počítačové grafice a se kterými se setkáváme každý den. Jako příklad můžeme uvést sklo či vodu, u kterých můžeme pozorovat obě tyto vlastnosti, které jsou dále popsány Fresnelovými rovnicemi.

Následující část textu čerpá převážně z knihy od J. Žáry [11], pokud není uvedeno jinak.

Odraz světla

Při dopadu světelného paprsku na povrch tělesa se jeho určitá část odrazí zpět do prostoru. Vlastnosti odraženého světla udává materiál povrchu, ze kterého se odrazilo. Odraz samotný se řídí zákonem který říká, že velikost úhlu odrazu se rovná velikosti úhlu dopadu.

Dopadá-li světelný paprsek na povrch tělesa pod úhlem θ_i sevřeným mezi paprskem a kolmicí k povrchu v daném bodě dopadu, je velikost úhlu odrazu θ_r mezi odchozím paprskem a již zmíněnou kolmicí rovna úhlu dopadu. Tedy $\theta_i = \theta_r$ viz obrázek 4.11.



Obrázek 4.11: Odraz světla

Často se setkáme s tím, že známe vektor udávající směr světla $\vec{i}$ a normálu $\vec{n}$ k povrchu a chceme zjistit vektor $\vec{r}$ udávající směr odrazu. Ten můžeme snadno dopočítat následujícím způsobem:

$$\vec{r} = \vec{i} - 2(\vec{i} \cdot \vec{n})\vec{n} \ ,$$

kde $i \cdot n$ značí skalární součin mezi normálou n a vektorem paprsku i.

Lom světla

K lomu světla dochází při přechodu světelných paprsků mezi dvěma prostředími s různou optickou hustotou a tedy odlišnou rychlostí šíření světla. Lom světla se řídí Snellovým zákonem lomu:

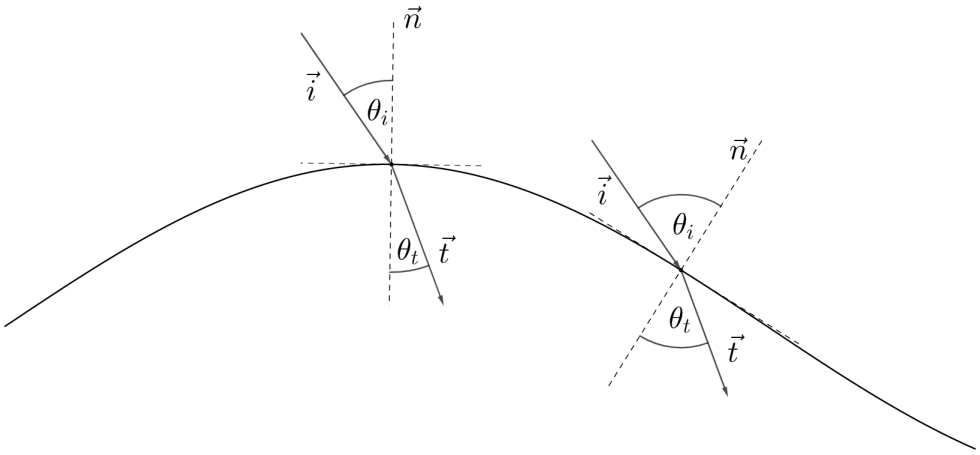
$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i} \ , \tag{4.3}$$

kde θ_i značí úhel dopadu a θ_t značí úhel lomu, které jsou měřeny mezi paprskem příchozím, respektive odchozím, a normálou povrchu v bodě dopadu. Symboly η_i a η_t značí tzn. index lomu, který je dán prostředím a lze ho vypočítat pomocí vzorce:

$$\eta = \frac{c}{v} \ ,$$

kde c je rychlost světla ve vakuu a v je rychlost světla v daném prostředí.

Index lomu rovněž závisí na vlnové délce. To má za následek, že se světlo různých barev láme jinak. Tento jev je nazýván *disperze* a můžeme ho například sledovat při průchodu světla skrz hranol. Disperze se v počítačové grafice většinou zanedbává, ačkoliv existují softwary, které ji dokáží simulovat, například komerční renderer V-ray.



Obrázek 4.12: Lom světla

Použijeme-li značení z obrázku 4.12 a předpoklad, že jsou vektory $\vec{i}$ a $\vec{n}$ jednotkové, tak pro výpočet vektoru lomu platí následující [23]:

$$t = \frac{\eta_i}{\eta_t}(\vec{i} + \vec{n}\cos\theta_i) - \vec{n}\cos\theta_t \ ,$$

$$\cos\theta_i = \vec{n} \cdot \vec{i} \ ,$$

$$\cos\theta_t = \sqrt{1 - (\sin\theta_t)^2} = \sqrt{1 - \left(\frac{\eta_i}{\eta_t}\right)^2 (1 - \cos\theta_i)} \ .$$

V případě, že se světlo šíří z opticky hustšího prostředí do jiného řidšího prostředí, může dojít k tomu, že úhel lomu je roven pravému úhlu. Pokud nastane taková situace, je $\sin\theta_t = 1$ a následně platí:

$$\sin\theta_c = \sin\theta_i = \frac{\eta_t}{\eta_i} \ .$$

Úhel θ_c se nazývá jako kritický nebo mezní úhel a jedná se o největší úhel, pod kterým ještě nastává lom světla. Pokud je úhel θ_i větší než θ_c, dochází k tzv. totálnímu odrazu. To znamená, že světlo putující z opticky hustšího prostředí se nedostane do druhého řidšího prostředí a místo toho se odrazí. Tento jev můžeme pozorovat například při přechodu světla mezi vodou a vzduchem.

Pro zjištění hodnoty kritického úhlu lze použít následující vzorec:

$$\theta_c = \arcsin\left(\frac{\eta_t}{\eta_i}\right) \ .$$

Fresnelovy rovnice

V úvodu této sekce bylo zmíněno, že některé materiály, nejčastěji voda či sklo, světlo odráží i lámou. Kolik světla se odrazí nebo lomí závisí na úhlu dopadu, kde platí, že čím větší úhel dopadu, tím více světla se odrazí a naopak. Poměr mezi světlem odraženým a světlem lomeným lze určit pomocí tzv. Fresnelových rovnic [24].

Pro lepší pochopení je třeba zmínit, že se světlo skládá ze dvou na sebe kolmých vln, které nazýváme jako kolmou (s) a rovnoběžnou (p) polarizaci. Pro výpočet Fresnelových rovnic je třeba brát v potaz obě tyto vlny a platí [24]:

$$R_s = \left(\frac{\eta_i\cos\theta_i - \eta_t\cos\theta_t}{\eta_i\cos\theta_i + \eta_t\cos\theta_t}\right)^2 \ ,$$

$$R_p = \left(\frac{\eta_i\cos\theta_t - \eta_t\cos\theta_i}{\eta_i\cos\theta_t + \eta_t\cos\theta_i}\right)^2 \ ,$$

kde R_s a R_p značí koeficienty odrazu každé z vln. Pro celkový koeficient odrazu světla R platí:

$$R = \frac{R_s + R_p}{2}$$

a pro koeficient lomu T platí:

$$T = 1 - R \ .$$

Výše zmíněné rovnice platí pro dielektrické materiály. Pro ty metalické platí jiné rovnice, kde spíše než lom světla uvažujeme absorpci světla. Tyto rovnice vypadají následovně [24]:

$$R_s = \frac{a^2 + b^2 - 2a\cos\theta_i + \cos^2\theta_i}{a^2 + b^2 + 2a\cos\theta_i + \cos^2\theta_i} \ ,$$

$$R_p = R_s \frac{a^2 + b^2 - 2a\sin\theta_i\tan\theta_i + \sin^2\theta_i\tan^2\theta_i}{a^2 + b^2 + 2a\sin\theta_i\tan\theta_i + \sin^2\theta_i\tan^2\theta_i} \ ,$$

$$a^2 = \frac{1}{2\eta_d^2}\left(\sqrt{(\eta_c - k^2 - \eta_d^2\sin^2\theta_i)^2 + 4\eta_c^2k^2} + \eta_c^2 - k^2 - \eta_d^2\sin^2\theta_i\right) \ ,$$

$$b^2 = \frac{1}{2\eta_d^2}\left(\sqrt{(\eta_c - k^2 - \eta_d^2\sin^2\theta_i)^2 + 4\eta_c^2k^2} - (\eta_c^2 - k^2 - \eta_d^2\sin^2\theta_i)\right) \ ,$$

kde θ_i je úhel dopadu, η_d je index lomu dielektrika, skrz které se světlo šíří, η_c je index lomu kovového materiálu a k je tzv. *extinkční koeficient* kovu.

V praxi se však spíše než s aplikací těchto rovnic setkáme s jejich aproximací, které přinášejí značné zjednodušení při výpočtech.

Kapitola 5

Lokální a globální osvětlovací modely

5.1 BRDF

To, jak vidíme své okolí a objekty v něm, je dáno světlem, které doputuje do našeho oka a vytvoří tak obraz. Většina tohoto světla je odražena od objektů kolem nás a výsledná barva, kterou vnímáme, je tak dána vlastnostmi povrchu těchto objektů. Abychom mohli simulovat odrazové vlastnosti materiálů, potřebujeme určitý způsob, jak je matematicky popsat. K tomuto slouží funkce BRDF, u které počítáme s následujícími předpoklady:

- Světlo se od povrchu odrazí okamžitě (nekonečná rychlost světla).
- Foton o vlnové délce λ bude odražen se stejnou vlnovou délkou, tím se zanedbává fluorescence.
- Světlo dopadající do bodu x na povrchu materiálu, bude z téhož bodu i odraženo. To znamená, že zanedbáváme šíření světla pod povrchem (subsurface scattering), tento jev popisuje jiná funkce, zvaná BSSRDF.

BRDF, neboli *Bidirectional Reflectance Distribution Function* (v češtině *dvousměrná odrazová distribuční funkce*), byla poprvé popsaná okolo roku 1965 Fredem Nicodemusem [25]. Tato funkce se používá především v počítačové grafice a jak již bylo řečeno, matematicky popisuje odrazové vlastnosti povrchu materiálu v určitém bodě.

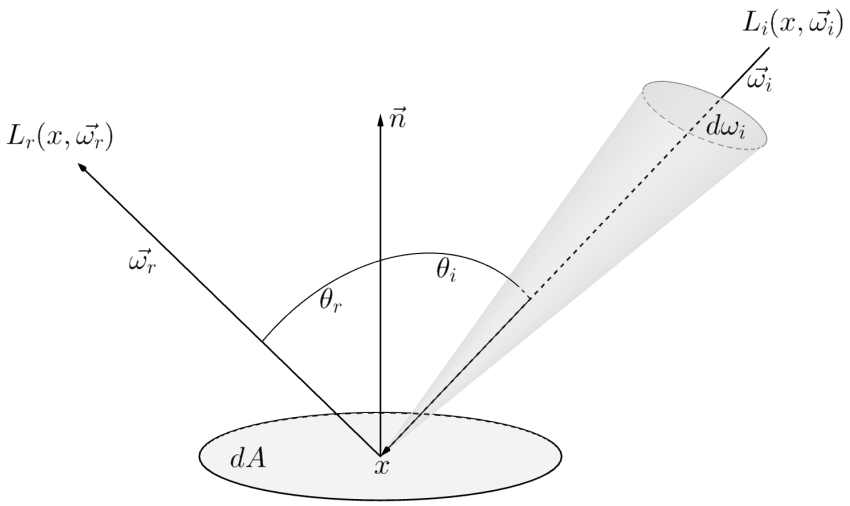
Mějme bod x na povrchu tělesa, na které dopadá světlo ze směru $\overrightarrow{\omega_i}$ a odráží se ve směru $\overrightarrow{\omega_r}$. Pro BRDF následně platí:

$$f_r(x, \overrightarrow{\omega_r}, \overrightarrow{\omega_i}) = \frac{dL_r(x, \overrightarrow{\omega_r})}{dE(x, \overrightarrow{\omega_i})} = \frac{dL_r(x, \overrightarrow{\omega_r})}{L_i(x, \overrightarrow{\omega_i}) \cos\theta_i d\omega_i} \ ,$$

kde:

- $dL_r(x, \overrightarrow{\omega_r})$ je radiance dopadající do bodu x ze směru $\overrightarrow{\omega_i}$,
- $L_i(x, \overrightarrow{\omega_i})$ je radiance vyzářená (odražená) z bodu x ve směru $\overrightarrow{\omega_r}$,
- $dE(x, \overrightarrow{\omega_i})$ je ozáření bodu x ze směru $\overrightarrow{\omega_i}$,
- θ_i je úhel mezi směrem $\overrightarrow{\omega_i}$ a normálou povrchu v bodě x,
- $d\omega_i$ je prostorový úhel kolem směru $\overrightarrow{\omega_i}$.

Formálně řečeno je BRDF funkce udávající poměr odražené radiance v daném bodě ku vstupní diferenciální radianci promítnuté na kolmou plochu.



Obrázek 5.1: Schéma BRDF

Vlastnosti BRDF

BRDF má několik vlastností:

- *Helmholtzův princip reciprocity* - Ten říká, že hodnota BRDF v určitém bodě zůstane stejná, pokud zaměníme směr dopadu světelného paprsku a směr jeho odrazu. $f_r(x, \omega_r, \omega_i) = f_r(x, \omega_i, \omega_r)$.
- *Pozitivita* - To znamená, že funkce nikdy není záporná $f_r(x, \omega_r, \omega_i) \geq 0$.
- *Linearita* - Ta říká, že hodnota BRDF pro daný vstupní úhel ω_i není závislá na hodnotách BRDF pro jiné vstupní úhly. To má za následek, že paprsek z daného směru je vyzářen bez ohledu na světelné paprsky přicházející z jiných směrů.
- *Izotropie* - BRDF většiny povrchů je izotropní. To znamená, že je invariantní vůči otočení kolem normály plochy a není tak závislá na orientaci materiálů vůči příchozímu a odchozímu paprsku. Pokud bychom chtěli simulovat anizotropní materiály, musíme počítat i s orientací daného materiálu. Takovým materiálem je například broušený kov.
- *Zákon zachování energie* - Ten říká, že plocha nemůže odrazit více zářivého toku, než kolik na tuto plochu dopadá (energie nemůže vzniknout ani zaniknout, pouze se přeměnit na jiný druh). To znamená, že platí následující pravidlo:

$$\int_\Omega f_r(x, \overrightarrow{\omega_r}, \overrightarrow{\omega_i}) \cos\theta_i d\omega_i < 1, \forall \overrightarrow{\omega_i} \ .$$

Jedna z nevýhod BRDF je, že nemá shora omezený obor hodnot. Proto se někdy pro vyjádření vlastností materiálu používá odrazivost (*reflectance*), která udává poměr odraženého světelného toku $\Phi_r(x)$ v bodě x vůči dopadajícímu světelnému toku $\Phi_i(x)$ ve stejném bodě:

$$\rho(x) = \frac{\Phi_r(x)}{\Phi_i(x)} = \frac{\int_\Omega L_r(x, \overrightarrow{\omega_r}) \cos\theta_r d\omega_r}{\int_\Omega L_i(x, \overrightarrow{\omega_i}) \cos\theta_i d\omega_i} \ .$$

Obor hodnot pro odrazivost je interval $\langle 0, 1 \rangle$. Odrazivost také můžeme najít pod názvem *albedo*.

Druhy BRDF podle odrazu

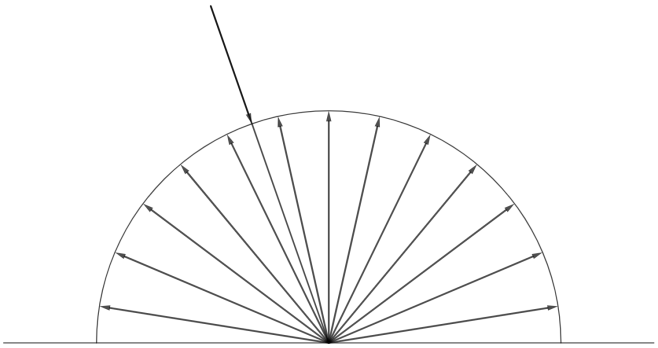
V běžném světě existuje mnoho druhů materiálů. V počítačové grafice si ale vystačíme s popisem několika základních druhů odrazů, pomocí kterých jsme schopni tyto materiály simulovat. Tyto materiály simulujeme za pomoci různých empirických vyjádření funkce BRDF, které jsou navrženy tak, aby pokud možno splňovaly její základní vlastnosti [11].

- **Difúzní odraz**

  Jedná se o empirický model odrazu od povrchu difúzních (matných) materiálů. Někdy se také hovoří o *Lambertovském* odrazu, který je však jen matematickou abstrakcí a říká, že příchozí světlo je odraženo rovnoměrné ve všech směrech. Pro BRDF popisující tento jev platí:

  $$f_d(x, \overrightarrow{\omega_r}, \overrightarrow{\omega_i}) = f_d(x) \ .$$

  Takový přepis si můžeme dovolit z toho důvodu, že BRDF pro difúzní povrch nezávisí na směru $\overrightarrow{\omega_i}$ ani $\overrightarrow{\omega_r}$ a je pro daný bod konstantní ve všech směrech.



Obrázek 5.2: Difúzní odraz

I v tomto případě lze difúzní odraz v určitém bodě vyjádřit pomocí reflektance, pro kterou rovněž platí že je konstantní:

$$\rho_d(x) = \frac{\Phi_r(x)}{\Phi_i(x)} = \pi f_d(x)$$

a lze z ní vyjádřit BRDF člen:

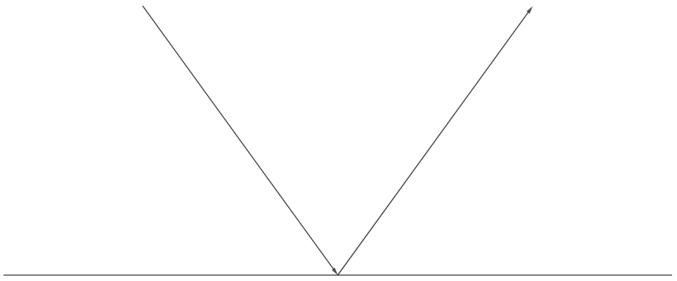
$$f_d(x) = \frac{\rho_d(x)}{\pi} \ .$$

- **Zrcadlový odraz**

  Dalším druhem odrazu, se kterým se často setkáme, je zrcadlový, či spekulární odraz (*specular reflection*), kterému odpovídají materiály jako leštěný kov, voda či sklo. Tento druh odrazu je matematicky totožný s běžným odrazem popsaným v kapitole 4.4 a pro spekulární BRDF funkci platí:

$$f_s(x, \overrightarrow{\omega_r}, \overrightarrow{\omega_i}) = \frac{1}{\cos\theta_i}\delta(\cos\theta_i - \cos\theta_r)\delta[\phi_i - (\phi_r \pm \pi)] \ ,$$

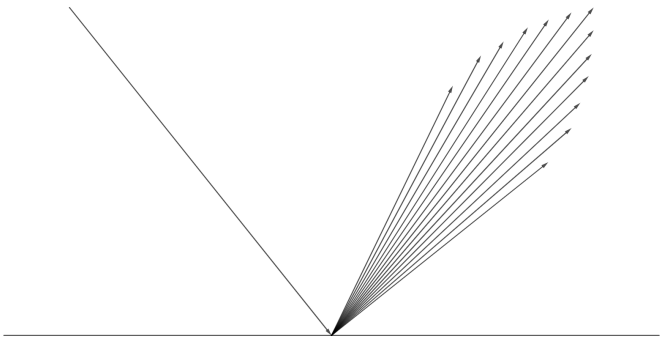
  kde ϕ je tzv. Diracova funkce. Pro dokonalé zrcadlo platí, že odráží pouze v jediném směru.



Obrázek 5.3: Zrcadlový odraz

- **Lesklý odraz**

  Za lesklý odraz (*glossy reflection*) považujeme nedokonalý zrcadlový odraz. Ten je výsledkem mnoha složitých jevů, které jsou ovlivněny povrchem složeným z mikroplošek, které se vzájemně stíní a umožňují světlu pronikat pod povrch materiálu.



Obrázek 5.4: Lesklý odraz

Tento odraz je velmi složité simulovat a neexistuje analytická formule pro BRDF takového povrchu. Proto se používají její různé aproximace. Nejjednodušší z nich je ta následující:

$$f_r(x, \omega_r, \omega_i) = \frac{DGF}{4 \cos \theta_r \cos \theta_i} \ ,$$

kde:

- D popisuje distribuci mikroplošek,
- G popisuje stínění mezi jednotlivými mikroploškami,
- F je Fresnelův koeficient odrazivosti, viz kapitola 4.4.

Tato BRDF funkce je známá jako Cook-Torrance BRDF funkce pro výpočet spekulárního odrazu [26].

5.2 Lokální osvětlovací metody

Tato část se věnuje metodám lokálního osvětlení (*local illumination*), někdy nazývané jako přímé osvětlení (*direct illumination*). Tyto metody zjišťují barvu sledovanou v konkrétním bodě objektu, která je ovlivněna pouze světlem přicházejícím přímo ze světelného zdroje (odtud přímě osvětlení). Jinými slovy se jedná o metody sledující paprsek světla vyzářený určitým zdrojem a odraženým přímo do místa pozorování. Často se setkáme s více světelnými zdroji ve scéně. V takovém případě bereme v potaz všechny tyto zdroje.

Rovnice pro výpočet lokálního osvětlení v určitém bodě bere v potaz všechny vstupní směry světla $\overrightarrow{\omega_i}$ a výsledkem je odražená radiance v určitém směru $\overrightarrow{\omega_r}$:

$$L_r(x, \overrightarrow{\omega_r}) = \int_\Omega f(x, \overrightarrow{\omega_r}, \overrightarrow{\omega_i}) L_i(x, \overrightarrow{\omega_i}) \cos\theta d\omega_i \ .$$

Abychom získali celkovou radianci opouštějící povrch ve směru $\overrightarrow{\omega_r}$ a tím i barvu v určitém bodě na povrchu, musíme zjistit integrál všech radiancí dopadajících na daný bod x a jejich poměr odrazu vůči výstupnímu úhlu. Proto se v rovnici výše nachází člen $f(x, \overrightarrow{\omega_r}, \overrightarrow{\omega_i})$ vyjadřující BRDF.

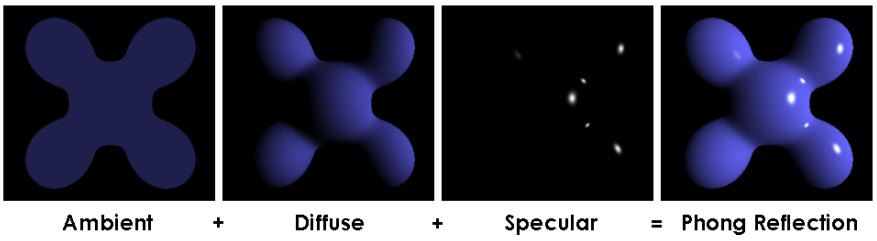
V praxi se nejčastěji setkáme s několika empirickými modely lokálního osvětlení, které si nyní představíme.

5.2.1 Phongův osvětlovací model

Tento model vznikl v 70. letech 20. století a je pojmenován po jeho tvůrci Bui-Tuong Phongovi [27]. Tento model narozdíl od toho Lambertovského umí pracovat i s lesklým povrchem a skládá se ze tří různých složek – difúzní, zrcadlové a ambientní (*diffuse, specular, ambient*). Každá složka zde zastává jinou funkci a pro výpočet výsledného osvětlení podle Phongova modelu platí následující:

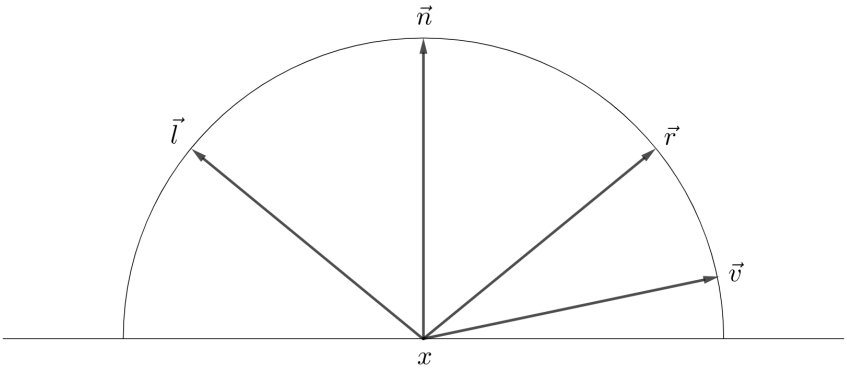
$$I = I_a + I_d + I_s \ .$$

Pro lepší představu se můžeme podívat na obrázek 5.5 níže, který tento vzorec znázorňuje v praxi.



Obrázek 5.5: Ukázka jednotlivých složek Phongova modelu a jejich výsledný součet [28].

Při výpočtech budeme vycházet z následujícího diagramu:



Obrázek 5.6: Geometrie odrazu Phongova modelu

Difúzní složka

Difúzní složka Phongova modelu vychází z teorie Lambertova odrazu, tedy že světlo je odráženo rovnoměrně do všech směrů. Pro jeho výpočet platí následující vzorec:

$$I_d = I_L r_d(\vec{l} \cdot \vec{n}) \ ,$$

kde I_L reprezentuje barevné složení dopadajícího paprsku ve formě tří-složkového vektoru (RGB), r_d je koeficient difúzního odrazu, rovněž trojsložkový vektor a $(\vec{l} \cdot \vec{n})$ je skalární součin mezi jednotkovým vektorem $\vec{n}$ značícím normálu plochy v bodě dopadu a $\vec{l}$ značícím směr dopadajícího paprsku.

Koeficient r_d můžeme do značné míry uvažovat jako barvu povrchu a celý vzorec má smysl pouze pokud platí $(\vec{l} \cdot \vec{n}) > 0$. V opačném případě povrch není osvětlen daným světelným paprskem.

Výsledkem skalárního součinu $(\vec{l} \cdot \vec{n})$ je $\cos\theta_i \in \langle 0, 1\rangle$, tedy kosinus úhlu sevřeného mezi normálou povrchu a směrem dopadajícího paprsku. Z toho plyne, že výsledné množství světla I_d je ovlivněno právě úhlem, pod kterým světlo dopadá na povrch.

Spekulární složka

Spekulární složka reprezentuje lesk materiálu, neboli intenzitu světla odraženou od tělesa ve směru pozorování a je vyjádřena jako:

$$I_s = I_L r_s (v \cdot r)^h \ ,$$

kde $r_s \in \langle 0, 1\rangle$ je koeficient zrcadlového odrazu určující míru zastoupení zrcadlové složky ve výsledném odrazu a zároveň se jedná o tří složkový vektor. Vektor $\vec{r}$ je vektor odrazu viz sekce 4.4, $\vec{v}$ je vektor pohledu a koeficient $h \in \langle 1, \infty\rangle$ udává ostrost zrcadlového odrazu.

Pokud je výsledek skalárního součinu $(\vec{v} \cdot \vec{r}) < 0$, znamená to, že pozorovatel neuvidí odraz daného paprsku a I_S bude nulový vektor. Čím větší je koeficient h, tím menší a ostřejší se jeví odlesk na povrchu.

Ambientní složka

Ambientní složka simuluje okolní rozptýlené světlo, vzniklé odrazem od ostatních objektů a molekul vzduchu a je popsána jednoduchým vztahem:

$$I_a = I_A r_a \ ,$$

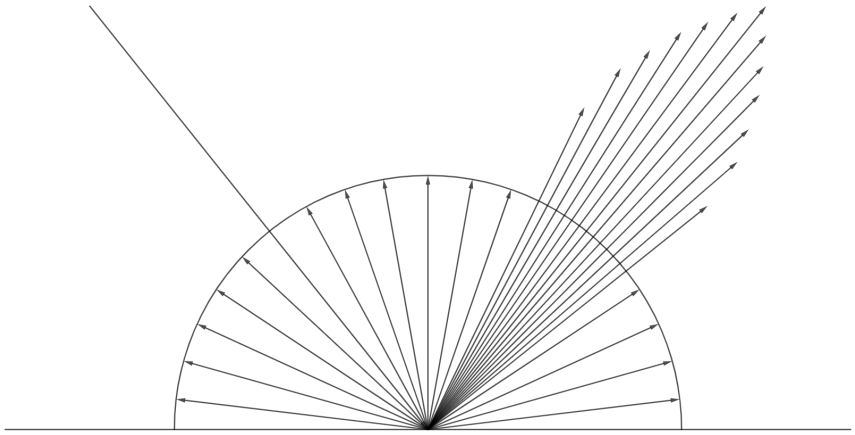
kde I_A je konstanta značící množství okolního světla ve scéně, které se upravuje vůči počtu světelných zdrojů a r_a je tří-složkový vektor určující odrazivou schopnost materiálu a bývá totožný s koeficientem r_d.

Ambientní složka se používá zejména pro to, aby se části objektů odvrácené od světelného zdroje nejevili jako černé.

Mnohdy se stává, že se ve scéně nachází více světelných zdrojů. Pro výpočet osvětlení v bodě P osvětleného M světelnými zdroji platí následující vztah [11]:

$$I = I_A r_a + \sum_{k=1}^{M} I_L \left[r_s (\vec{v} \cdot \vec{r})^h + (\vec{l} \cdot \vec{n}) \right] \ .$$

Právě tato rovnice je to, co nazveme jako Phongův osvětlovací model. Jak již bylo řečeno, jedná se pouze o empirický model, který nesplňuje všechny požadavky dané BRDF funkcí a tudíž není fyzikálně korektní, je však často používán pro jeho jednoduchost a uspokojivé výsledky.

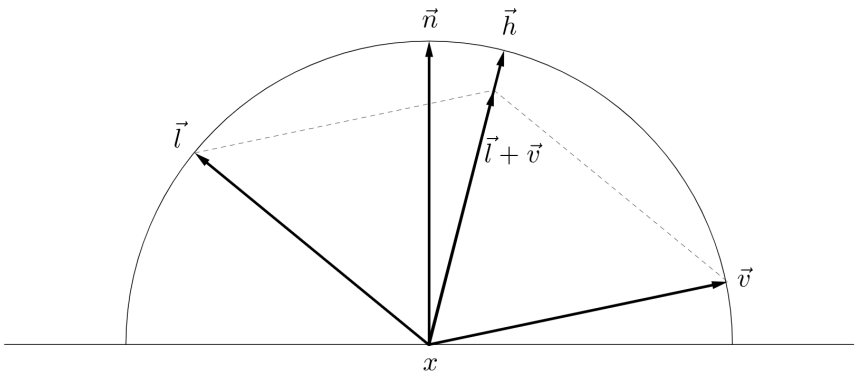


Obrázek 5.7: Ukázka odraženého světla ve Phongově modelu

5.2.2 Blinn-Phongův osvětlovací model

Blinn-Phongův osvětlovací model [29] vychází z Phongova osvětlovacího modelu s modifikací výpočtu spekulární složky, kde se nahrazuje výpočet skalárního součinu mezi vektorem odrazu a vektorem pohledu za skalární součin mezi tzv. půlvektorem $\vec{h}$ a normálou $\vec{n}$. Pro půlvektor platí:

$$\vec{h} = \frac{\vec{l} + \vec{v}}{||\vec{l} + \vec{v}||} \ .$$



Obrázek 5.8: Geometrie odrazu Blinn-Phongova modelu

Tato úprava ve výpočtu zde slouží jako optimalizace, jelikož výpočet půlvektoru je rychlejší, než výpočet vektoru odrazu. Pro spekulární složku tak platí:

$$I_s = I_L r_s (\vec{h} \cdot \vec{n})^{h_b} \ .$$

Mocnitel h_b neodpovídá mocniteli h z původního výpočtu a přibližně platí, že $(\vec{r} \cdot \vec{v})^h \approx (\vec{h} \cdot \vec{n})^{4h_b}$.

Takto upravený model má mírně odlišné výsledky od toho původního ale vizuální rozdíl je téměř zanedbatelný.

5.2.3 Lambertův osvětlovací model

Jedná se o jeden z nejjednodušších modelů lokálního osvětlení, který počítá pouze s difúzními (matnými) povrchy. Stejně jako difúzní složka Phongova modelu vychází z teorie Lambertovského odrazu. Jeho výpočet je rovněž totožný s výpočtem difúzní složky Phongova modelu a platí tedy následující vzorec [30]:

$$I_d = I_L r_d (\vec{l} \cdot \vec{n}) \ .$$

5.3 Globální osvětlovací metody

Globální osvětlení (*global illumination*), také nazývané jako nepřímé osvětlení (*indirect illumination*), narozdíl od lokálního osvětlení vyhodnocuje vliv okolí na výsledný vzhled zkoumaného objektu. Jinými slovy, zkoumá světlo dopadající na povrch nejen ze světelného zdroje, ale i světlo odražené od objektů v okolí.

Vzájemné vztahy mezi objekty lze popsat pomocí tzv. zobrazovací rovnice, jejíž řešení je však analyticky nemožné a v praxi se využívají její různé aproximace.

Zobrazovací rovnice

Zobrazovací rovnici publikoval v roce 1986 James T. Kajiya [31] a popisuje přenos světla ve scéně. Nejdříve si ale ukažme rovnici odrazu, ze které si vyjádříme tu zobrazovací:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + L_r(x, \vec{\omega}) \ ,$$

kde $L_o(x, \vec{\omega})$ značí celkovou radianci opuštějící bod x ve směru $\vec{\omega}$, $L_e(x, \vec{\omega})$ je radiance vyzářená z bodu x ve směru $\vec{\omega}$ a $L_r(x, \vec{\omega})$ vyjadřuje celkovou odraženou radianci z bodu x ve směru $\vec{\omega}$, která lze vyjádřit jako integrál:

$$L_r(x, \vec{\omega}) = \int_\Omega f(x, \vec{\omega}, \vec{\omega_i}) L_i(x, \vec{\omega_i}) \cos\theta d\omega_i \ ,$$

čímž dostáváme konečnou rovnici odrazu:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_\Omega f(x, \vec{\omega}, \vec{\omega_i}) L_i(x, \vec{\omega_i}) \cos\theta d\omega_i \ .$$

S touto rovnicí jsme se již setkali v části 5.2 a popisuje lokální odraz světla.

Mějme nyní funkci $r(x, \vec{\omega_i})$, která vrhne paprsek z bodu x ve směru $\vec{\omega_i}$ a najde nejbližší průsečík s jiným objektem ve scéně. Jelikož je radiance podél paprsku konstantní, můžeme nahradit příchozí radianci $L_i(x, \vec{\omega_i})$ za radianci odchozí v opačném směru z bodu nalezeného pomocí funkce $r(x, \vec{\omega_i})$. Platí tedy:

$$L_i(x, \vec{\omega_i}) = L_o(r(x, \vec{\omega_i}), -\vec{\omega_i}) \ .$$

Pokud nahradíme výraz $L_i(x, \vec{\omega_i})$ v původní rovnici odrazu, za výraz $L_o(r(x, \vec{\omega_i}), -\vec{\omega_i})$, dostaneme následující rovnici:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_\Omega f(x, \vec{\omega}, \vec{\omega_i}) L_o(r(x, \vec{\omega_i}), -\vec{\omega_i}) \cos\theta d\omega_i \ , \qquad (5.1)$$

kterou nazveme zobrazovací. Z této rovnice si můžeme všimnout, že neznámá L_o se nachází na obou stranách rovnice a vede na rekurzivní řešení.

Tato rovnice využívá integrálu přes polokouli Ω. V praxi je však výhodnější použít formulaci, která odraženou radianci udává jako integrál přes všechny přispívající plochy ve scéně. Ta má tvar:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + \int_S f(x, y \to x, \vec{\omega}) L_o(y \to x) G(x, y) V(x, y) dA_y \ ,$$

kde S značí všechny plochy ve scéně, y značí všechny body plochy A_y. V tomto vztahu se využívá notace $y \to x$, která značí směr odrazu radiance. Rovněž se zde využívají dva nové členy, kde $V(x, y)$ určuje viditelnost bodu x z bodu y a tedy platí:

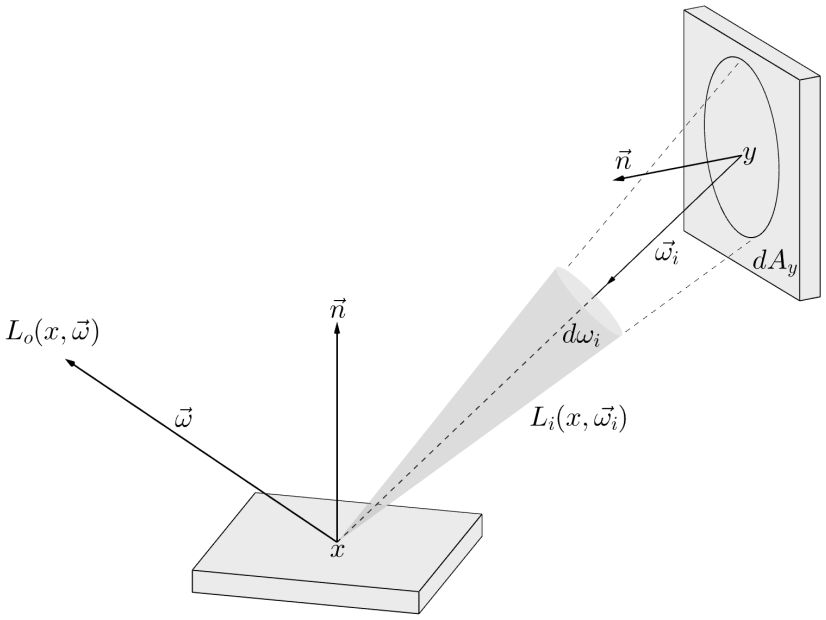
$$V(x, y) = \begin{cases} 1 & \text{pokud je bod } x \text{ viditelný z bodu } y \\ 0 & \text{v opačném případě} \end{cases}$$

a $G(x, y)$ je geometrický člen, který zohledňuje orientaci daných ploch v prostoru a platí:

$$G(x, y) = \frac{\cos\theta_x \cos\theta_y}{||x - y||^2} \ ,$$

kde $\cos\theta_x$ je kosinus úhlu mezi normálou k povrchu v bodě x, $\cos\theta_y$ je kosinus úhlu mezi normálou k povrchu v bodě y.

Většina renderovacích metod využívajících globálního osvětlování využívá aproximace této rovnice pomocí Monte Carlo integrování.



Obrázek 5.9: Geometrie zobrazovací rovnice

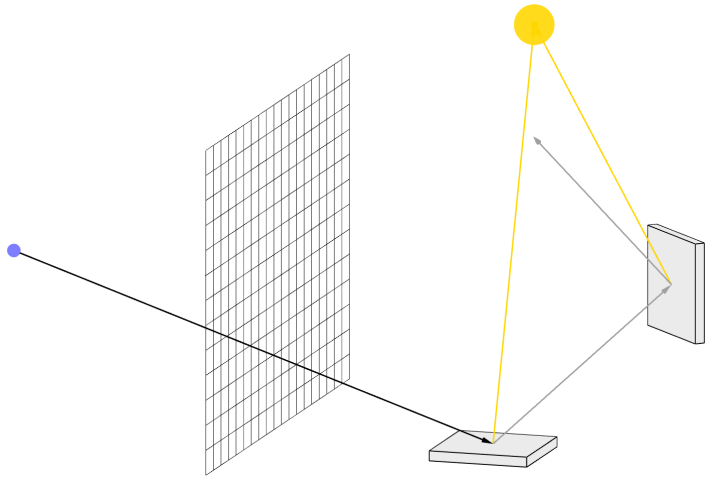
Tato sekce čerpá poznámky z přednášek pana Křivánka [32] a z knížky pana Žáry [11].

5.3.1 Ray tracing

Ray tracing vznikal ve stejné době jako ray casting zmíněný v části 3 a jeho autorem je T. Whitted [33]. Ačkoliv tato metoda vznikla nezávisle na ray castingu, dá se říct, že je jeho rozšířením. Jedná se o rekurzivní sledování paprsku a jeho putování scénou, přičemž tato metoda patří mezi nejznámější metody globálního osvětlování.

Rozšíření oproti ray castingu spočívá v tom, že po nalezení průsečíku s objektem je paprsek rozdělen na další paprsky, které jsou následně sledovány a jsou tak rekurzivně vyhodnocovány. Tyto paprsky dělíme na několik druhů:

- *Primární paprsek (primary ray)* – Tento paprsek je to, co známe z algoritmu ray casting. jedná se o paprsek vyslaný z místa pozorování skrze průmětnu a směrem do scény.
- *Sekundární paprsek (secondary ray)* – Sekundární paprsek je vytvořen poté, co je nalezen průsečík primárního nebo sekundárního paprsku s objektem ve scéně. Z nalezeného bodu je vyslán paprsek ve směru odrazu a pro určité materiály i ve směru lomu. Počet sekundárních paprsků je dán hloubkou rekurze, kde ze sekundárních paprsků mohou vznikat další sekundární paprsky.
- *Stínový paprsek (shadow ray)* – Tento paprsek se vysílá z každého nalezeného průsečíku objektu s primárním nebo sekundárním paprskem ve směru k světelnému zdroji. Jeho úkolem je zjistit, zda-li je nalezený bod v zákrytu vůči světelnému zdroji, či nikoliv. Tento paprsek se vysílá ke každému světelnému zdroji a pokud platí, že bod není v zákrytu vůči určitému světelnému zdroji, je tento zdroj zahrnut do výpočtu výsledného osvětlení v tomto bodě.



Obrázek 5.10: Ukázka ray tracingu

Tento algoritmus má v jeho základní verzi určitá omezení a nedostatky, jmenovitě [11]:

- Stíny vytvořené tímto postupem jsou ostré.
- Počítá se pouze s bodovými zdroji světla.
- Ačkoliv zrcadlové plochy odrážejí obraz okolních objektů, nepřispívají odrazem světelných paprsků při výpočtu globálního osvětlení.
- Při změnách ve scéně je potřeba celý výpočet provést znovu.
- Nevyužívá efektivnější metody pro výpočet osvětlení nelesknoucích se ploch.
- Nedokáže počítat kaustiky ani difúzní přenos barvy.

Existují však modifikace tohoto algoritmu, které umožňují pracovat i s jinými zdroji světla a tím vytvářet i polostíny. Nejdůležitější z nich je pak *distributed ray tracing*, ačkoliv je nutné zmínit, že tyto modifikace jsou výpočetně velmi náročně.

Rozšíření Phongova osvětlovacího modelu

Pro výpočet barvy paprsku se používají metody lokálního osvětlení. Chceme-li použít Phongovu metodu, musíme ji nejprve rozšířit a to tak, že bude platit následující vzorec:

$$I = I_s + I_d + I_a + I_r + I_t \ .$$

Můžeme si všimnout, že zde přibyly další dvě složky I_r a I_t, kde:

$$I_r = r_s I_R \ ,$$
$$I_t = r_s I_T \ .$$

Členy I_R a I_T značí barvu paprsku přicházející ze směru odrazu, respektive lomu a r_s je koeficient zrcadlového odrazu, který je stejný jako při výpočtu spekulární složky [11].

5.3.2 Path tracing

Path tracing metoda byla popsána Jamesem. T. Kajiyaou jako řešení zobrazovací rovnice ve stejném článku, ve kterém byla tato rovnice uvedena [31]. Path tracing se podobá metodě ray tracing s tím rozdílem, že dokáže pracovat s libovolnou formou světelných zdrojů, počítá s příspěvky od nepřímých difúzních odrazů a je schopna počítat kaustiky a difúzní přenos barvy.

Tato metoda využívá metody Monte Carlo, a pro to se také někdy nazývá jako *Monte Carlo path tracing*.

V algoritmu path tracing se na počátku vyšlou primární paprsky z místa pozorování skrze jednotlivé pixely tak, jako tomu je u metody ray tracing. Pokud pro daný paprsek nalezneme průsečík s objektem, vypočítáme lokální osvětlovací model v tomto bodě za využití stínových paprsků. V případě plošných světelných zdrojů vyšleme stínové paprsky do náhodného bodu na jejich povrchu.

V dalším kroku vyšleme sekundární paprsek v náhodném směru určeným hemisférou nad průsečíkem. Tento postup generování sekundárních paprsků, hledání jejich průsečíků a vyhodnocování lokálního osvětlovacího modelu opakujeme, dokud nedosáhneme předem dané hloubky rekurze, případně do platnosti určitých kritérií.

Mohli jsme si všimnout, že rozdíl mezi metodou path tracing a ray tracing spočívá v generování sekundárních paprsků, kde v případě meotody path tracing negenerujeme tyto paprsky podle zrcadlového odrazu, ale jejich směr vybíráme náhodně.

Takto popsaný algoritmus však není příliš efektivní a často vzniká šum. Existují však postupy, jak tento algoritmus vylepšit [11]:

- Vylepšení této metody spočívá právě v použití metody Monte Carlo, kde skrze každý pixel vedeme více než jeden paprsek a jejich průměrem získáme odhad integrálu příchozí radiance. Velký počet těchto paprsků pomáhá eliminovat šum, ale přidává na složitosti a zpomaluje tak celkový výpočet.

- Plošné světelné zdroje se mohou vzorkovat i více než jedním stínovým paprskem.

- Vytváření sekundárních paprsků pomocí vzorkování podle BRDF daného materiálu. To v praxi znamená, že generování náhodného směru sekundárního paprsku bude záviset na hustotě pravděpodobnosti dané BRDF funkce. To znamená, že většina generovaných paprsků bude mít směr podobný směru skutečného odrazu.

Obecně můžeme výsledek tohoto algoritmu vylepšit nastavením větší hloubky rekurze a zvětšením počtu primárních paprsků, ale za cenu vyššího výkonu.

Úskalím tohoto algoritmu je, že ačkoliv dokáže kaustiky vytvářet, je potřeba generovat velmi mnoho paprsků.

5.3.3 Metody vycházející od světelného zdroje

Výše zmíněné metody začínají sledovat světelné paprsky v obráceném pořadí, tedy od pozorovatele a postupně se trasují až k světelnému zdroji. Tyto metody nejsou příliš vhodné pro scény, ve kterých se nachází malé, nebo skryté světelné zdroje, jelikož je potřeba velké hloubky rekurze a mnoha paprsků, aby byl výsledek uspokojivý. Z tohoto důvodu existují i metody vycházející

od světelného zdroje, které jsou vhodnější pro tento typ scén a snadněji počítají kaustiky.

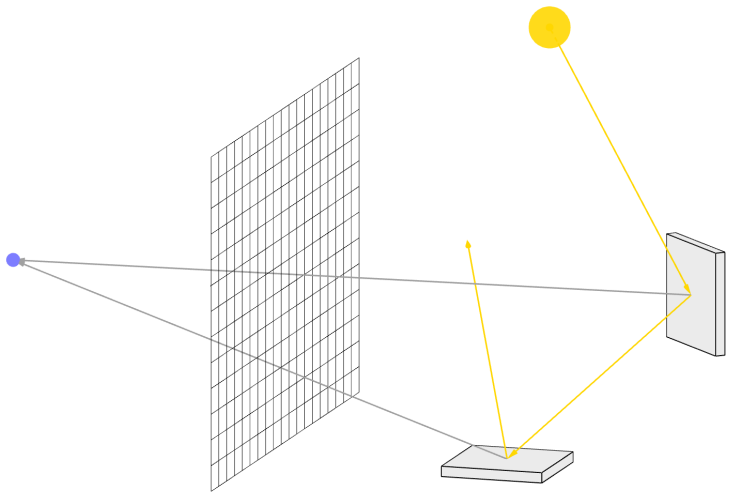
Tyto metody aproximují řešení zobrazovací rovnice náhodným sledováním paprsků vycházejících ze světelného zdroje směrem do scény. Nejprve se určí počáteční výkon odcházejícího paprsku, který je následně rekurzivně sledován při průchodu scénou. V každém průsečíku tohoto paprsku se vypočítá osvětlení pro tento bod a vyšle se paprsek směrem k pozorovateli, který určuje barvu daného bodu.

Tento paprsek určuje příspěvek do výsledné barvy pixelu, skrze který prochází. Každý pixel pak počítá s příspěvky všech paprsků, které jím prochází a výsledná barva je průměrem těchto příspěvků. Původní paprsek pokračuje ve své cestě, dokud nedosáhne určité hloubky rekurze nebo jiné ukončovací podmínky.

Problémem těchto metod je potenciální potřeba velkého množství paprsků, z nichž se mnoho nemusí vůbec využít z toho důvodu, že vůbec nedorazí k pozorovateli. Dalším problémem je velký šum, jelikož není možné ovlivnit vzorkování jednotlivých pixelů, a tak nezbývá než jen generovat další náhodné paprsky a doufat, že ovlivní potřebné pixely.

Z těchto metod můžeme jmenovat například *Photon tracing (sledování fotonů)*, který odpovídá obrácené metodě ray tracingu, nebo *Monte Carlo light tracing (sledování světla)*, která je pro změnu opačnou metodou path tracingu.

V praxi se tyto metody samostatně příliš nevyužívají, ale ve spojení s metodami vycházejícími od pozorovatele vznikají tzv. *dvousměrové metody*, které využívají výhod obou postupů.



Obrázek 5.11: Ukázka metod vycházejících od světelného zdroje

5.3.4 Bidirectional path tracing

Tuto metodu zveřejnil Eric P. Lafortune v roce 1993 [34] a využívá jak paprsků začínajících od pozorovatele, tak i od světelných zdrojů a vychází z metody path tracingu. To znamená, že všechny odrazy jsou určovány v náhodném směru určeným hustotou funkce BRDF. Tato metoda je schopna generovat všechny optické jevy a to i ve stížených podmínkách.

Jak již bylo řečeno, metoda generuje dva paprsky najednou, z nichž jeden je veden od světelného zdroje směrem do scény a druhá od pozorovatele skrze pixel. Tyto paprsky se ve scéně odráží a průsečíky těchto paprsků si označíme jako $x_1, \ldots, x_n$ pro paprsek vycházející od pozorovatele a $y_1, \ldots, y_n$ pro paprsek vycházející ze světelného zdroje. Jakmile je cesta obou paprsků ukončena, určí se viditelnost a vzájemné světelné příspěvky mezi všemi kombinacemi průsečíků x_i a y_j tzn. každý průsečík x_i se spojí s každým průsečíkem y_j a vypočítá se vzájemný vliv. Pro paprsek vyslaný od pozorovatele platí, že i v tomto případě vysíláme stínové paprsky směrem ke světelnému zdroji, ke zjištění stínění a vlivu světla na tento bod.

Poté, co vypočteme vliv mezi výše zmíněnými průsečíky, připočteme vliv radiance na barvu pixelu, skrze který prochází paprsek od pozorovatele, a všechny pixely, skrze které projde paprsek od průsečíků y_j směrem k pozorovateli.

Ačkoliv je tato metoda schopna vytvářet kaustiky i mnohem efektivněji než v předchozích metodách, stále není příliš vhodná na simulování tohoto jevu, jelikož pro uspokojivé výsledky je potřeba počítat s velmi mnoha paprsky [11].

5.3.5 Photon mapping

S touto metodou přišel v roce 2001 Henrik W. Jensen [35] a stejně jako u metody bidirectional path tracing se i zde využívá generování paprsků z místa pozorování i od světelných zdrojů. Tento algoritmus je navíc rozdělen na dvě části, přičemž v prvním kroku se nejdříve vygeneruje tzv. *Photon map (fotonová mapa)* a ve druhém se vytváří výsledný obraz. Photon mapping je vhodý k simulaci složitějších světelných interakcí, zejména pak k vytváření věrohodné kaustiky.

První část algoritmu

V první části se ze všech světelných zdrojů náhodně vystřelují fotony s určitou energií, danou výkonem světelného zdroje a celkovým počtem vyzářených fotonů. Každý foton je následně sledován a vyhodnocuje se jeho průsečík s objekty ve scéně. Pokud je tento průsečík nalezen, uložíme jeho pozici, směr dopadu a energii fotonu do fotonové mapy. Následně pomocí metody ruské rulety vyhodnotíme, zda-li bude foton odražen, absorbován nebo jestli se jeho paprsek bude lomit. V případě že tento foton nezanikne, bude jeho další směr určen náhodně pomocí pravděpodobnostní funkce BRDF materiálu, na který

foton dopadl. Je nutné podotknout, že energie fotonu se v tomto případě nemění a bude stále stejná.

Často se vytvářejí dvě takovéto mapy – globální a kaustická, kde globální mapa obsahuje především odražené fotony a kaustická ty fotony, které procházejí skrz průhledné objekty a pomáhají vytvářet kaustiky. Jako datová struktura pro ukládání informací o fotonech je doporučen kd-strom. Takto uložené fotony jsou pak použity pro výpočet především nepřímého osvětlení, jelikož pro výpočet přímého osvětlení by bylo potřeba velkého množství počátečních fotonů.

Druhá část algoritmu

Druhý krok odpovídá algoritmu path tracing, který je mírně modifikován a využívá uložené fotonové mapy. Ty se aplikují při výpočtu osvětlení v každém nalezeném průsečíku, kdy se hledá N nejbližších fotonů danému místu a z jejich hodnot se vypočítá příspěvek k osvětlení. Jelikož je dána pouze energie fotonů, potřebujeme vypočítat odhad výsledné radiance směrem k pozorovateli. K tomu využijeme metody Monte Carlo a vyjádříme tak odchozí radianci jako sumu N nejbližších fotonů:

$$L_r(x, \vec{\omega}) \approx \frac{1}{\pi r^2} \sum_{p=1}^{N} f(x, \vec{\omega}, \overrightarrow{\omega_p}) \Delta\Phi_p(x, \overrightarrow{\omega_p}) \ ,$$

kde πr^2 představuje plochu základny polokoule nad bodem x, ve které se hledají dané fotony, $\vec{\omega}$ je směr k pozorovateli, $\overrightarrow{\omega_p}$ je směr dopadu fotonu, $f(x, \vec{\omega}, \overrightarrow{\omega_p})$ je BRDF zkoumaného povrchu a Φ_p značí světelný tok(energii) fotonu. Nutno podotknout, že tento postup platí především pro difúzní povrchy, pro výpočet spekulárních povrchů může být velmi nepřesný a je tedy vhodnější využít původního path tracing algoritmu.

5.3.6 Radiozita

Radiozita vznikla jako první pokus o vytvoření metody, která bude korektně a v dostatečné kvalitě simulovat šíření světla scénou. Je inspirována výpočtem tepelného záření a vznikla na půdě Cornellské univerzity v roce 1984 [36].

Základní algoritmus této metody vychází ze zákona o zachování energie a předpokládá síření světelného záření v energeticky uzavřené scéně a pracuje pouze s difúzními povrchy. Omezení na difúzní povrchy je z toho důvodu, že algoritmus předpokládá, že hodnota radiance odražené z určitého bodu je konstantní ve všech směrech, z čehož plyne, že je tento algoritmus nezávislý na směru pozorování.

Algoritmus radiozity předpokládá, že energie bude na celé ploše konstantní, proto se na počátku všechny plochy ve scéně rozdělí na menší plošky zvané *patches* z důvodu lepší aproximace zobrazovací rovnice. Pro každou dvojici

plošek je následně vyhodnocena jejich vzájemná poloha, zvaná jako *form factor*. Pokud se mezi dvěma ploškami nachází nějaká překážka, je tato hodnota zmenšena a v případě úplného zakrytí bude nulová. Následně je tato hodnota spolu s vlastnostmi difúzních materiálů využita k výpočtu radiozity určité plošky [11].

Mějme nyní rovnici pro výpočet radiozity [36]:

$$B(x) = E(x) + \rho(x) \int_S B(y)G(x,y)dy \ ,$$

kde:

- $B(x)$ je radiozita vyzařovaná povrchem v bodě x,
- $E(x)$ je vlastní vyzářovaná radiozita z povrchu v bodě x,
- $\rho(x)$ udává difúzní odrazivost plochy v daném bodě x,
- $G(x,y)$ je geometrický člen, udávající vzájemnou polohu dvojice plošek, tedy výše zmíněný form factor,
- S je plocha scény
- celý integrál $\int_S B(y)G(x,y)dy$ představuje příspěvek radiozity ostatních plošek odražené z bodu x dále do scény.

Jelikož je tato rovnice analyticky neřešitelná, je celá scéna rozdělena do rovinných plošek, jak bylo zmíněno výše. Tuto rovnici tak můžeme zjednodušit a vyjádřit následovně [37]:

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \ ,$$

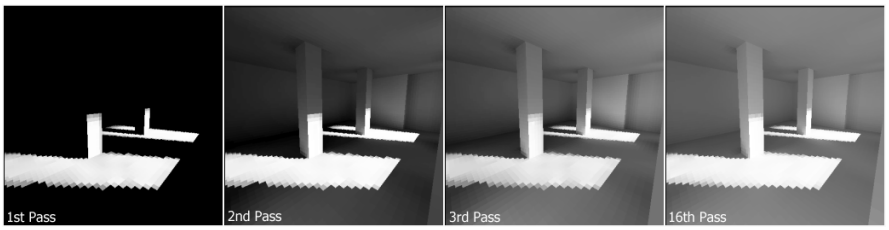
kde:

- B_i je radiozita plošky i,
- E_i je vlastní radiozita vyzařovaná ploškou i,
- F_{ij} je *form factor* mezi ploškami i a j,
- suma $\sum_{j=1}^{n} B_j F_{ij}$ reprezentuje součet radiozit všech plošek ve scéně příchozích na plošku i.

Z výše uvedeného vzorce si můžeme odvodit, že čím větší počet plošek, tím přesnější a lépe vypadající je výsledný obraz. To v3e ale za cenu velkého výkonu potřebného pro výpočet.

Řešení výše zmíněné rovnice i tak zůstává numericky náročné z důvodu závislosti jednotlivých radiozit mezi ploškami. Proto se často využívá tzv.

progresivní radiozita jejíž autory jsou pánové Cohen, Wallace a Grennberg [38]. Tato metoda spočívá v iterativním přístupu, kde se v každém kroku radiozita vystřeluje z plošky, která má největší energii (světelné zdroje). Následně je vypočítán form factor mezi touto ploškou a všemi ostatními ploškami ve scéně. Pokud je form factor nenulový, je následně spočtena radiance pro tyto plošky, které se pro další krok stanou zdroji světla. Tento postup se opakuje, dokud není výsledek uspokojivý.



Obrázek 5.12: Ukázka postupu algoritmu progresivní radiozity [38].

Na obrázku 5.12 můžeme vidět, jak světlo s každou iterací algoritmu postupuje od světelných zdrojů skrze scénu.

Po ukončení výpočtu radiozity je výsledek zobrazován například pomocí ray tracingu, který je schopen zobrazit i jiné, než difúzní povrchy. Výhodou tohoto postupu je, že můžeme libovolně měnit místo pozorování, ale nemusíme opakovat výpočet radiozity, jelikož je spočtena pro celou scénu.

Část II

Realizace

Kapitola 6

Implementace

Tato část textu se věnuje implementaci teoretických poznatků a rozšíření teoretických základů určitých metod využívaných při implementaci metody path-tracing.

6.1 Hledání průsečíků

Jedním z nejdůležitějších úkolu renderovacího softwaru je hledání průsečíků mezi paprskem a objekty. Těchto operací se za běhu programu obvykle provádí v řádech milionů, v závislosti na rozlišení výsledného obrazu. Z toho důvodu je třeba klást důraz na rychlost těchto operací, kde si mnohdy nevystačíme s obyčejným analytickým řešením.

Průsečík parpsku s trojúhelníkem

V počítačové grafice je většina modelů tvořena pomocí trojúhelníků. Z tohoto důvodu byla snaha vymyslet rychlejší algoritmy pro hledání průsečíku paprsku s trojúhelníkem, než ten, který byl představen v úvodní kapitole tohoto textu. Jedním z takovýchto algoritmů je právě *Möller-Trumbornův* algoritmus [39]. Tento algoritmus se snaží minimalizovat počet operací potřebných k najití průsečíku za pomoci lineární algebry a barycentrických souřadnic.

Pro ty platí, že libovolný bod $\boldsymbol{p}$ ležící uvnitř trojúhelníku určeném body $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ lze vyjádřit jako

$$\boldsymbol{p} = x\boldsymbol{a} + y\boldsymbol{b} + z\boldsymbol{c} \ ,$$

kde platí, že $x + y + z = 1$ a x, y, z značí barycentrické souřadnice. Mnohem častěji se však setkáme s tím, že jsou jednotlivé souřadnice vyjádřeny jako:

$$\begin{aligned} x &= 1 - u - v \ , \\ y &= u \ , \\ z &= v \ , \end{aligned}$$

a bod $\boldsymbol{p}$ je tak vyjádřen jako:

$$\boldsymbol{p} = (1 - u - v)\boldsymbol{a} + u\boldsymbol{b} + v\boldsymbol{c} \ .$$

Pro průsečík paprsku s rovinou, ve které leží trojúhelník $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, pak platí:

$$\boldsymbol{o} + t\vec{d} = \boldsymbol{a} + u(\overrightarrow{\boldsymbol{b} - \boldsymbol{a}}) + v(\overrightarrow{\boldsymbol{c} - \boldsymbol{a}}) \ .$$

Aby průsečík ležel uvnitř trojúhelníku, musí být splněna podmínka $u + v \leq 1$ a $u, v \geq 0$. Výše zmíněná rovnice lze zapsat i v maticovém tvaru:

$$\left[-\vec{d} \quad (\overrightarrow{\boldsymbol{b} - \boldsymbol{a}}) \quad (\overrightarrow{\boldsymbol{c} - \boldsymbol{a}})\right] \cdot \begin{bmatrix} t \\ u \\ v \end{bmatrix} = \overrightarrow{\boldsymbol{o} - \boldsymbol{a}} \ .$$

Tato soustavu tří rovnic o třech neznámých je v *Möller-Trumbornově* algoritmu řešena pomocí *Cramerova pravidla*, pro které platí:

$$A\boldsymbol{x} = \boldsymbol{b}, \ x_i = \frac{\det A_i}{\det A} \ ,$$

kde A musí být regulární matice a A_i značí matici A, kde byl i-tý sloupec nahrazen vektorem $\boldsymbol{b}$.

Tento algoritmus dále využívá pravidla, podle kterého lze determinant 3×3 matice vypočítat pomocí skalárního a vektorového součinu:

$$\det\left[\vec{a} \quad \vec{b} \quad \vec{c}\right] = \vec{a} \cdot (\vec{b} \times \vec{c}) \ .$$

Výsledný algoritmus [40] pak využívá minimum možných operací a jeho implementace je uvedena na následující straně.

Algoritmus 1: Möller-Trumbornův algoritmus

```
Function TriangleIntersection(a, b, c, o d):
    e1 = b - a
    e2 = c - a
    p = d × e2
    det = e1 · p
    if det ≤ ϵ then
        return false
    detInv = 1 / det
    q = o - a
    u = (q · p)detInv
    if u < 0 ∨ u > 1 then
        return false
    r = q × e1
    v = (r · d) if v < 0 ∨ u+v > 1 then
        return false
    t = (e2 · r)detInv
    return true
```

Průsečík BVH a paprsku

Další důležitou operací je hledání průsečíku paprsku s ohraničením objektu ($AABB$). Tato ohraničení se používají kvůli jejich schopnosti urychlit výpočty průsečíků, jelikož v případě, že paprsek dané ohraničení neprotíná, můžeme z dalších výpočtů vyřadit všechny objekty v něm obsažené. Tohoto faktu využívá například BVH strom viz sekce 2.4.

Výpočet průsečíku mezi paprskem a ohraničením je navíc rychlejší, než například výpočet průsečíku mezi paprskem a trojúhelníkem. Tato ohraničení jsou definována dvěma body, které určují jeho minimální a maximální souřadnice. Efektivní řešení tohoto problému se nazývá *slab method* a bylo navrženo v roce 1986 pány T. L. Kayou a J. T. Kajiyaou [41]. Toto řešení využívá faktu, že ohraničení je reprezentováno osově zarovnaným obdélníkem.

Strany tohoto obdélníku můžeme považovat jako tři dvojice rovnoběžných rovin, pomocí nichž postupně ořezáváme paprsek. Pokud zbude alespoň část takto ořezaného paprsku, musí nutně ležet uvnitř obdélníku a tím potvrdíme či vyvrátíme existenci průsečíku [42].

Mějme body $\boldsymbol{pMin}$ a $\boldsymbol{pMax}$, určující obdélník a paprsek $\boldsymbol{o} + t\vec{d}$. Algoritmus pro zjištění průniku takového paprsku s daným obdélníkem můžeme vidět na další stránce.

Algoritmus 2: Algoritmus pro zjištění průsečíku paprsku s obdélníkem

Function `AABBIntersection`(*pMin, pMax, o, d*)**:**

```
tx1 = (pMin.x - d.x)/d.x
tx2 = (pMax.x - d.x)/d.x

tmin = min(tx1,tx2)
tmax = max(tx1,tx2)

ty1 = (pMin.y - d.y)/d.y
ty2 = (pMax.y - d.y)/d.y

tmin = max(tmin, min(ty1,ty2))
tmax = min(tmax, max(ty1,ty2)))

tz1 = (pMin.z - d.z)/d.z
tz2 = (pMax.z - d.z)/d.z

tmin = max(tmin, min(tz1,tz2))
tmax = min(tmax, max(tz1,tz2)))

return tmax ≥ tmin
```

6.2 Aplikace metody Monte Carlo

Jak již bylo řečeno v úvodní kapitole, metoda Monte Carlo je v počítačové grafice hojně využívaná díky svým vlastnostem dostatečně dobře aproximovat výsledky integrálů. V případě algoritmů globálního osvětlení slouží k řešení zobrazovací rovnice 5.1 uvedené dříve.

V případě, že chceme vyhodnotit světlo, které se z určitého bodu odráží směrem k místu pozorování, potřebujeme nejdříve pro tento bod vyhodnotit přicházející radianci ze všech možných směrů a spočítat BRDF funkci pro příchozí směr $\vec{w_i}$ a odchozí směr $\vec{w_o}$. To ale vyžaduje výpočet integrálu nad hemisférou obklopující daný bod.

Tento problém je tedy vhodné řešit pomocí metody Monte Carlo, kde se budeme snažit odhadnout hodnotu rovnice odrazu světla:

$$L_o(x, \vec{\omega}) = \int_\Omega f(x, \vec{\omega}, \vec{\omega_i}) L_i(x, \vec{\omega_i}) \cos\theta d\omega_i \ .$$

Vyjádření této rovnice pomocí Monte Carlo metody vypadá následovně [2]:

$$L_o(x, \vec{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x, \vec{\omega}, \vec{\omega_i}) L_i(x, \vec{\omega_i}) |\cos\theta|}{p(\vec{\omega_i})} \ ,$$

kde $p(\overrightarrow{\omega_i})$ značí hustotu pravděpodobnosti, že je bod osvětlen ze směru $\overrightarrow{\omega_i}$ a $L_i(x, \overrightarrow{\omega_i})$ je hodnota příchozí radiance. Směr, který bude sledován je volen náhodně vzhledem k celé polokouli a je sledován do doby, než narazí na světelný zdroj.

Výše zmíněný postup však není příliš optimální. Jako příklad si můžeme uvést scénu, kde je většina osvětlení tvořena pomocí malých plošných zdrojů. Pravděpodobnost, že náhodně generovaný paprsek na takovýto světelný zdroj narazí, je velice malá a hloubka rekurze může být velmi velká, což má za následek téměř nulový světelný příspěvek v původním bodu. Z těchto důvodů by bylo potřeba vygenerovat velmi mnoho paprsků, aby výsledný odhad osvětlení dostatečně konvergoval k reálné hodnotě [2].

V praxi se tak tento problém rozděluje do dvou částí – přímé a nepřímé osvětlení. Příspěvek přímého osvětlení lze vyjádřit jako integrál [2]:

$$\int_{S^2} f(x, \overrightarrow{\omega}, \overrightarrow{\omega_i}) L_d(x, \overrightarrow{\omega_i}) \cos\theta d\omega_i \ ,$$

kde $L_d(x, \overrightarrow{\omega_i})$ značí světlo dopadající do bodu x ze všech světelných zdrojů. Tento integrál lze následně upravit jako součet příspěvků všech světelných zdrojů nacházejících se ve scéně:

$$\sum_{j=1}^{n} \int_{S^2} f(x, \overrightarrow{\omega}, \overrightarrow{\omega_i}) L_{d_j}(x, \overrightarrow{\omega_i}) \cos\theta d\omega_i \ ,$$

kde $L_{d_j}(x, \overrightarrow{\omega_i})$ značí radianci dopadající do bodu x z j-tého světelného zdroje ze směru $\overrightarrow{\omega_i}$ a platí:

$$L_d(x, \overrightarrow{\omega}) = \sum_{i=1}^{N} L_{d_j}(x, \overrightarrow{\omega}) \ .$$

Následně můžeme postupovat dvěma způsoby – aproximovat příspěvek všech světelných zdrojů a výsledek sečíst a nebo náhodně zvolit jeden světelný zdroj a výsledek upravit váhou, odpovídající pravděpodobnosti výběru daného zdroje [2].

Pro výpočet přímého osvětlení pak po úpravě pomocí metody Monte Carlo platí:

$$L_o(x, \overrightarrow{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x, \overrightarrow{\omega}, \overrightarrow{\omega_r}) L_d(x, \overrightarrow{\omega_r}) |\cos\theta|}{p(\overrightarrow{\omega_r})} \ ,$$

kde $\overrightarrow{\omega_r}$ je náhodně vybraný směr, ze kterého je bod x ozářen světelným zdrojem a $p(\overrightarrow{\omega_r})$ je hustota pravděpodobnosti výběru tohoto směru.

Výsledná rovnice pro odhad odraženého světla z určitého bodu je tak dána jako součet přímého a nepřímého osvětlení a vypadá následovně:

$$L_o(x, \overrightarrow{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x, \overrightarrow{\omega}, \overrightarrow{\omega_r}) L_d(x, \overrightarrow{\omega_r}) |\cos\theta|}{p(\overrightarrow{\omega_r})} + \frac{f(x, \overrightarrow{\omega}, \overrightarrow{\omega_i}) L_i(x, \overrightarrow{\omega_i}) |\cos\theta|}{p(\overrightarrow{\omega_i})} \ .$$

6.3 Sampling, importance sampling a multiple importance sampling

Řešení předchozí sekce přineslo další problém, který je nutné vyřešit. Jedná se o náhodné vybírání směrů a bodů. K tomuto existuje několik různých přístupů, z nichž některé dokáží snížit počet generovaných vzorků nutných k tomu, aby výpočet integrálu globálního osvětlení konvergoval rychleji ke správné hodnotě.

Naivním řešením generování náhodných bodů a směrů je použití uniformního rozdělení pravděpodobností. V tomto případě má každý bod na povrchu tělesa pravděpodobnost:

$$\frac{1}{A}$$

kde A je plocha povrchu. Pravděpodobnost libovolného směru určeného jednotkovou polokoulí je pak:

$$\frac{1}{2\pi} \ .$$

Lepším přístupem je však náhodně generovat takové směry a body, které mají největší vliv na odhad výsledné hodnoty zobrazovací rovnice. Tomuto přístupu se říká *importance sampling* a umožňuje lepší odhad pomocí méně vzorků. Tento fakt je důležitý obzvlášť pří výpočtu globálního osvětlení, jelikož snižuje počet paprsků nutných k vytvoření obstojně vypadajícího výsledku.

V případě aproximace zobrazovací rovnice je vhodné generovat náhodné směry vzhledem k funkci BRDF a k umístění světelných zdrojů. Pro příklad si můžeme uvést, že pokud světlo dopadá na povrch nějakého objektu téměř kolmo vůči jeho normále, je příspěvek tohoto světla k výsledku minimální, jelikož kosinus úhlu mezi příchozím směrem a normálou, kterým násobíme v zobrazovací rovnici, se bude blížit nule. Pro to je výhodné tyto směry zanedbat a ušetřit tak výpočetní výkon. Pravděpodobnost generování směrů, které favorizují směr bližší směru normály a se kterou svírají úhel θ je pak :

$$\frac{\cos\theta}{\pi}$$

a v zobecněném tvaru:

$$\frac{n+1}{2\pi}\cos^n\theta \ .$$

Jak již bylo řečeno, metoda Monte Carlo nám umožňuje odhadnout integrál funkce ve tvaru $\int f(x)dx$. Mnohdy se ale setkáme s funkcemi ve tvaru $\int f(x)g(x)dx$. V takovém případě jsme schopni použít vzorkování buď podle funkce $f(x)$ a nebo funkce $g(x)$. Nicméně není triviální určit, kterou z těchto dvou funkcí vybrat a mnohdy ani jedna nemusí přinášet dostatečně dobrý výsledek. S tímto problémem se setkáme například i při řešení přímého osvětlení, viz předchozí sekce.

K řešení této situace slouží *multiple importance sampling (MIS)*, který využívá obou distribučních funkcí a na základě určité heuristiky je vyvažuje. V takovém případě můžeme integrál funkce $\int f(x)g(x)dx$ odhadnout pomocí metody Monte Carlo a MIS jako:

$$\frac{1}{n_f}\sum_{i=1}^{n_f}\frac{f(X_i)g(X_i)w_f(X_i)}{p_f(X_i)}+\frac{1}{n_g}\sum_{j=1}^{n_g}\frac{f(Y_j)g(Y_j)w_g(Y_j)}{p_g(Y_j)} \quad ,$$

kde n_f je počet vzorků podle p_f distribuce, n_g je počet vzorků podle p_g distribuce a w_f a w_g představují onu vyvažovací heuristiku. Často je využívána tzv. *power heuristics* funkce, která lze vypočítat podle vzorce:

$$w_s(x)=\frac{(n_s p_s(x))}{\sum_i (n_i p_i(x))^\beta} \quad .$$

Pomocí výše zmíněných technik jsme nyní schopni při výpočtu přímého osvětlení použít vzorkování jak podle funkce BRDF, tak i podle jednotlivých světelných zdrojů. Takovýto postup vede k rychlejší konvergenci celého výpočtu a šetří výpočetní výkon.

Tato část textu vychází z poznatků v knížce *Physically based rendering: from theory to implementation* [2].

6.4 Generování náhodných bodů a směrů

Náhodné směry a body můžeme generovat podle jednoduchých algoritmů vycházejících z barycentrických a sférických souřadnic. Mějme náhodný uniformní generátor čísel na rozsahu $[0, 1]$, který vygeneroval dvě čísla u a v.

Pro vygenerování náhodného bodu uvnitř libovolného trojúhelníku $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ můžeme použít následující algoritmus:

Algoritmus 3: Algoritmus generování náhodných bodů na trojúhelníku

Function `PointOnTriangle(`*u, v, a, b, c*`):`

```
beta = 1 - sqrt(u)
gamma = (1 - beta)*v
alpha = 1 - beta - gamma
p = alpha*a + beta*b + gamma*c
return p
```

Pro generování náhodných bodů na kouli o poloměru r a ležící v počátku platí:

Algoritmus 4: Algoritmus generování náhodných bodů na kouli

Function `PointOnSpehere(`*u, v, r*`):`

```
a = 1 - 2*u
b = sqrt(1-a*a)
phi = 2*π*v
x = r*b*cos(phi)
y = r*b*sin(phi)
z = r*a
p = (x,y,z)
return p
```

Generování náhodného směru, který má větší váhu vzhledem k menší velikosti úhlu vůči normále:

Algoritmus 5: Algoritmus generování náhodných směrů na polokouli favorizující směr normály

Function `RandomCosineWeightedDirection(`*u, v*`):`

```
x = sqrt(u)*cos(2*π*v)
y = sqrt(u)*sin(2*π*v)
z = sqrt(1-u)
d = (x,y,z)
return d
```

Generování náhodného směru omezeného kuželem, jehož velikost je dána úhlem θ_{max}:

Algoritmus 6: Algoritmus generování náhodných směrů uvnitř kuželu

Function `RandomConeDirection(`*u, v,* θ_{max}`):`

```
cosTheta = (1-u) + u*θmax
sinTheta = sqrt(1 - cosTheta*cosTheta))
phi = v*2*π
x = cos(phi)*sinTheta
y = sin(phi)*sinTheta
z = cosTheta d = (x,y,z)
return d
```

Generování náhodného směru podle Phongova modelu spekulárního odrazu s exponentem s:

Algoritmus 7: Generování náhodného směru podle Phongova spekulárního modelu

Function `RandomPhongDirection(`*u, v, s*`):`

```
cosTheta = pow((1-u),1/(1+s))
sinTheta = sqrt(1 - cosTheta*cosTheta))
phi = v*2*π
x = cos(phi)*sinTheta
y = sin(phi)*sinTheta
z = cosTheta d = (x,y,z)
return d
```

Výše zmíněné algoritmy generují body a směry v tangentovém prostoru a je potřeba, aby před použitím byly správně transformovány vůči objektu, pro který se generují. Jedinou výjimkou je algoritmus pro generaci náhodného bodu v trojúhelníku.

Tyto algoritmy byly převzaty z kapitoly o vzorkování v knížce *Ray Tracing Gems* [43].

Kapitola 7

Popis implementace

Tato část práce je věnována popisu implementace jednoduchého rendereru založeném na metodě path tracing. Tato metoda byla zvolena z toho důvodu, že oproti běžnému ray tracing algoritmu využívá pokročilejší metody pro řešení zobrazovací rovnice. Dalším důvodem byla snaha se s těmito metodami více sblížit a tím položit základ dalšímu vzdělávání v oblasti počítačové grafiky a prohloubit stávající znalosti.

Práce obsahuje akcelerační strukturu BVH strom (AABB strom), multiple importance sampling a podporuje renderování za pomoci více vláken. Zhodnocení těchto optimalizačních prvků je v závěrečné kapitole.

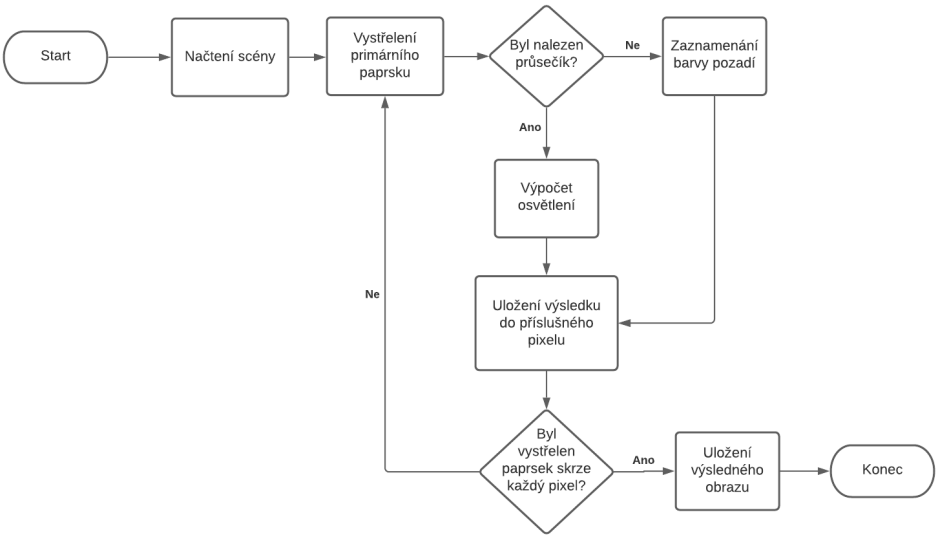
7.1 Volba technologií

V první fázi byla potřeba zvolit vhodné technologie. Volba programovacího jazyka byla v tomto případě přímočará a byl zvolen programovací jazyk C++ a vývojářské prostředí Visual Studio 2019.

Jazyk C++ je nejrozšířenějším jazykem v oblasti počítačové grafiky díky jeho rychlosti a správě paměti. A jelikož největšími zkušenostmi disponuji právě v tomto jazyce, byla volba v tomto případě jasná. Prostředí Visual Studio jsem rovněž zvolil z důvodu již předchozí zkušenosti a také kvůli nástrojům pro debugování a snadný vývoj.

7.2 Analýza a Návrh

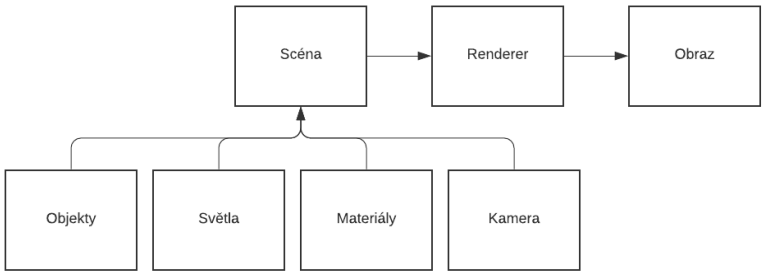
V další fázi byla potřeba vytvořit architekturu renderovacího systému. Renderovací proces si můžeme zjednodušeně naznačit v obrázku 7.1.



Obrázek 7.1: Postup renderovacího systému

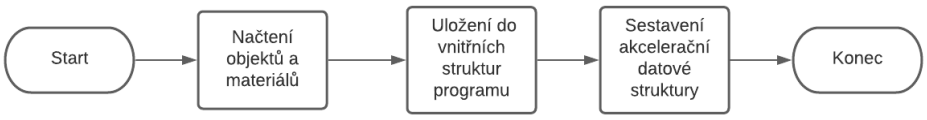
Tento diagram znázorňuje základní postup renderovacího systému a každý z procesů, který je zde znázorněn, představuje určitou problematiku, kterou lze rozdělit do menších podproblému.

S touto myšlenkou byl vytvořen i návrh, který systém rozděluje do menších částí a jehož schéma můžeme vidět v diagramu 7.2 níže.

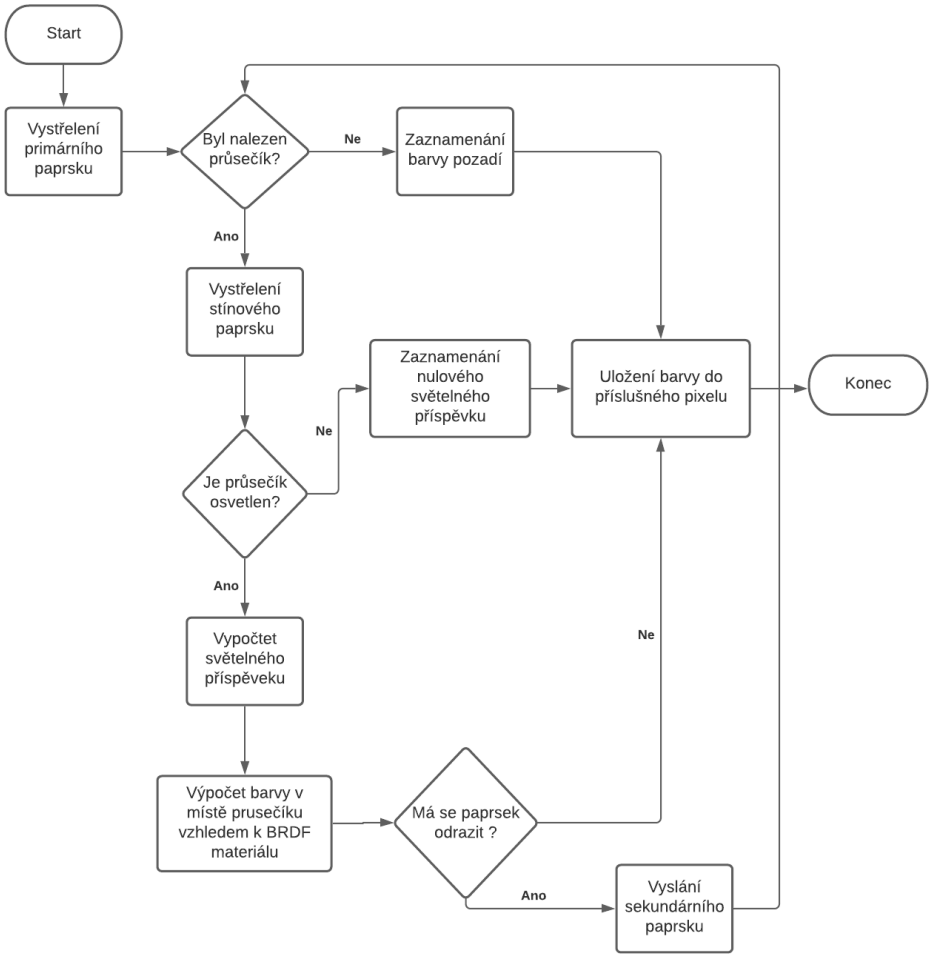


Obrázek 7.2: Blokové schéma návrhu

V tomto schématu hraje hlavní roli reprezentace scény a samotný renderer, který se scénou pracuje. Jejich jednotlivé procesy lze naznačit v obrázku 7.3, respektive 7.4.



Obrázek 7.3: Postup při načítání scény



Obrázek 7.4: Postup renderování

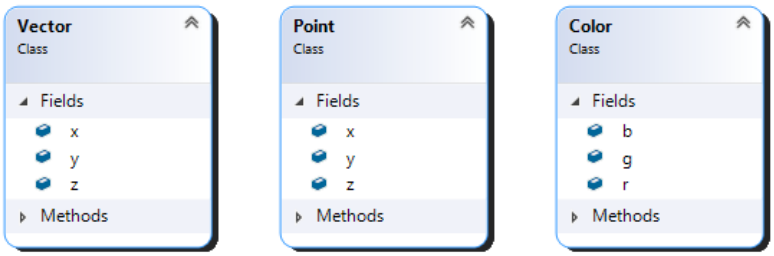
Z tohoto důvodu bylo cílem vytvořit co nejvíce modulární řešení, které lze jednoduše rozšířit. Důraz byl kladen na abstrakci nejdůležitějších částí, zejména reprezentaci objektů, materiálů a světelných zdrojů. Tyto části hrají důležitou roli při výsledném výpočtu osvětlení a je důležité brát v potaz jejich různorodost.

7.3 Základní struktury

Ze všeho nejdříve byla potřeba implementovat základní třídy, které se následně využívají napříč celým programem. Jedná se zejména o aplikaci matematických základů popsaných v první kapitole této práce – vektor, bod, paprsek a matematické operace s nimi pracující.

Mezi stěžejní třídy této části implementace a celého řešení tak patří:

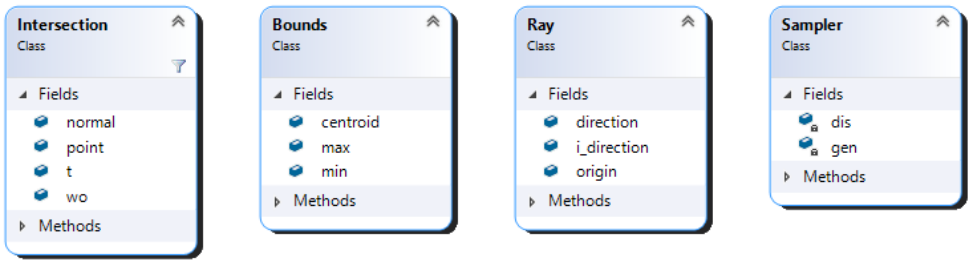
- `Vector` – jedná se o reprezentaci matematického vektoru ve 3D a všech příslušných operací nad vektory.
- `Point` – tato třída reprezentuje bod ve 3D a implementuje operace mezi dvěma body a bodem a vektorem.
- `Color` – třída `Color` reprezentuje barvu v RGB barevném prostoru a implementuje funkce pro práci s nimi.



Obrázek 7.5: Tři základní třídy systému

Za pomoci různých kombinací objektů výše zmíněných tříd je následně implementována většina zbylých tříd v této části, viz následující seznam.

- `Ray` – reprezentace paprsku, který je definován počátečním bodem a jeho směrem. Místo směru lze použít i cílový bod paprsku.
- `Bounds` – tato třída reprezentuje kvádr, jehož stěny jsou kolmé na souřadnicové osy a je vyjádřen pomocí dvou rohových bodů.
- `Intersection` – představuje informace o průsečíku mezi objektem a paprskem. Obsahuje pozici nalezeného průsečíku, normálu vůči povrchu, parametr paprsku `t` (viz sekce 1.5) a směr $\overrightarrow{wo}$, který je opačný ke směru paprsku.
- `Sampler` – tato třída slouží čistě ke generování náhodných čísel a směrů podle určitých kritérií, viz sekce 6.4.



Obrázek 7.6: Ukázka tříd Ray, Bounds, Intersection a Sampler

7.4 Reprezentace objektů

Jak již bylo řečeno v úvodu této kapitoly, reprezentace objektů je v této práci důležité a všechny geometrické objekty musí implementovat abstraktní třídu `Primitive`. Ta určuje, jaké vlastnosti a funkce musí každá třída reprezentující objekt mít, aby její objekty mohly být zobrazeny ve výsledném renderu.

Tato abstraktní třída má následující rozhraní:

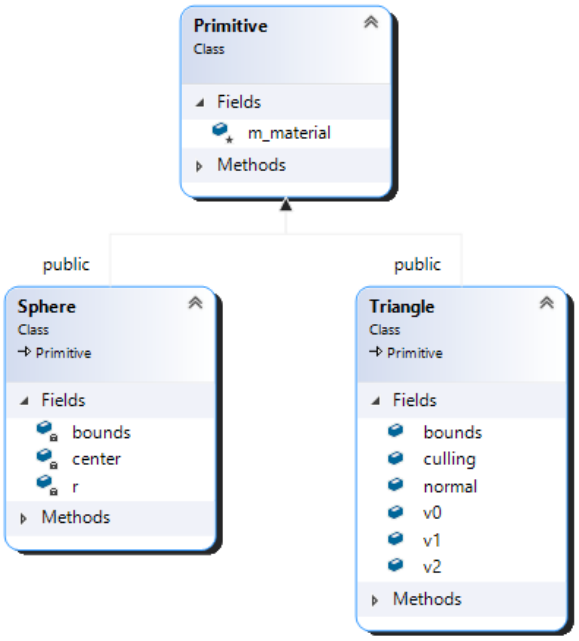
```
class Primitive{
public:
  Primitive(const std::shared_ptr<Material>& m_material);
  virtual bool intersect(const Ray& ray, Intersection& intersection) const =
     0;
  virtual bool boundsIntersect(const Ray& ray) const = 0;
  virtual Bounds getBounds() const = 0;
  virtual std::shared_ptr<Material> getMaterial() const;
  virtual double area() const = 0;
  virtual Intersection sample(double& pdf) const = 0;
  virtual double pdf() const;
  virtual Intersection sample(const Point& point, double& pdf);
  virtual double pdf(const Point& point, const Vector& wi);
  virtual Vector getNormal(const Point& point) const = 0;
protected:
  std::shared_ptr<Material> m_material;
};
```

kde jednotlivé funkce mají následující význam:

- `intersect` – má za úkol najít průsečík paprsku s objektem,
- `boundsIntersect` – má za úkol zjistit průsečík s ohraničením objektu, reprezentovaného třídou `Bounds`,
- `area` – vrátí velikost plochy objektu,
- `sample` – zde existují dvě implementace této funkce. První z nich vrátí náhodný bod na povrchu objektu, který byl vygenerovaný s uniformní pravděpodobností. Druhá implementace vrátí náhodný bod viditelný z místa pozorování s pravděpodobností určenou velikostí prostorového úhlu, který tento objekt vymezuje vzhledem k místu pozorování.

- `pdf` – obdobně jako u předchozí funkce existují dvě řešení. První z nich vrátí pravděpodobnost vygenerování náhodného bodu vůči ploše celého objektu. Druhá z nich vrátí pravděpodobnost vygenerování (přesněji řečeno hustotu pravděpodobnosti) tohoto bodu vzhledem k místu pozorování.

V této práci jsou implementovány dva různé typy objektů – trojúhelník a koule, které jsou reprezentovány třídami `Triangle` a `Sphere`. Obě tyto třídy mají jako atribut `bounds`, který popisuje ohraničení objektu. Koule je definována pomocí poloměru a bodu, který udává pozici jejího středu v prostoru. Trojúhelník je zde reprezentován třemi vertexy popsanými strukturou `Vertex`, která obsahuje informaci o pozici bodu a jeho normále. Třída `Triangle` následně slouží jako základ pro tvorbu složitějších modelů, viz sekce 2.1.



Obrázek 7.7: Ukázka tříd geometrických objektů.

7.5 Reprezentace materiálů

V předchozí sekci jsme si mohli všimnout, že třída `Primitive` má jako atribut materiál. Ten je další důležitou částí implementace.

Třída `Material` slouží jako kompozice jednotlivých BRDF funkcí, kde vlastnosti materiálu udáváme právě pomocí nich. Z tohoto důvodu je zde představena další abstraktní třída `BxDF`, jejíž rozhraní je popsáno na další stránce.

```
class BxDF{
public:
  BxDF(BxDFType type) : type(type){}
  virtual Color f(const Intersection& it, const Vector& wi, const Vector& wo
    ) const = 0;
  virtual Color sample(const Intersection& it, const Vector& wi, Vector& wo,
     double& pdf) const = 0;
  virtual double pdf(const Intersection& it, const Vector& wi, const Vector&
     wo) const = 0;
  virtual BxDFType getMaterialType() const { return type; }
  bool MatchesFlags(BxDFType t) const { return (type & t) == type; }
  BxDFType type;
};
```

Všechny třídy reprezentující vlastnosti materiálu musí dědit právě z této třídy a musí implementovat následující tři funkce:

- `f` – Tato funkce vypočte hodnotu odrazivosti materiálu v daném bodě pro daný incidentní směr $\vec{\omega_i}$ a odchozí směr $\vec{\omega_o}$.
- `sample` – Tato funkce náhodně zvolí jeden z možných směrů odrazu světla dopadajícího do daného bodu.
- `pdf` – Tato funkce vypočte hustotu pravděpodobnosti, že světlo dopadající do daného bodu ze směru $\vec{\omega_i}$ bude odraženo ve směru $\vec{\omega_i}$.

Tato třída má navíc atribut `type`, který pomocí `enum` struktury popisuje vlastnosti dané BRDF funkce. Například určuje, zda-li BRDF funkce popisuje lesklý odraz, difúzní odraz a nebo třeba lom světla.

V této práci jsou implementovány čtyři druhy BRDF funkcí, které dědí z této abstraktní třídy. Jedná se o Lambertův model difúzního odrazu, Phongův model lesklého odrazu, perfektní odraz (zrcadlo) a perfektní lom světla (založený na Snellově zákonu). Tyto třídy jsou znázorněny v diagramu na obrázku 7.8.

Díky tomuto návrhu mohou být jednoduše přidávány další BRDF funkce a tím i další druhy materiálů. Stačí pouze vytvořit novou třídu, která bude dědit z abstraktní třídy `BxDF` a bude implementovat všechny virtuální metody.

Následuje třída `Material`, která spojuje výše zmíněné BRDF funkce dohromady a díky tomu udává vzhled jednotlivých objektů. Tato třída má tři základní funkce: `f`, `sample`, a `pdf`. Ty plní stejnou roli, jako v případě třídy `BxDF`, pouze s tím rozdílem, že kombinuje vlastnosti těchto BRDF funkcí. Tím tak určuje finální podobu materiálu.

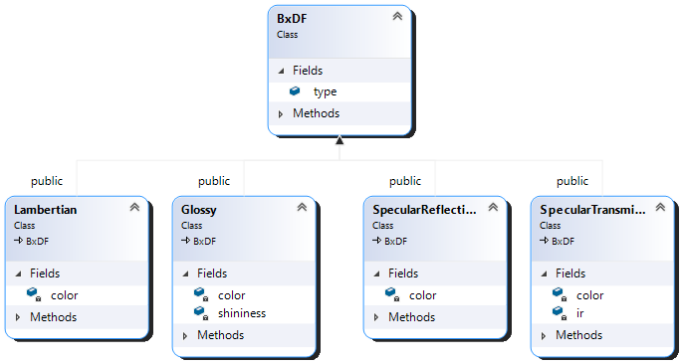
Je však možné těmto funkcím říct, se kterými vlastnostmi BRDF funkcí mají počítat a ignorovat tak ty BRDF funkce, které tyto vlastnosti nesplňují.

Tato třída má tři atributy:

- `m_BxDFs` – přestavuje kolekci všech BRDF funkcí popisujících daný materiál,
- `m_type` – popisuje vlastnosti daného materiálu,
- `m_emission` – určuje schopnost materiálu vyzařovat světlo a defaultně je nastaven jako nula.

Kromě toho tato třída přidává další jednoduché funkce pro práci s ní. Definici této třídy můžeme vidět na následujícím diagramu:

```
class Material{
public:
  Material(const Color& emission = Color(0.0));
  void addBxDF(std::shared_ptr<BxDF> bxdf);
    Color f(const Intersection& it, const Vector& wi, const Vector& wo,
    BxDFType type = BxDFType::ALL) const;
    bool sample(const Intersection& it, const Vector& wi, Vector& wo, Color
    & f, double & pdf, BxDFType type = BxDFType::ALL) const;
    double pdf(const Intersection& it, const Vector& wi, const Vector& wo,
    BxDFType type = BxDFType::ALL) const;
    Color emission() const;
  bool isEmissive() const;
  BxDFType getType() const;
  int NumOfComponents(BxDFType flags) const;
  int NumOfBxdfs() const;
private:
  std::vector<std::shared_ptr<BxDF>> m_BxDFs;
  BxDFType m_type;
  Color m_emission;
};
```



Obrázek 7.8: Diagram BxDF tříd.

7.6 Světelné zdroje

Další část implementace se věnuje světelným zdrojům. Stejně jako tomu bylo u předchozích částí, i zde se využívá abstraktní třída. Ta definuje základní funkcionalitu světelných zdrojů a nazývá se `Light`. Její rozhraní vypadá následovně:

```
class Light{
public:
  Light(const Color& color, double intensity, int nSamples = 1);
  virtual bool sample(const Point& point, Vector& wi, Color & Le, double&
    pdf, const Scene & scene) const = 0;
  virtual double pdf(const Point& point, const Vector& wi) const = 0;
  virtual bool Le(const Point & point, const Vector & wi, Color & Le, const
    Scene& scene) const = 0;
  virtual Color getColor() const;
  virtual double getIntensity() const;
  virtual int numberOfSamples() const;
protected:
  double m_intensity;
  Color m_color;
  int m_nSamples;
};
```

Z tohoto vyplývá, že každý druh světelného zdroje musí implementovat tyto následující tři funkce:

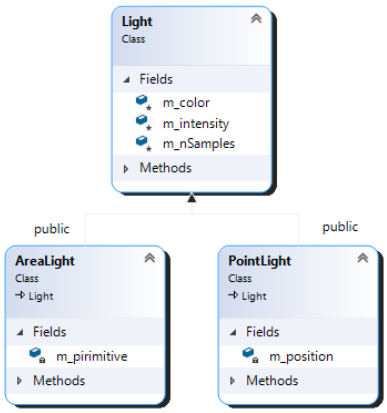
- `sample` – tato funkce má za úkol vzhledem k danému bodu náhodně zvolit směr, ze kterého bude osvětlen, určit zda takový směr existuje a zjistit, jestli je zvolený světelný paprsek blokován jiným objektem.
- `pdf` – tato funkce vrací hustotu pravděpodobnosti, že na daný bod dopadá paprsek světla ze směru $\vec{\omega_i}$ vyzářený z daným světelným zdrojem.
- `Le` – určuje hodnotu (barvu) světelného paprsku dopadajícího na daný bod ze směru $\vec{\omega_i}$. Nenulovou hodnotu vrací pouze v případě, že daný světelný zdroj je schopen v daném směru bod osvětlit a paprsek není blokován jiným objektem.

Tato práce se věnuje hlavně plošným světelným zdrojům, jelikož nejvěrněji imitují reálné světelné zdroje. Zároveň se pomocí nich dají vytvářet vizuálně realističtější obrázky. Tomu vděčí zejména jejich schopnosti vytvářet měkké stíny.

Plošný světelný zdroj je reprezentován třídou `AreaLight`. Každý objekt této třídy musí obsahovat informaci o geometrickém objektu, který světelný zdroj reprezentuje. V tomto případě se jedná o libovolnou třídu implementující abstraktní třídu `Primitive`. Objekt se stává plošným světelným zdrojem ve chvíli, kdy materiál, který popisuje jeho vlastnosti, obsahuje emitující složku.

Kromě plošného světelného zdroje je v této práci implementován také bodový zdroj světla. Ten je reprezentován pouze svojí polohou ve scéně. Všechny světelné zdroje pak sdílejí atributy, definované v abstraktní třídě `Light`:

- `m_intensity` – tento atribut určuje intenzitu světelného zdroje a většinou se odvíjí od scény, ve které se světelný zdroj nachází,
- `m_color` – určuje barvu světelného záření,
- `m_nSamples` – určuje počet vzorků, podle kterých má být vyhodnocen světelný příspěvek ve zkoumaném bodě. Tento atribut je využíván hlavně u plošných světelných zdrojů a může vylepšit vzhled stínů, ale za cenu náročnějšího výpočtu.



Obrázek 7.9: Diagram tříd představující světelné zdroje

7.7 Reprezentace scény

V této fázi byla potřeba spojit všechny předchozí části dohromady a vytvořit tak scénu. Ta obsahuje informace o všech objektech, které se v ní nacházejí a poskytuje funkcionalitu pro hledání průsečíků. Toho lze docílit naivní cestou a kontrolovat každý objekt s každým paprskem a nebo využít akcelerační datové struktury. V tomto případě volba padla na akcelerační datovou strukturu, konkrétně BVH strom.

Třída reprezentující BVH strom se jmenuje `BVHTree` a přináší jednoduchou funkcionalitu k jeho stavbě a následnému procházení. Jednotlivé vrcholy a listy stromu jsou reprezentovány strukturou `BVHNode`, která umožňuje sestavit nový vrchol v závislosti na tom, jestli se jedná o list nebo vnitřní uzel.

Samotný strom využívá rekurzivní algoritmus k jeho stavbě, který postupně rozděluje ohraničení celé scény na menší skupiny, dokud skupinu netvoří pouze jeden objekt a nebo více objektů s již dále nedělitelným ohraničením. V takovém případě vzniká nový list stromu, který obsahuje tyto objekty. Vnitřní uzly obsahují pouze informaci o ose dělení a celkovém ohraničení všech objektů obsažených v jeho potomcích.

Následné procházení tohoto stromu a hledaní průsečíků je rovněž prováděno rekurzivně, kde se nejprve kontroluje, jestli paprsek protíná ohraničení určité skupiny. V případě, že paprsek toto ohraničení neprotíná, můžeme z dalšího procházení vynechat celý podstrom a zrychlit tak celkové hledání. V opačném případě je potřeba se zanořit hlouběji a hledat průsečík v potomcích daného uzlu.

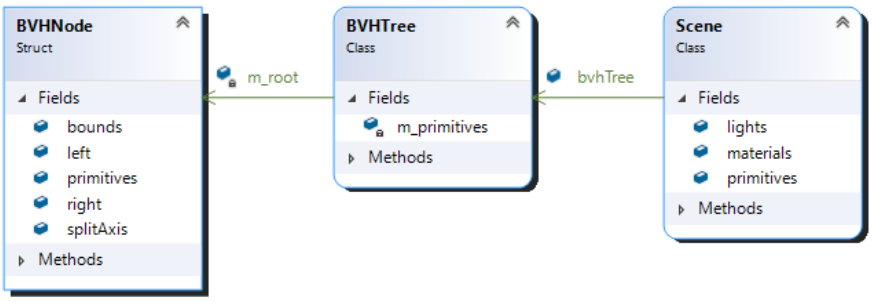
Třída představující scénu pak obsahuje tento strom, který je postaven ihned po načtení scény. Tato práce podporuje načítání souborů typu OBJ a využívá volně dostupné C++ knihovny, zvané *tinyobjloader* [44]. I přesto, že je v práci využit BVH strom, byla vytvořena i funkce pro naivní hledání průsečíku, z důvodu testování. Poslední funkcí kterou poskytuje třída `Scene` je funkce pro zjištění viditelnosti mezi dvěma body. Tato funkce je používána hlavně u stínových paprsků a počítání osvětlení. Celkové rozhraní této třídy můžeme vidět níže.

```
class Scene
{
public:
  bool intersection(const Ray& ray, Intersection& it) const;
  bool naiveIntersection(const Ray& ray, Intersection& it) const;
  bool isVisible(const Point & p0, const Point & p1) const;
  void loadScene(const std::string& path);
  std::vector<std::shared_ptr<Light>> lights;
  std::vector<std::shared_ptr<Primitive>> primitives;
  std::vector<std::shared_ptr<Material>> materials;
  BVHTree bvhTree;
};
```

Rozhraní samotného BVH stromu pak vypadá následovně:

```
struct BVHNode
{
  void initLeaf(std::vector<std::shared_ptr<Primitive>> prims, const Bounds&
     b);
  void initInternal(Axis axis, BVHNode* leftChild, BVHNode* rightChild);
  BVHNode* left, * right;
  std::vector<std::shared_ptr<Primitive>> primitives;
  Bounds bounds;
  Axis splitAxis;
};

class BVHTree
{
public:
  BVHTree();
  ~BVHTree();
  BVHTree(std::vector<std::shared_ptr<Primitive>> primitives);
  void build(std::vector<std::shared_ptr<Primitive>> primitives);
  bool intersect(const Ray& ray, Intersection& intersection) const;
private:
  bool hit(const Ray& ray, Intersection& intersection, BVHNode* node) const;
  BVHNode* recursiveBuild(std::vector<std::shared_ptr<Primitive>>&
    primitives, int start, int end);
  BVHNode* m_root;
  std::vector<std::shared_ptr<Primitive>> m_primitives;
};
```



Obrázek 7.10: Diagram scény

7.8 Kamera a obraz

Dále bylo nutné vytvořit třídu, která bude reprezentovat kameru a obraz, do kterého se bude ukládat výsledek renderu.

Kamera je reprezentována třídou `Camera`, vychází z teorie představené v sekci 2.2 a je určena pěti parametry:

- `width` – vyjadřuje šířku výsledného obrazu v pixelech,
- `height` – vyjadřuje výšku výsledného obrazu v pixelech,
- `position` – představuje bod, ze kterého je scéna pozorována,
- `lookAt` – vyjadřuje bod, na který je kamera zaměřena,
- `fov` – vyjadřuje úhel zorného pole.

Jedinou funkcionalitu, kterou tato třída obsahuje, je funkce pro generování paprsků procházejících zadaným pixelem pomocí převodu obrazového prostoru do prostoru scény.

Obraz je reprezentován třídou `Image` a využívá strukturu `Pixel`, jejíž objekty jsou ukládány do dvourozměrného pole pixelů, díky čemuž je vytvořena klasická mřížku pixelů.

Třída `Image` umožňuje zápis do jednotlivých pixelů a uložení výsledného obrazu ve formátu ppm do předem zadaného souboru. Velikost obrazu je pak určena parametry `width` a `height`, které odpovídají stejně pojmenovaným parametrům třídy `Camera`.

7.9 Renderer

Hlavní a poslední třídou je `Renderer`. Ta spojuje dohromady všechny dříve popsané části a je zodpovědná za běh celého programu, včetně výpočtu lokálního a globálního osvětlení. Její rozhraní vypadá následovně:

```
class Renderer
{
public:
  Renderer(const Camera& camera, const std::string& scenePath, const std::
    string & savePath);
  void render(int samples, int bounces, int threadsCount, RenderType flags =
     RenderType::INDIRECT);
private:
  void threadJob(int start, int end, int samples, int bounces, RenderType
    flags);
  void multiThreadRender(int samples, int bounces, int threadsCount,
    RenderType flags);
  double powerHeuristic(int nf, double fPdf, int ng, double gPdf) const;
  Color indirectIllumination(const Intersection& intersection, int bounces,
    RenderType flags) const;
  Color directIllumination(const Intersection& intersection, RenderType
    flags) const;
  Color directShading(const Intersection& intersection, const std::
    shared_ptr<const Light>& light) const;
  Color indirectShading(const Intersection& intersection, int bounces,
    RenderType flags) const;
  Color m_background;
  std::string m_scenePath, m_savePath;;
  Image m_image;
  Scene m_scene;
  Camera m_camera;
  std::vector<Block> m_blocks;
  std::mutex m_mutex;
  std::condition_variable m_cv;
  std::atomic<int> m_completedThreads;
  std::vector<std::thread> m_threads;
};
```

Můžeme si všimnout, že tato třída má kromě konstruktoru jen jednu veřejně přístupnou metodu `render`, která podle zadaných parametrů spouští celý proces renderování. Jejími parametry jsou

- `sample` – udává kolik paprsků má být vystřeleno skrze každý pixel,
- `bounces` – počet odrazů každého paprsku,
- `threadsCount` – počet vláken použitých na výpočet osvětlení,
- `lookAt` – vyjadřuje bod, na který je kamera zaměřena,
- `flags` – vyjadřuje nastavení renderovacího procesu pomocí enum třídy `RenderType`, viz následující stránka.

```
enum RenderType
{
  BASIC = 1 << 0,
  MULTIPLE_LIGHT_SAMPLING = 1 << 1,
  ALL_LIGHTS_SAMPLIG = 1 << 2,
  INDIRECT = 1 << 3,
  BEST = MULTIPLE_LIGHT_SAMPLING | ALL_LIGHTS_SAMPLIG | INDIRECT,
};
```

Z výše uvedené kódu lze vypozorovat, že renderovací proces má několik různých nastavení. Ty lze libovolně kombinovat a jednotlivé parametry mají následující význam:

- `BASIC` – renderovací proces počítá pouze s přímým osvětlením,
- `MULTIPLE_LIGHT_SAMPLING` – světelné zdroje jsou vzorkovány větším množstvím stínových paprsků,
- `ALL_LIGHTS_SAMPLIG` – vysílá stínové paprsky do každého světelného zdroje, namísto náhodné volby pouze jednoho z nich,
- `INDIRECT` – počítá i s nepřímým osvětlením,
- `BEST` – zapíná všechny funkce popsané výše.

Všechny zbylé funkce jsou privátní a slouží k výpočtu přímého a nepřímého osvětlení. Ty probíhají paralelně v několika vláknech, jejichž počet určuje výše zmíněný parametr `threadsCount` u funkce `render`. Výpočet osvětlení probíhá podle algoritmu path tracing za použití multiple importance sampling metody.

Kapitola 8

Výsledky a zhodnocení

Tato závěrečná kapitola slouží k zhodnocení vytvořeného programu a k porovnání různých metod renderování a srovnání optimalizačních prvků.

Veškeré testování probíhá na následující počítačové sestavě:

- CPU – AMD Ryzen 7 3700x s 8 jádry a 16 vlákny,
- GPU – NVIDIA RTX 3070,
- RAM – 32 GB, 3200 MHz.

8.1 Zrychlení výpočtu pomocí paralelizace

Tato sekce se věnuje tomu, jak velké rozdíly jsou ve výsledném čase potřebném k vyrenderování obrazu vzhledem k počtu použitých vláken.

Test byl proveden na klasické scéně známé jako *Cornell-Box*, která se skládá celkově z 36 trojúhelníků. Testování bylo provedeno s následujícím nastavením:

- 16 vzorků na pixel,
- 3 odrazy primárního paprsku (hloubka rekurze),
- rozlišení obrazu 800x800 pixelů,
- výpočet přímého i nepřímého osvětlení,
- vzorkování pouze jednoho světelného zdroje,
- jeden vzorek na světlo.

Výsledky lze vidět v následující tabulce:

Tabulka 8.1: Vliv počtu použitých vláken na dobu výpočtu

Počet vláken	Celkový čas	Procentuální snížení času oproti jednomu vláknu
1	1038 s	-
2	694 s	≈ 33,1%
4	223 s	≈ 78,5%
8	115 s	≈ 88,9%
16	71 s	≈ 93,2%

8.2 Zrychlení pomocí BVH stromu

Tato sekce se věnuje vlivu akcelerační datové struktury na dobu výpočtu. Porovnávání probíhá mezi naivním hledáním průsečíků, které spočívá v testování každého objektu ve scéně a BVH stromem.

Metodika měření spočívá ve vystřelování paprsků skrze každý pixel obrazu a následném hledání průsečíků. Nepočítá se tedy žádné osvětlení, čímž jsem se snažil omezit vliv ostatních výpočtů na výsledné měření.

Měření probíhá na několika scénách o různém počtu objektů a využita jsou všechna dostupná vlákna procesoru (16). Výsledky jsou zaznamenány v tabulce níže.

Tabulka 8.2: Srovnání časové náročnosti naivního řešení a BVH struktury

Počet objektů	Doba výpočtu pomocí naivního algoritmu	Doba výpočtu pomocí BVH stromu
32	2677 ms	2550 ms
224	3143 ms	2580 ms
992	4910 ms	2637 ms
4000	9582 ms	2822 ms
15904	25784 ms	2962 ms
63520	162229 ms	2871 ms
253984	773249 ms	2818 ms

Z tabulky můžeme vidět, že ačkoliv pro velmi malý počet objektů jsou oba algoritmy srovnatelné, pro větší počet objektů je rozdíl velmi markantní.

Celkový čas je také ovlivněn rozložením objektů ve scéně, kde BVH strom vyniká především ve scénách, kde jsou objekty rozmístěny dostatečně daleko od sebe a tvoří samostatné skupinky, které je možné rychle vyřadit z výpočtů.

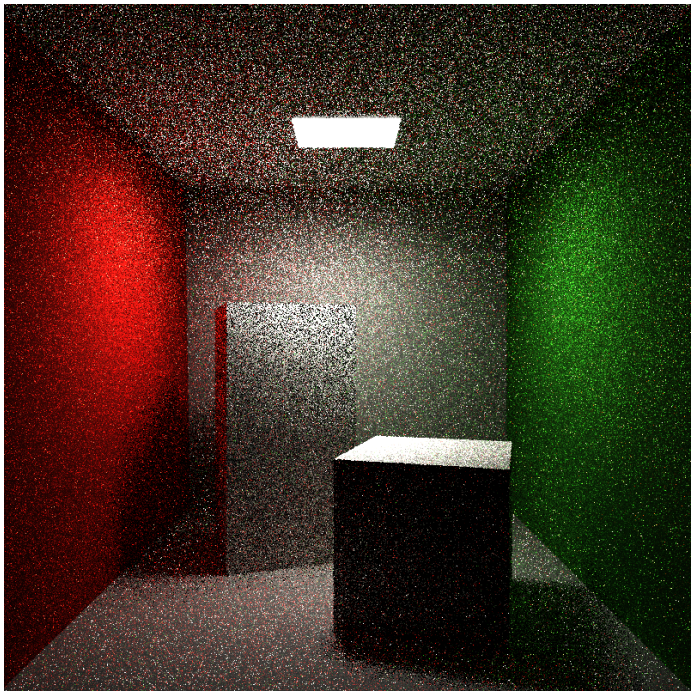
8.3 Ukázka práce a srovnání různých nastavení

Tato sekce slouží k ukázání výstupů algoritmu a k vizuálnímu srovnání různých nastavení.

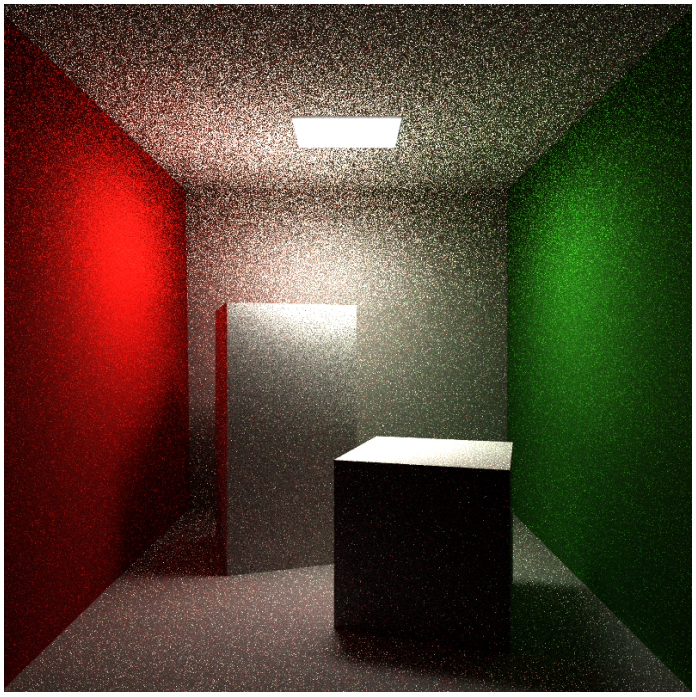
Vliv počtu vzorků na vzhled obrazu

Následující série obrázků slouží k ukázce toho, jaký vliv má počet vzorků na pixel (*spp* - samples per pixel) na výsledný vzhled scény. Tyto scény byly renderovány s následujícím nastavením:

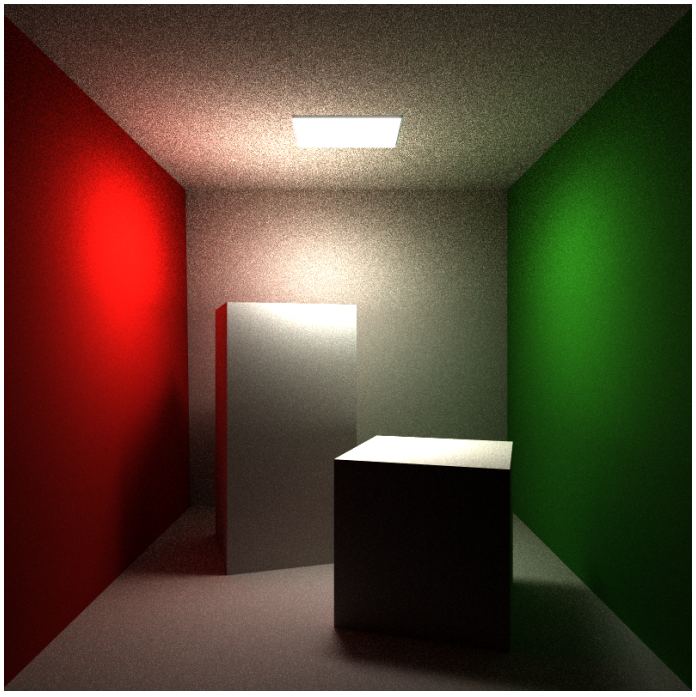
- 4 odrazy primárního paprsku (hloubka rekurze),
- rozlišení obrazu 800x800 pixelů,
- výpočet přímého i nepřímého osvětlení,
- vzorkování pouze jednoho světelného zdroje,
- jeden vzorek na světlo.



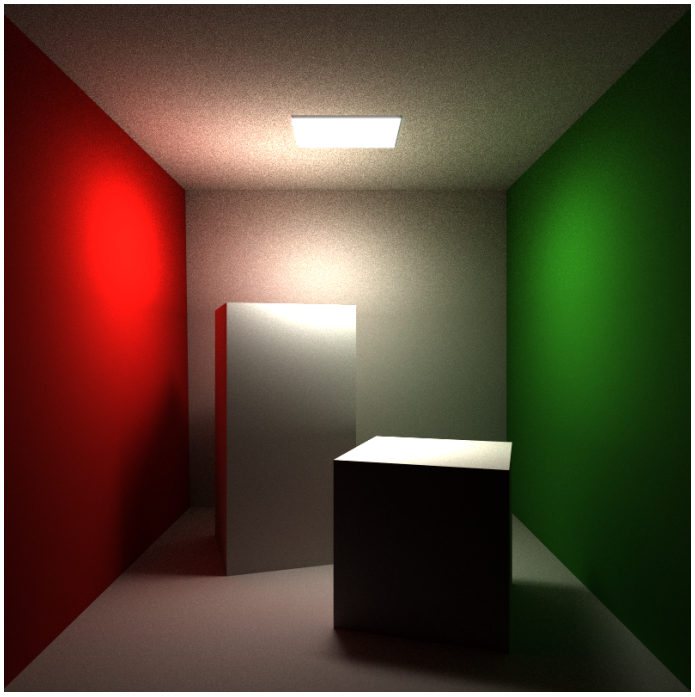
Obrázek 8.1: 1 vzorek na pixel



Obrázek 8.2: 16 vzorků na pixel



Obrázek 8.3: 256 vzorků na pixel



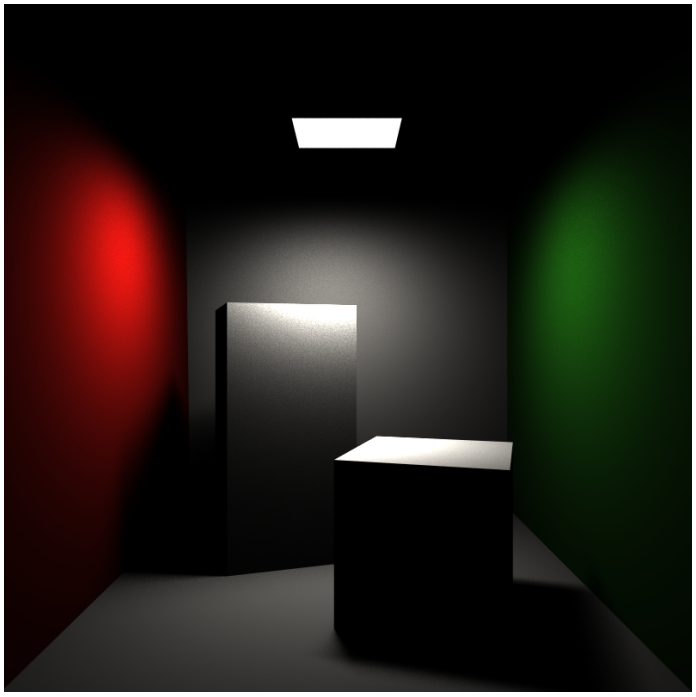
Obrázek 8.4: 1024 vzorků na pixel

Můžeme vidět, že čím více vzorků použijeme, tím více je redukován šum ve výsledném obrazu. To je dáno vlastnostmi Monte Carlo metody, pro kterou platí, že čím více vzorků použijeme, tím více konverguje odhad k skutečné hodnotě integrálu. Více vzorků tak vyvažuje nepřesnosti vzniklé při výpočtu a výsledkem je lépe vypadající obraz. Cenou je ale delší doba výpočtu. Pro srovnání, výpočet obrázku 8.1 trval zhruba 6 vteřin, zatímco výpočet snímku 8.4 trval téměř 89 minut.

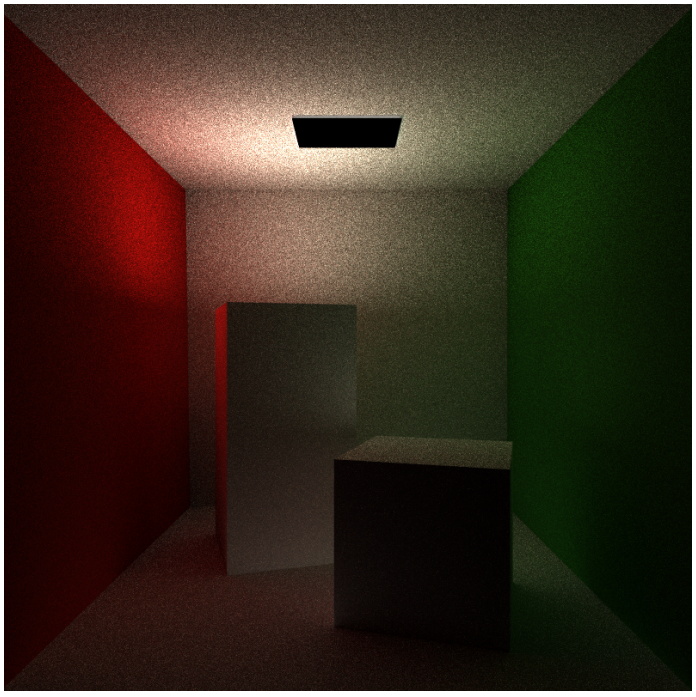
Rozdíl mezi přímým a nepřímým osvětlením

Tato sekce má za úkol srovnat vliv nepřímého osvětlení na výslednou podobu obrázku. Testovací parametry jsou uvedeny níže.

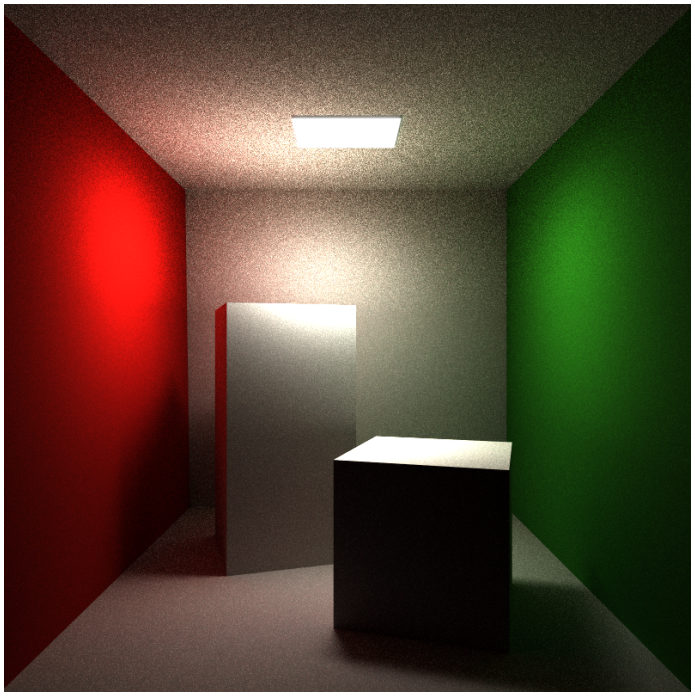
- 4 odrazy primárního paprsku (hloubka rekurze),
- rozlišení obrazu 800x800 pixelů,
- vzorkování pouze jednoho světelného zdroje,
- jeden vzorek na světlo,
- 256 vzorků na pixel.



Obrázek 8.5: Přímé osvětlení



Obrázek 8.6: Nepřímé osvětlení



Obrázek 8.7: Přímé a nepřímé osvětlení dohromady

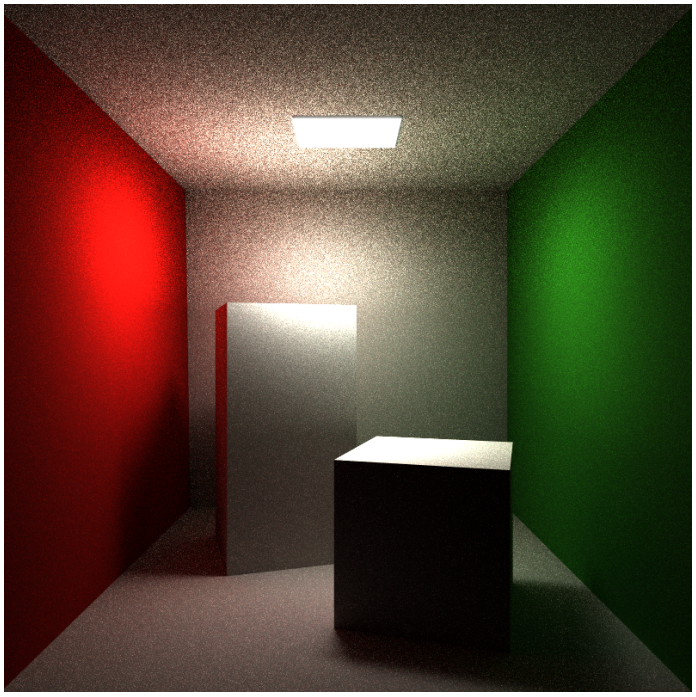
Výše můžeme vidět rozdíly mezi přímým a nepřímým osvětlením. V případě přímého osvětlení nejsme schopni vyhodnotit přenos barev mezi objekty a místa, která nejsou osvětlena přímo ze světelného zdroje, jsou zcela tmavá. Výhodou ale je, že výsledný obraz je téměř bez šumu.

Oproti tomu je možné výše zmíněné efekty vytvářet pomocí nepřímého osvětlení, ale za cenu většího množství paprsků. Ideálním řešením je tedy spojit oba zmíněné postupy, čímž zredukujeme čas potřebný k výpočtu a výsledný obraz vypadá věrohodněji.

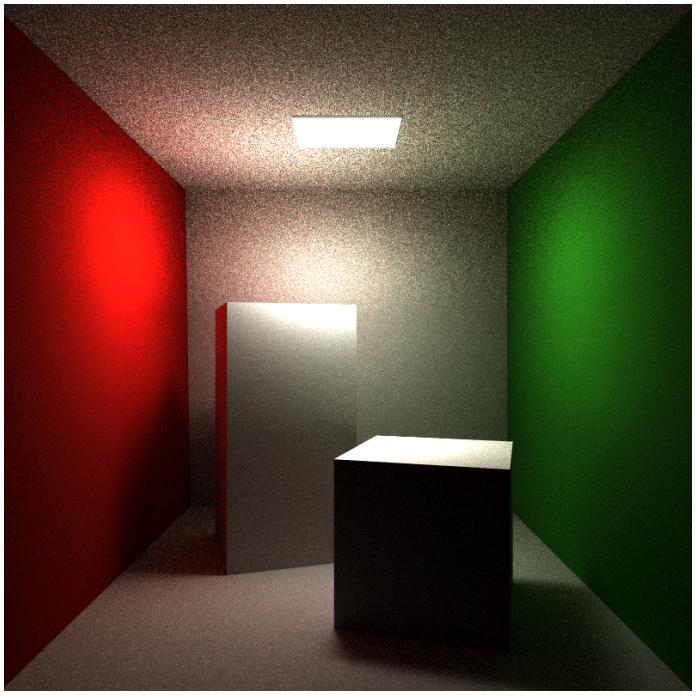
Vliv vzorkování světelných zdrojů

Tato část srovnává rozdíl v obraze, pokud je světlo vzorkováno pomocí více bodů a pokud je vzorkováno pouze jedním bodem.

- 3 odrazy primárního paprsku (hloubka rekurze),
- rozlišení obrazu 800x800 pixelů,
- vzorkování pouze jednoho světelného zdroje,
- 128 vzorků na pixel.



Obrázek 8.8: 1 vzorek na světlo



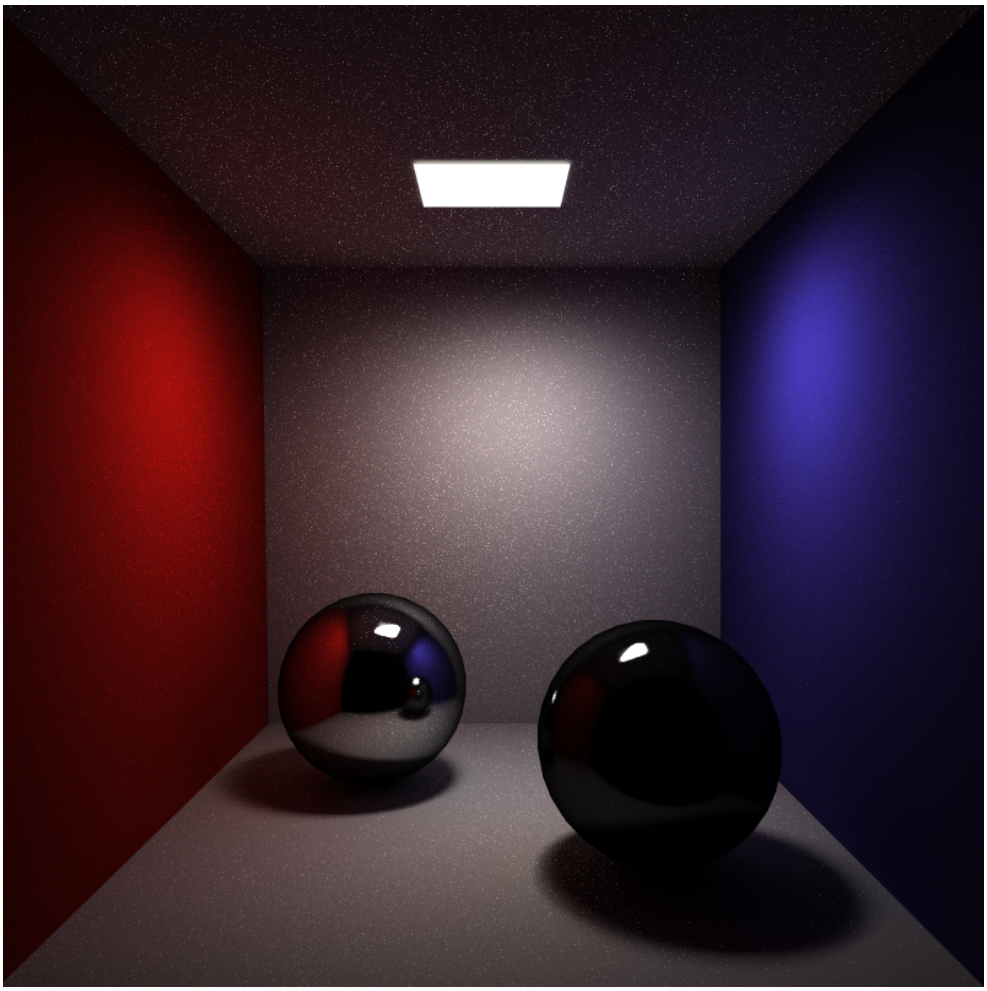
Obrázek 8.9: 8 vzorků na světlo

Zde je rozdíl téměř minimální a subjektivně považuji oba obrázky za stejně vypadající. Kde ale rozdíl pozorovat lze, je rychlost výpočtu.

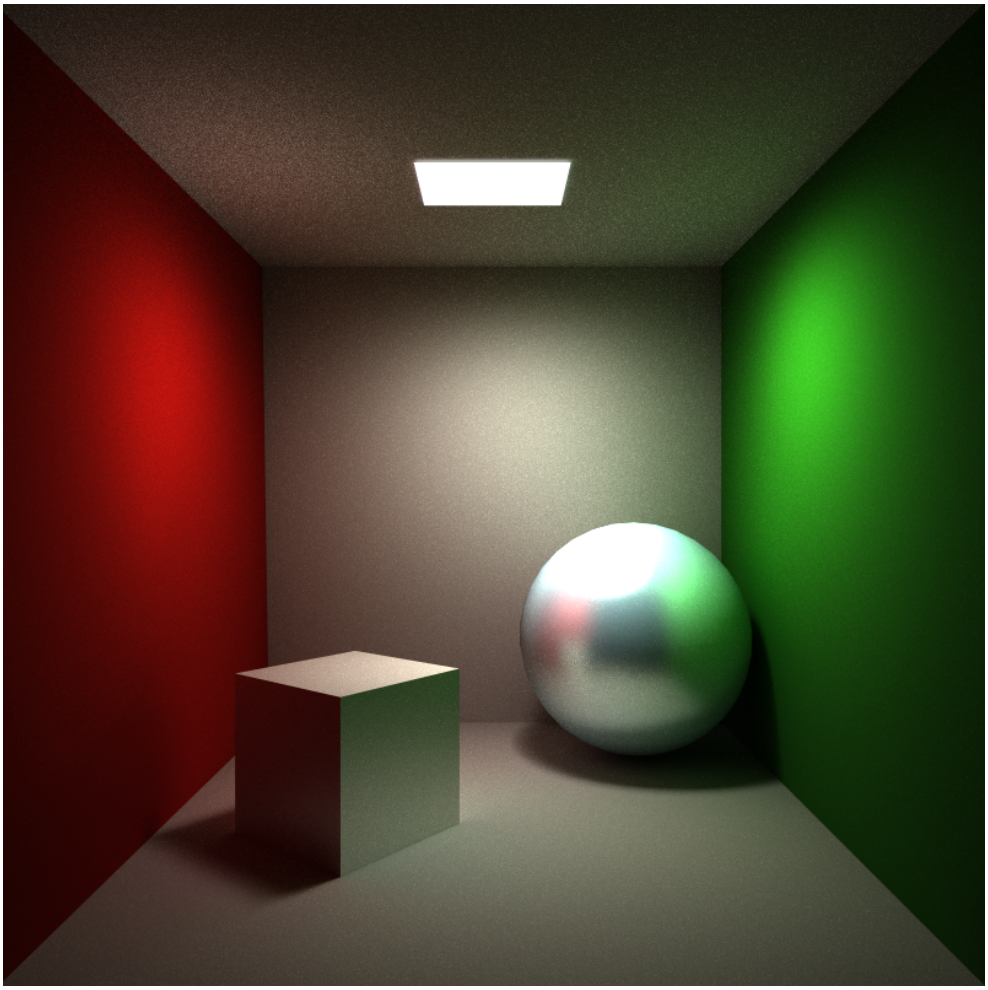
Výpočet Obrázku 8.9 zabral necelých 41 minut, oproti 9 minutám u obrázku 8.8. Je možné usoudit, že pro tuto konkrétní scénu nemá význam použít více vzorků na světlo. Tento fakt může být hodně ovlivněn jednoduchostí scény a použitím pouze jedno světelného zdroje (technicky vzato je tvořen dvěma trojúhelníky, ale jejich příspěvky jsou téměř totožné).

Ukázka materiálů a výsledků

Tato sekce slouží čistě k ukázce různých implementovaných materiálů a výstupů renderování.



Obrázek 8.10: Zrcadlový povrch, 77 minut



Obrázek 8.11: Simulace metalického povrchu, 87 minut

Závěr

Jedním z cílů práce bylo vytvořit tento text jako možný výukový materiál. Z toho důvodu je teoretická část poněkud obsáhlejší, jelikož jsem se snažil co nejlépe popsat veškerou problematiku a dovysvětlit určité nejasnosti, které mohou nastat. Dále jsem práci rozšířil o vlastní ilustrace, jelikož ty uvedené u některých zdrojů jsou často chaotické, nebo nejsou příliš názorné. Tím byl zároveň splněn cíl, který měl přiblížit problematiku globálního osvětlování a představit metody určené pro její řešení.

Dalším cílem bylo implementovat vlastní renderovací systém. Ten je vytvořen pomocí metody path tracing za využití Monte Carlo integrace a MIS. Také byly zvoleny a implementovány dva optimalizační prvky: akcelerační struktura BVH strom a zrychlení výpočtu pomocí paralelizmu. Měření této optimalizace, společně s ukázkou výsledků je možné vidět v poslední kapitole.

Implementované řešení je možné jednoduše rozšířit o další druhy materiálů, světelné zdroje, či typy objektů. Do budoucna bych rád přidal podporu textur a simulaci vlivu prostředí na přenos světla, čemuž se tato práce nevěnovala. Možné další vylepšení spočívá v lepší optimalizaci a vytvoření GUI k ovládání systému.

Literatura

[1] Dombek, D.; Klouda, K.: Lineární algebra: Studijní text [online]. 2020, [cit. 2021-02-22]. Dostupné z: https://courses.fit.cvut.cz/BI-LIN/@master/lectures/

[2] Pharr, M.; Jakob, W.; Humphreys, G.: *Physically based rendering: from theory to implementation*. Cambridge, MA: Morgan Kaufmann Publishers/Elsevier, třetí vydání, 2017, ISBN 978-0-12-800645-0.

[3] Barto, L.: Afinní a euklidovský prostor [online]. 2005, [cit. 2021-03-03]. Dostupné z: https://www2.karlin.mff.cuni.cz/~barto/student/Afineuklid.pdf

[4] Wikipedie: Otevřená encyklopedie: Normála [online]. 2019, [cit. 2021-02-22]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Norm%C3%A1la&oldid=17526035

[5] Nykamp, D. Q.: Magnitude of a vector definition [online]. 2021, [cit. 2021-02-22]. Dostupné z: http://mathinsight.org/definition/magnitude_vector

[6] Brilliant.org: Unit Vectors [online]. 2021, [cit. 2021-02-22]. Dostupné z: https://brilliant.org/wiki/unit-vectors/

[7] Lengyel, E.: *Mathematics for 3D game programming and computer graphics*. Boston, MA: Course Technology, Cengage Learning, třetí vydání, 2012, ISBN 978-1-4354-5886-4.

[8] Wikipedia: The Free Encyclopedia: Cross product [online]. 2021, [cit. 2021-02-21]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Cross_product&oldid=1006434274

[9] Wikipedie: Otevřená encyklopedie: Regulární matice [online]. 2016, [cit. 2021-02-23]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Regul%C3%A1rn%C3%AD_matice&oldid=14164480

[10] Curless, B.: Ray-triangle intersection [online]. 2006, [cit. 2021-03-03]. Dostupné z: https://courses.cs.washington.edu/courses/csep557/09sp/lectures/triangle_intersection.pdf

[11] Žára, J.; Beneš, B.; Felkel, P.: *Moderní počítačová grafika*. Brno: Computer Press, druhé vydání, 2004, ISBN 80-251-0454-0.

[12] Křivánek, J.: Počítačová grafika III – Radiometrie [online]. 2014, [cit. 2021-03-10]. Dostupné z: https://cgg.mff.cuni.cz/~jaroslav/teaching/2014-npgr010/slides/02%20-%20npgr010-2014%20-%20radiometrie.pdf

[13] Loem, M.: Monte Carlo Integration and Sampling Methods [online]. 2020, [cit. 2021-04-15]. Dostupné z: https://towardsdatascience.com/monte-carlo-integration-and-sampling-methods-25d5af53e1

[14] Wikipedie: Monte Carlo integrování — Wikipedie: Otevřená encyklopedie [online]. 2021, [cit. 2021-04-15]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Monte_Carlo_integrov%C3%A1n%C3%AD&oldid=19370181

[15] Wikipedie: Metoda Monte Carlo — Wikipedie: Otevřená encyklopedie [online]. 2021, [cit. 2021-04-15]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Metoda_Monte_Carlo&oldid=19780304

[16] Pham, G.; Lee, S.-H.; Kwon, K.-R.: Interpolating Spline Curve-Based Perceptual Encryption for 3D Printing Models. *Applied Sciences*, ročník 8, č. 2, Únor 2018: str. 242, ISSN 2076-3417, doi:10.3390/app8020242. Dostupné z: http://www.mdpi.com/2076-3417/8/2/242

[17] Meagher, D.: Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, ročník 19, č. 2, 1982: s. 129–147, ISSN 0146-664X, doi:https://doi.org/10.1016/0146-664X(82)90104-6. Dostupné z: https://www.sciencedirect.com/science/article/pii/0146664X82901046

[18] Hughes, J. F.: *Computer graphics: principles and practice*. Upper Saddle River, New Jersey: Addison-Wesley, third edition vydání, 2014, ISBN 9780321399526.

[19] Roth, S. D.: Ray casting for modeling solids. *Computer Graphics and Image Processing*, ročník 18, č. 2, Únor 1982: s. 109–144, ISSN 0146-664X, doi:10.1016/0146-664X(82)90169-1. Dostupné z: https://www.sciencedirect.com/science/article/pii/0146664X82901691

[20] Möller, T.: *Real-time rendering*. Boca Raton: Taylor & Francis, CRC Press, fourth edition vydání, 2018, ISBN 978-1-138-62700-0.

[21] Wikipedia contributors: Wave–particle duality — Wikipedia, The Free Encyclopedia [online]. https://en.wikipedia.org/w/index.php?title=Wave%E2%80%93particle_duality&oldid=1021968830, 2021, [cit. 2021-04-03].

[22] Ashdown, I.: Photometry and Radiometry: A Tour Guide for Computer Graphics Enthusiasts [online]. 1996, [cit. 2021-03-10]. Dostupné z: https://www.researchgate.net/publication/238294094_Photometry_and_Radiometry_A_Tour_Guide_for_Computer_Graphics_Enthusiasts

[23] de Greve, B.: Reflections and Refractions in Ray Tracing [online]. 2006, [cit. 2021-03-26]. Dostupné z: https://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf

[24] Glassner, A. S.: *Principles of digital image synthesis*. The Morgan Kaufmann series in computer graphics and geometric modeling, San Francisco: Morgan Kaufmann, 1995, ISBN 978-1-55860-276-2.

[25] Nicodemus, F. E.: Directional Reflectance and Emissivity of an Opaque Surface. *Appl. Opt.*, ročník 4, č. 7, Jul 1965: s. 767–775, doi:10.1364/AO.4.000767. Dostupné z: http://ao.osa.org/abstract.cfm?URI=ao-4-7-767

[26] Cook, R. L.; Torrance, K. E.: A Reflectance Model for Computer Graphics. *ACM Trans. Graph.*, ročník 1, č. 1, Leden 1982: str. 7–24, ISSN 0730-0301, doi:10.1145/357290.357293. Dostupné z: https://doi.org/10.1145/357290.357293

[27] Phong, B. T.: Illumination for Computer Generated Pictures. *Commun. ACM*, ročník 18, č. 6, Červen 1975: str. 311–317, ISSN 0001-0782, doi:10.1145/360825.360839. Dostupné z: https://doi.org/10.1145/360825.360839

[28] Smith, B.: Illustration of the components of the Phong reflection model (Ambient, Diffuse and Specular reflection). 2006, [cit. 2021-03-26]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/6/6b/Phong_components_version_4.png

[29] Blinn, J. F.: Models of Light Reflection for Computer Synthesized Pictures. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '77, New York, NY, USA: Association for Computing Machinery, 1977, ISBN 9781450373555, str. 192–198, doi:10.1145/563858.563893. Dostupné z: https://doi.org/10.1145/563858.563893

[30] Koppal, S. J.: *Lambertian Reflectance*. Boston, MA: Springer US, 2014, ISBN 978-0-387-31439-6, s. 441–443, doi:10.1007/978-0-387-31439-6_534. Dostupné z: https://doi.org/10.1007/978-0-387-31439-6_534

[31] Kajiya, J. T.: The Rendering Equation. *SIGGRAPH Comput. Graph.*, ročník 20, č. 4, Srpen 1986: str. 143–150, ISSN 0097-8930, doi:10.1145/15886.15902. Dostupné z: https://doi.org/10.1145/15886.15902

[32] Křivánek, J.: Computer graphics III – Rendering equation and its solution [online]. 2015, [cit. 2021-04-20]. Dostupné z: https://cgg.mff.cuni.cz/~jaroslav/teaching/2015-npgr010/slides/07%20-%20npgr010-2015%20-%20rendering%20equation.pdf

[33] Whitted, T.: An Improved Illumination Model for Shaded Display. *Commun. ACM*, ročník 23, č. 6, Červen 1980: str. 343–349, ISSN 0001-0782, doi:10.1145/358876.358882. Dostupné z: https://doi.org/10.1145/358876.358882

[34] Lafortune, E. P.; Willems, Y. D.: Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, Alvor, Portugal, Prosinec 1993, s. 145–153.

[35] Jensen, H. W.: *Realistic Image Synthesis Using Photon Mapping*. USA: A. K. Peters, Ltd., 2001, ISBN 1568811470.

[36] Goral, C. M.; Torrance, K. E.; Greenberg, D. P.; aj.: Modeling the Interaction of Light between Diffuse Surfaces. *SIGGRAPH Comput. Graph.*, ročník 18, č. 3, Leden 1984: str. 213–222, ISSN 0097-8930, doi:10.1145/964965.808601. Dostupné z: https://doi.org/10.1145/964965.808601

[37] Wikipedia contributors: Radiosity (computer graphics) — Wikipedia, The Free Encyclopedia [online]. 2020, [cit. 2021-04-03]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Radiosity_(computer_graphics)&oldid=996453063

[38] Cohen, M. F.; Chen, S. E.; Wallace, J. R.; aj.: A Progressive Refinement Approach to Fast Radiosity Image Generation. *SIGGRAPH Comput. Graph.*, ročník 22, č. 4, Červen 1988: str. 75–84, ISSN 0097-8930, doi:10.1145/378456.378487. Dostupné z: https://doi.org/10.1145/378456.378487

[39] Möller, T.; Trumbore, B.: Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphics Tools*, ročník 2, č. 1, Leden 1997: s. 21–28, ISSN 1086-7651, doi:10.1080/10867651.1997.10487468, publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10867651.1997.10487468. Dostupné z: https://doi.org/10.1080/10867651.1997.10487468

[40] Ray Tracing: Rendering a Triangle (Möller-Trumbore algorithm) [online]. [cit. 2021-05-03]. Dostupné z: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection

[41] Kay, T. L.; Kajiya, J. T.: Ray Tracing Complex Scenes. *SIGGRAPH Comput. Graph.*, ročník 20, č. 4, Srpen 1986: str. 269–278, ISSN 0097-8930, doi:10.1145/15886.15916. Dostupné z: https://doi.org/10.1145/15886.15916

[42] Tavian Barnes: Fast, Branchless Ray/Bounding Box Intersections [online]. 2011, [cit. 2021-05-06]. Dostupné z: https://tavianator.com/2011/ray_box.html

[43] Shirley, P.; Laine, S.; Hart, D.; aj.: *Sampling Transformations Zoo*. Berkeley, CA: Apress, 2019, ISBN 978-1-4842-4427-2, s. 223–246, doi: 10.1007/978-1-4842-4427-2_16. Dostupné z: https://doi.org/10.1007/978-1-4842-4427-2_16

[44] tinyobjloader: tinyobjloader. https://github.com/tinyobjloader/tinyobjloader, 2020.

Příloha A

Seznam použitých zkratek

BRDF Bidirectional Reflectance Distribution Function

MIS Multiple importance sampling

SPP Samples per pixel

GUI Graphical user interface

BVH Bounding Volume hierarchies

BSP Space subdivision hieararchies

AABB Axis-Aligned Bounding Box

Příloha B

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `renderer` ..... adresář s projektem praktické práce ve Visual Studiu 2019
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF