VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

REALISTICKÁ SIMULACE KOMPLEXNÍCH MATERIÁLŮ

REALISTIC SIMULATION OF COMPLEX MATERIALS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATĚJ TOUL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL VLNAS**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

152768

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Toul Matěj**
Program: Informační technologie
Název: **Realistická simulace komplexních materiálů**
Kategorie: Počítačová grafika
Akademický rok: 2023/24

Zadání:

1. Nastudujte techniky realistického zobrazování a zaměřte se na způsoby řešení komplexních materiálů (vosk, lidská kůže, mramor, kapaliny, ...) pomocí simulace BSSRDF.
2. Diskutujte možnosti využití GPU s použitím Vulkan API.
3. Vyberte vhodnou metodu zobrazování a vhodný algoritmus pro simulaci materiálů.
4. Navrhněte implementaci pro zobrazování a simulaci materiálů.
5. Implementujte zvolenou metodu a vytvořte demonstrační aplikaci s několika různými materiály.
6. Vyhodnoťte dosažené výsledky a též možnosti dalšího vývoje.

Literatura:

- Wojciech Jarosz. *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. Ph.D. dissertation, UC San Diego, September 2008.
- Modest MF. *Radiative Heat Transfer*. 3rd ed. New York: Academic Press; 2013. http://site.ebrary.com/id/10661889.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Vlnas Michal, Ing.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 9.11.2023

Abstrakt

Cílem této práce je simulace a vykreslování komplexních materiálů ze skutečného světa, jakými jsou např. kapaliny, plyny, vosk atd. Tyto komplexní materiály jsou často označovány jako zúčastněná média. U této simulace je kladen důraz na fyzikální věrohodnost a zároveň výpočetní efektivitu. Implementace využívá metody sledování cest pro co nejrealističtější výsledky. V rámci této metody jsou světelné interakce řešeny pomocí BSSRDF a všechny výpočty jsou vysoce paralelizovány na GPU s pomocí rozhraní Vulkan. Za pomoci těchto postupů je možné věrně simulovat široké spektrum materiálů na základě jejich optických vlastností. Výsledný program je možné využít k věrohodné simulaci komplexního světelného přenosu, ať už pro vzdělávací či vědecké účely, nebo pro vizuální požitek.

Abstract

The aim of this paper is to simulate and render complex real world materials (liquids, gases, wax and other materials), often referred to as participating media. The priorities of this simulation are both physical plausibility and computational efficiency. The simulation is implemented using a path tracer for results as close to photorealism as possible. Inside the path tracer, light interactions are handled using BSSRDF and all the calculations are highly parallelized using GPU and Vulkan API. Using these techniques, a wide range of materials can be accurately simulated based purely on their measured light scattering properties. The output of this work can be used to plausibly simulate complex light transport across different scenes and materials, either for scientific reasons or for visual entertainment.

Klíčová slova

Vulkan API, fotorealistické zobrazování, Monte Carlo sledování cesty, zúčastněná média, Henyey-Greensteinova fázová funkce, BSSRDF, GLSL, C++

Keywords

Vulkan API, photorealistic rendering, Monte Carlo path tracing, participating media, Henyey-Greenstein phase function, BSSRDF, GLSL, C++

Citace

TOUL, Matěj. *Realistická simulace komplexních materiálů.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Vlnas

Realistická simulace komplexních materiálů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Vlnase. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Matěj Toul
2. května 2024

Poděkování

Chtěl bych poděkovat Ing. Michalu Vlnasovi za skvělé vedení práce, jeho vstřícný přístup a ochotu vysvětlit jakoukoliv nejasnost, která se při vypracování naskytla. Za korekturu práce děkuji PaedDr. Jitce Gruntové a dále děkuji všem svým nejbližším za trpělivost, kterou se mnou při psaní měli. V neposlední řadě bych chtěl poděkovat také sobě. Za to, že jsem si věřil. Za to, že jsem tuto práci zvládl. Chtěl bych si poděkovat za to, že jsem si nedal pohov a za to, že jsem vše dotáhl do konce.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá studiem různých technik a algoritmů využívaných pro realistickou a fyzikálně věrnou simulaci materiálů, jež vykazují komplexní optické vlastnosti. Dále ukazuje, jak je možné tyto materiály klasifikovat, a které optické jevy je v nich možno pozorovat. Popisuje vývoj programu demonstrujícího jeden z možných přístupů pro realistickou simulaci a jeho implementaci určenou pro běh na grafickém procesoru.

Realistické simulace jsou předmětem výzkumu již desítky let. Existuje tak nespočet metod pevně zakořeněných v exaktním fyzikálním popisu skutečného světa, které jsou schopny věrně zachytit světelný přenos v rozmanitých trojrozměrných scénách. Tyto metody však většinou zůstaly pouhými teoriemi, nebo jejich výpočty probíhaly velmi dlouho a jen na těch nejvýkonnějších strojích.

V současnosti jsou však realistické simulace čím dál více dostupné také široké veřejnosti. Především díky obrovským technologickým pokrokům na poli vývoje grafických procesorů tak již nejde jen o edukační, úzce zaměřený nástroj, ale své využití tyto simulace často nalézají například ve filmovém či videoherním průmyslu. Dnes tyto simulace mohou spouštět i běžní uživatelé a fotorealistické snímky generovat v rámci minut, sekund, nebo dokonce v reálném čase. Některé opticky komplexnější materiály jsou však pro realistické zobrazování problematické i v dnešní době.

Cílem této práce je porozumět metodám pro fotorealistické zobrazování a možnostem jejich aplikace pro komplexní materiály, jako např. kapaliny, lidská kůže, vosk apod. Dalším cílem je pak navrhnout a vytvořit implementaci některé z těchto metod, uzpůsobenou pro použití na grafickém procesoru za pomoci rozhraní Vulkan.

Následující kapitola popisuje základní fyzikální vlastnosti světla a jejich reprezentaci v prostředí počítačové grafiky. Dále prozkoumává obecné přístupy pro řešení světelného přenosu v trojrozměrných scénách a popisuje konkrétní metody realistického zobrazování. Důraz je zde kladen na metodu sledování cest (path tracing), její princip a algoritmy, které zahrnuje. Ve třetí kapitole jsou do hloubky popsány zmíněné komplexní materiály, často označované jako zúčastněná média, a jejich fyzikální vlastnosti. Dále zde jsou nastíněny různé postupy klíčové pro simulaci médií v kontextu počítačové grafiky. Čtvrtá část představuje zvolený postup pro realistickou simulaci a návrh programu, jehož implementace a výstupy jsou předmětem poslední kapitoly této práce.

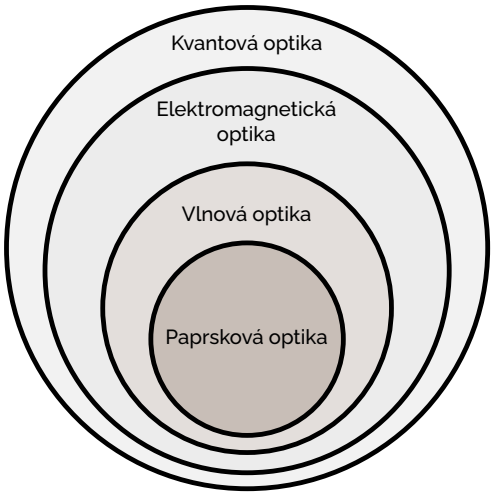
Kapitola 2

Techniky realistického zobrazování

Úkolem realistického zobrazování je zachytit snímek virtuální scény do dvojrozměrného obrazu, a přitom co nejvěrněji simulovat skutečné jevy vyskytující se v přírodě. Techniky realistického zobrazování v počítačové grafice se proto často přírodou a jejím fyzikálním popisem inspirují a tyto jevy blízce napodobují. Simulují světlo vyzářené světelnými zdroji, jeho rozmanité interakce napříč scénou a výsledek těchto interakcí posléze zachycují virtuálním senzorem – ten je běžně nazýván „kamera“, stejně jako jeho fyzický protějšek. Tato kapitola objasňuje nutné teoretické pozadí a dále konkrétní techniky realistického zobrazování a vykreslování, které se v počítačové grafice využívají.

2.1 Světelné jevy a interakce

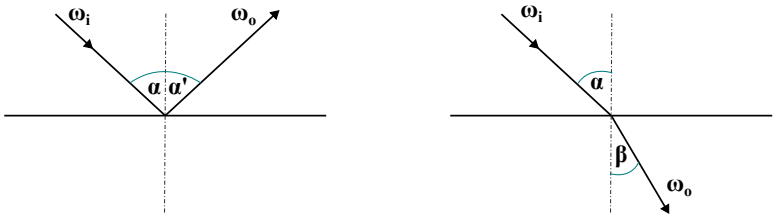
Světelnými jevy a interakcemi se ve fyzice zabývá *optika*. Z pohledu klasické fyziky světlo precizně popisuje *elektromagnetická teorie* [28]. Ta vysvětluje většinu jevů, které se v přírodě vyskytují a které jsou pro realistické zobrazení klíčové. Pro praktické využití v počítačové grafice i jiných aplikovaných oborech však bývá její popis často příliš složitý. V praxi se proto využívají specializované zjednodušené modely řešící konkrétní jevy. Na obrázku 2.1 jsou modely hierarchicky znázorněny.



Obrázek 2.1: Poddisciplíny optiky.

Zjednodušené modely

S tím nejjednodušším modelem pracuje *paprsková optika*. Ta popisuje světlo s pomocí pojmů obyčejné geometrie. Pracuje s předpoklady, že ve vakuu světlo cestuje přímočaře ve formě paprsků, přičemž jednotlivé paprsky se navzájem neovlivňují [30]. Dále zcela zanedbává vlnovou povahu světla. I přes tato omezení však velmi jednoduše a korektně popisuje ty nejzákladnější světelné interakce – **odraz** a **lom** (viz obr. 2.2).



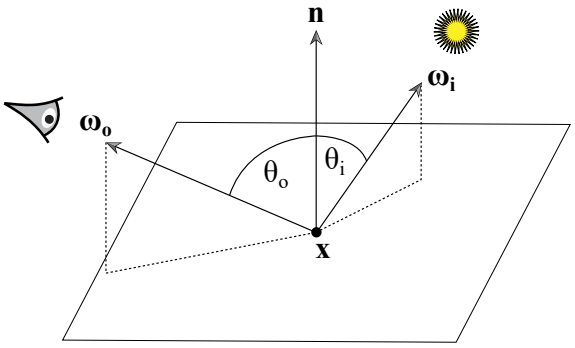
(a) Odraz (b) Lom

Obrázek 2.2: Světelné jevy paprskové optiky.

Pro realističtější zobrazování je však třeba brát v úvahu i další jevy. Na pomezí mezi paprskovou a elektromagnetickou optikou je optika *vlnová*, která vysvětluje jevy interference, difrakce, polarizace a další. Tyto jevy, obecněji jako **rozptyl**, popisuje *kvantová optika*, která světlo popisuje jako proud částic s určitou energií – fotonů. Objasňuje tak také vznik záření, **absorpci** a další jevy, u kterých nelze opomíjet kvantovou mechaniku, a v některých ohledech tak ještě obohacuje klasické elektromagnetické pojetí.

2.2 Distribuční funkce odrazu světla

Pro řešení odrazu, lomu nebo absorpce se v počítačové grafice využívá několik pravděpodobnostních funkcí. První z nich je obousměrná distribuční funkce odrazu, anglicky *bidirectional reflectance distribution function* (zkráceně **BRDF**). BRDF pracuje se zjednodušujícím předpokladem, že se světlo odrazí zpět právě z toho místa, kam dopadlo [13]. Řeší tedy interakce s nepropustnými povrchy. BRDF ve zkratce udává hustotu pravděpodobnosti, že se světlo dopadající ze směru $\omega_\mathbf{i}$ odrazí do směru $\omega_\mathbf{o}$ (viz obr. 2.3).



Obrázek 2.3: Diagram klíčových veličin pro výpočet BRDF[1].

[1] Převzato z https://pbr-book.org/3ed-2018/Color_and_Radiometry/Surface_Reflection.

Matematicky lze obecnou BRDF zapsat jako:

$$f_r(x, \omega_i \rightarrow \omega_o) = \frac{dL_r(x, \omega_o)}{dE(x)} = \frac{dL_r(x, \omega_o)}{L_i(x, \omega_i) \cos(\theta_i)\ d\omega_i}, \tag{2.1}$$

kde:

- $dL_r(x, \omega_o)$ značí diferenciální *zář* odraženou ve směru ω_o,
- $dE(x)$ je diferenciální *intenzita ozáření* v bodě x,
- $L_i(x, \omega_i)$ je *zář* dopadající ze směru ω_i,
- θ_i je úhel dopadu vztažený k normále plochy.

Vlastnosti BRDF

Aby byl fyzikálně založený model BRDF pro využití v počítačové grafice validní, musí splňovat tři klíčové vlastnosti:

1. **Helmholtzův zákon o reciprocitě** vychází z paprskové optiky, konkrétně zákona odrazu, a říká, že pokud dojde k záměně ω_o a ω_i (světelný paprsek se otočí), hodnota BRDF se nezmění. Matematicky je tato vlastnost zapsána jako:

$$f_r(x, \omega_i \rightarrow \omega_o) = f_r(x, \omega_o \rightarrow \omega_i). \tag{2.2}$$

2. **Zákon o zachování energie** udává, že výstupní *zářivý tok* nesmí převyšovat vstupní.

3. **Pozitivita** znamená, že všechny výstupní hodnoty BRDF musí být nezáporné, tedy:

$$f_r(x, \omega_i \rightarrow \omega_o) \geq 0. \tag{2.3}$$

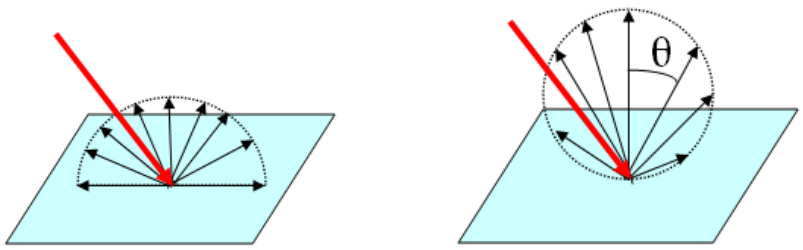
BRDF je čtyřrozměrnou funkcí (sférické souřadnice pro směr ke světlu a pro směr k pozorovateli), případně šestirozměrnou, může-li se funkce měnit v různých místech vykreslovaného objektu. [13] Jako další rozměr se často udává i *vlnová délka* λ pro explicitní vyjádření spektrální závislosti BRDF.

Modely BRDF

Běžně používané modely BRDF se liší vlastnostmi, výpočetní náročností a typem materiálu, který jsou schopny simulovat.

Ideální difúzní model (taktéž Lambertův model) patří mezi ty nejjednodušší BRDF. Velmi rychle modeluje perfektně matné (difúzní) materiály a byť není fyzikálně věrohodný, lze jím relativně přesně aproximovat skutečné povrchy jako například matnou barvu [24].

- Je specifický tím, že světlo odráží do všech směrů rovnoměrně (viz obr. 2.4). Objekt tak vypadá stejně bez ohledu na vektor směřující k pozorovateli.



(a) Úhel odrazu (b) Intenzita

Obrázek 2.4: Diagramy Lambertova modelu[2].

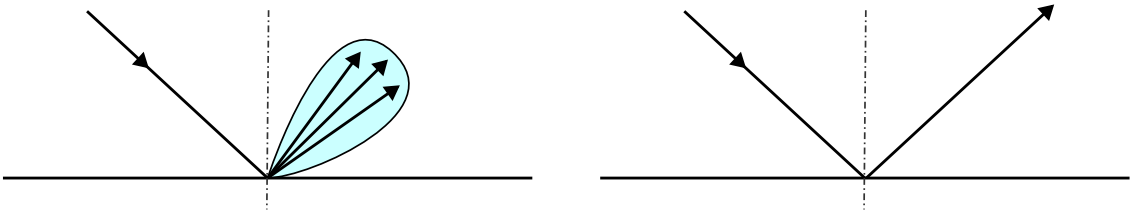
- Výstupní intenzita světla pak podléhá *Lambertovu zákonu* a závisí na úhlu dopadu a intenzitě dopadajícího světla. Lambertův zákon je dán vztahem:

$$I = I_0 \cos\theta, \tag{2.4}$$

kde I je výstupní intenzita, I_0 je vstupní intenzita a θ je úhel vektoru k pozorovateli.

Ideální spekulární (zrcadlový) model slouží k simulování lesklých povrchů, jež téměř perfektně dodržují zákon odrazu. Zpravidla se rozlišují povrchy lesklé (angl. glossy), jako např. voda nebo kovy, a povrchy přímo zrcadlové (angl. specular).

- Čím více zrcadlový materiál je, tím více se obor hodnot BRDF přibližuje ideálnímu odrazu (viz obr. 2.5). Vektor odrazu zrcadlového modelu je tak silně závislý na vektoru k pozorovateli.
- Intenzita po odrazu často zůstává neměnná, případně se na výstupním paprsku může projevit zrcadlový odlesk (angl. specular color).



(a) Lesklý povrch (b) Zrcadlový povrch

Obrázek 2.5: BRDF pro odrazivé povrchy.

Phongův model je jedním z mnoha modelů, kde se využívá jak difúzních, tak spekulárních odrazů. V přírodě se totiž ideálně matné, ani ideálně zrcadlové materiály nevyskytují, a tak se mezi nimi velmi často interpoluje. Model byl publikován výzkumníkem Phongem již v roce 1975 [25] a na dnešní poměry realistických simulací je již zastaralý, v počítačové grafice má však stále své místo, zejména v modifikaci známé jako Blinn-Phongův model [2]. Jeho složky jsou celkem tři (viz obr. 2.6):

[2]Převzato z https://www.oceanopticsbook.info/view/surfaces/lambertian-brdfs.

- **Ambientní složka** (I_A) je konstantní, daná pouze intenzitou ambientního světla ve scéně I_a a odrazivostí materiálu k_a.

$$I_A = I_a k_a \tag{2.5}$$

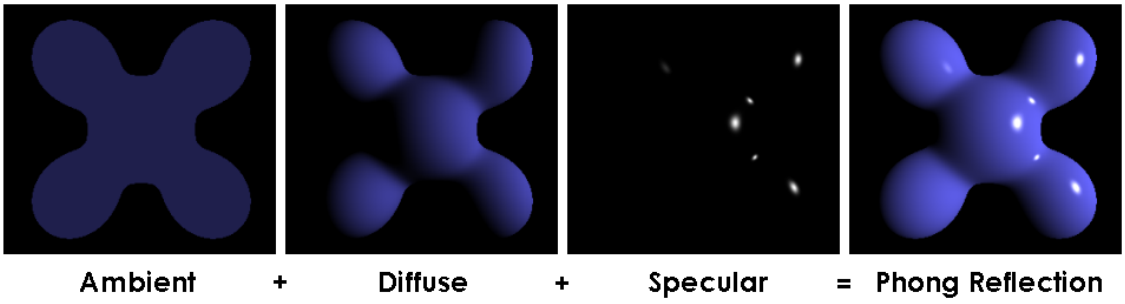
- **Difúzní složka** (I_D) je odvozena ze vztahu 2.4 pro ideální matný povrch.

$$I_D = I_L C(\vec{L} \cdot \vec{N}) \tag{2.6}$$

  Hodnotu $\cos\theta$ zde s použitím definice skalárního součinu nahrazuje skalární součin vektoru ke světlu $\vec{L}$ a normály $\vec{N}$. Oba vektory musí být normalizované. Konstanta C pak symbolizuje barvu povrchu.

- **Spekulární složka** (I_S) závisí na intenzitě odrazů I_s, odrazivosti k_s určující procento spekulárně odraženého světla vůči difúznímu a vektorech $\vec{V}$ a $\vec{R}$ směřujících k pozorovateli a ve směru odrazu světla. Koeficient n udává, jak intenzivní odlesky budou.

$$I_S = I_s k_s (\vec{V} \cdot \vec{R})^n \tag{2.7}$$

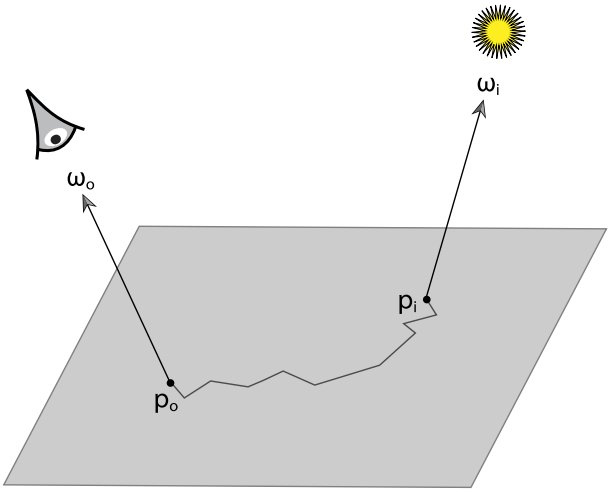


Obrázek 2.6: Složky Phongova modelu[3].

Prakticky opačnou funkcí k BRDF je potom obousměrná distribuční funkce propustnosti, anglicky *bidirectional transmittance distribution function* (zkráceně **BTDF**). Ta bere v úvahu skutečnost, že se světlo může do materiálu zalomit a řeší směr, kterým paprsek dále poputuje po tom, co z materiálu vystoupí. Podstata BTDF je prakticky stejná jako BRDF, a tak se obě funkce v praxi definují jako obousměrná distribuční funkce rozptylu, anglicky *bidirectional scattering distribution function* (zkráceně **BSDF**) [1].

Řada komplexních materiálů (např. vosk, lidská kůže, kapaliny, mlha, ...) však světlo ovlivňuje i mezi těmito hraničními interakcemi, uvnitř materiálu. Tento jev se běžně označuje jako **podpovrchový rozptyl**, anglicky *subsurface scattering* [13]. Po jeho působení se světlo mimo jiné již nemusí odrážet z místa dopadu. Spojení BRDF, BTDF (tedy BSDF) a podpovrchového rozptylu se souhrnně označuje jako obousměrná distribuční funkce odrazu rozptylového povrchu, anglicky *bidirectional scattering-surface reflectance distribution function* (zkr. **BSSRDF** viz obr. 2.7).

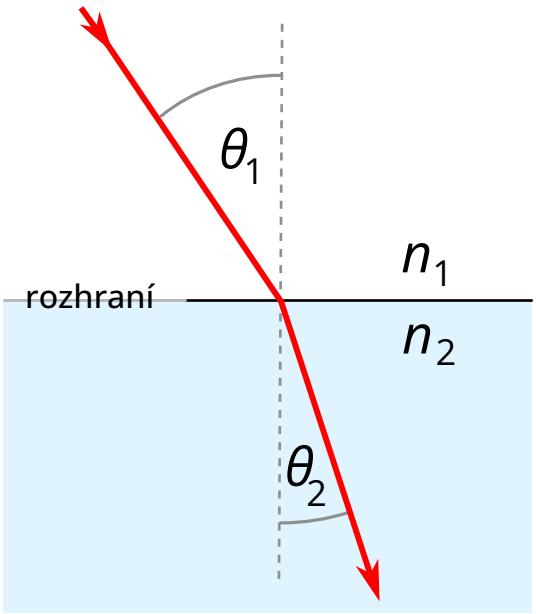
[3]Převzato z https://commons.wikimedia.org/wiki/File:Phong_components_version_4.png.



Obrázek 2.7: Po dopadu paprsku do bodu p_i dojde k jeho zanoření. Paprsek dále pokračuje pod povrchem materiálu, až dokud se nevynoří z bodu p_o[4].

Snellův zákon

Směr, kterým se paprsek vydá po zalomení do materiálu, udává **Snellův zákon**, pojmenovaný po nizozemském vědci Willebrordu Snellovi, který zákon uvedl ve své práci roku 1621. Výrazně detailnější popis nabídl roku 1637 René Descartes, a tak jsou tyto vztahy v některých zemích známy také jako *Snell-Descartův zákon*. Poslední označení jako *Ibn Sahlův zákon* připisuje objevení perskému učenci již v roce 984. Při nejmenším není pochyb, že Ibn Sahl rozuměl sinusovému zákonu lomu [18].



Obrázek 2.8: Snellův zákon.

[4]Převzato z https://pbr-book.org/3ed-2018/Color_and_Radiometry/Surface_Reflection.

Snellův zákon (obr. 2.8) je uceleně odvozen mj. v publikaci *Principles of Optics* od M. Borna a E. Wolfa (2013, s. 38 [3]) jako:

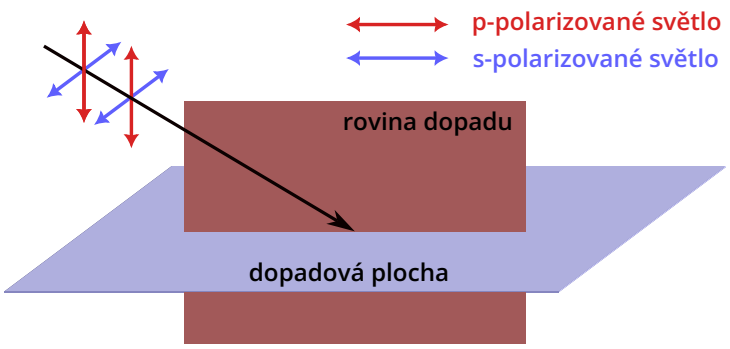
$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}, \tag{2.8}$$

kde θ_1 je úhel dopadu a θ_2 je úhel lomu. Konstanty n_1 a n_2 jsou potom *indexy lomu* obou materiálů a pro obvyklé materiály jde o tabelované hodnoty.

Fresnelovy vztahy

Pro účely věrné simulace průhledných a průsvitných materiálů, je nutné znát, jaká část světla se po interakci s materiálem odrazí (dle BSDF) a jaká se do materiálu zanoří dle rovnice 2.8. Tuto problematiku řeší **Fresnelovy vztahy** (není-li uvedeno jinak, rovnice v této podsekci jsou převzaty z knihy *Optics*, E. Hecht, 2002 [10]).

Fresnelovy vztahy závisí na *polarizaci* – vlastnosti světla, kterou definuje vlnová optika. Existuje několik druhů polarizace, Fresnelovy vztahy však uvažují dvě různé *lineární polarizace* (viz obr. 2.9), konkrétně v rovině dopadu (*polarizace typu p*) a v rovině na ni kolmé (*polarizace typu s*).



Obrázek 2.9: Lineární polarizace světla.

Koeficienty odrazu pro jednotlivé polarizace udávají následující Fresnelovy rovnice:

$$r_s = \frac{n_1\cos\theta_1 - n_2\cos\theta_2}{n_1\cos\theta_1 + n_2\cos\theta_2}, \tag{2.9}$$

$$r_p = \frac{n_2\cos\theta_1 - n_1\cos\theta_2}{n_1\cos\theta_2 + n_2\cos\theta_1}. \tag{2.10}$$

Tyto vztahy jsou přesným fyzikálním popisem obecně libovolného světla. Pro účely počítačových simulací se však velmi často využívá přírodního, nepolarizovaného světla. V tomto případě lze koeficienty zprůměrovat:

$$r = \frac{1}{2}(r_s + r_p). \tag{2.11}$$

Pro zjištění **koeficientu přenosu** t udávající tu část světla, která se do materiálu zanoří, je možné vyjít ze zákona zachování energie. Procento odraženého r a přeneseného světla t musí dát, bez započítání absorpce a dalších jevů, dohromady celek. Platí tedy $r + t = 1$.

Výpočet těchto koeficientů pro každý úhel býval v počítačových simulacích náročný. Christophe Schlick [27] proto v roce 1994 představil aproximaci s 0,6% chybou, dnes známou jako **Schlickova aproximace**:

$$r = r_0 + (1 - R_0)(1 - \cos\theta_1)^5, \tag{2.12}$$

kde:

$$r_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2. \tag{2.13}$$

Na moderních strojích je již rozdíl v rychlosti výpočtu přesného a aproximovaného koeficientu minimální a běžně se tak obě varianty zaměňují.

2.3 Světelné zdroje

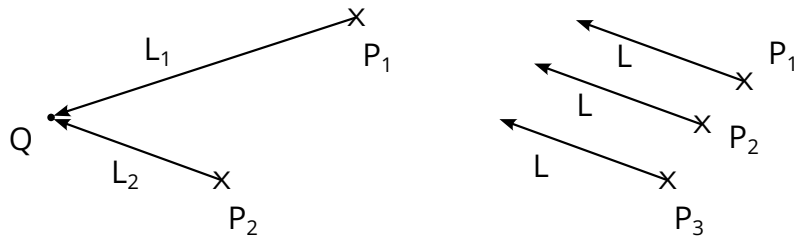
Pro výpočet výstupní intenzity světla za pomoci BSDF je třeba vědět, jakým směrem se světelný zdroj nachází a jaká je jeho intenzita v daném bodě. Proto se v počítačové grafice zavedly **modely světelných zdrojů**, do značné míry vycházející z fotometrie. V počítačové grafice se nejčastěji pracuje se třemi hlavními typy [5]:

Bodové světelné zdroje šíří světlo rovnoměrně do všech směrů – často jsou tak označovány jako *všesměrové zdroje*.

- Bodový světelný zdroj je plně definován svou **pozicí** a **intenzitou**.
- Vektor $\vec{L}$ ke světlu Q je pro účely výpočtů BSDF odlišný v každém daném bodě P trojrozměrné scény (viz obr. 2.10).

$$\vec{L} = Q - P \tag{2.14}$$

- Slouží k aproximaci reálných světelných zdrojů jako např. žárovka nebo plamen.



(a) Bodový světelný zdroj (b) Směrový světelný zdroj

Obrázek 2.10: Světelné zdroje.

Směrové světelné zdroje jsou aproximací prakticky bodových, ovšem velmi vzdálených, světelných zdrojů. Byť by se takové zdroje teoreticky daly simulovat právě takto – jako vzdálené bodové zdroje – vzdálenost světla od objektů by byla mnohonásobně vyšší než vzdálenost mezi objekty a mohlo by tak docházet k numerickým chybám.

- Směrový světelný zdroj je plně definován **směrem** a **intenzitou**.

- Aproximují bodové zdroje natolik vzdálené, že by rozdíly ve směru vektoru $\vec{L}$ byly na různých místech scény zanedbatelné (např. Slunce).
- Vektor $\vec{L}$ je v každém bodě trojrozměrné scény konstantní a je součástí definice směrového světelného zdroje.

Mapování prostředí (angl. *environment mapping*, nebo *environment lighting*) umožňuje osvětlení scény na základě projekce světa, tzv. **mapy prostředí**. Běžně jde o proces vzorkování textury, díky kterému je možné scénu nasvítit ze všech úhlů a navíc získat realistické odrazy prostředí bez nutnosti ho modelovat jako trojrozměrné objekty.

- Vzorkovaná textura nejčastěji obklopuje celou scénu jako krychle nazývaná anglickým pojmem **skybox**. Může jít ale i o kouli nebo paraboloid.
- Mapa prostředí může být zachycena přímo ze scény (používá se pro rychlé odrazy bez *sledování paprsku*), nebo může jít o skutečnou fotografii.
- Při použití mapy prostředí coby světelný zdroj je možné s obstojnou přesností simulovat zář příchozí ze všech okolních směrů.

Intenzita světelného zdroje

Fotometrie definuje osvětlení E jako světelný tok Φ dopadající na jednotku plochy A jako [31]:

$$E = \frac{d\Phi}{dA}. \tag{2.15}$$

Pro účely počítačové grafiky je možné využít také vztah pro bodový světelný zdroj:

$$E = \frac{I}{r^2} \cos \alpha. \tag{2.16}$$

Je zřejmé, že tento vztah úzce souvisí s výpočtem intenzity v ideálním difúzním modelu. Dosazením do vztahu 2.4 je možné získat rovnici pro vstupní intenzitu v daném bodě I_0 jako podíl svítivosti světelného zdroje I a mocniny vzdálenosti světelného zdroje r, tedy

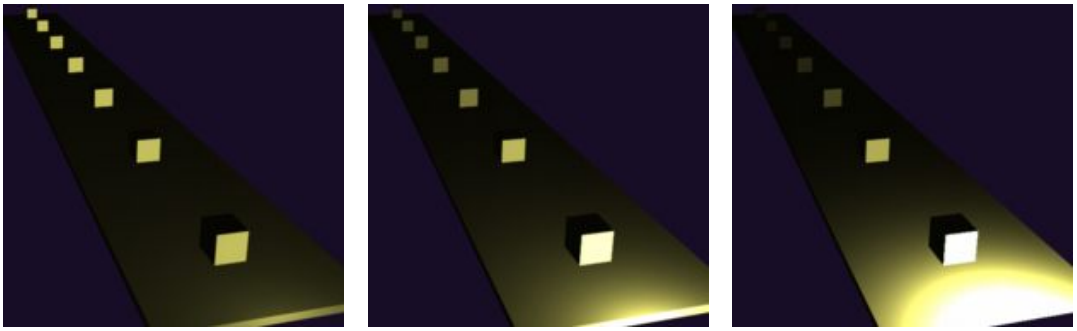
$$I_0 = \frac{I}{r^2}. \tag{2.17}$$

S rostoucí vzdáleností je tak působení světelného zdroje čím dál slabší. Tento jev se označuje jako světelný **útlum** (angl. attenuation).

Byť jde o fyzikálně věrný model, v počítačové grafice často nevypadá nejlépe. Při vzdálenostech $r < 1$ je totiž výsledná intenzita vlivem umocnění výrazně vyšší a ve snímcích může působit až rušivě (viz obr. 2.11). Vedle *kvadratického útlumu* se tak často využívá *lineární útlum*

$$I_0 = \frac{I}{r}, \tag{2.18}$$

případně se útlum úplně zanedbává [15].



(a) Bez útlumu (b) Lineární útlum (c) Kvadratický útlum

Obrázek 2.11: Typy útlumu intenzity světelného zdroje[5].

2.4 Zobrazovací rovnice

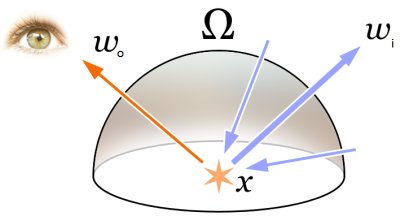
Výše bylo zmíněno, že realistické simulace často čerpají z geometrické optiky. Světlo je v nich tedy popsáno jako paprsek. Prvně definoval *zobrazovací rovnici* popisující *zář* tohoto paprsku James Kajiya v roce 1986 [16] jako integrál světla z množiny okolních ploch.

Prakticky zároveň s ním, dokonce na stejné konferenci, představil svou alternativu také David Immel [12], jehož varianta využívá již výše definovanou BRDF (viz kapitola 2.2) a nepočítá s konečnými paprsky (popisuje je pouze počátečním bodem a směrem). Rovnice se ale liší pouze zápisem, jinak jsou fakticky ekvivalentní. Zobrazovací rovnici Immel definoval jako integrál světla z množiny všech směrů (viz obr. 2.12) jako:

$$L(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i \to \omega_o) L_i(x, \omega_i) \cos(\theta_i) d\omega_i, \tag{2.19}$$

kde:

- $L(x, \omega_o)$ je celková zář vyzářena z bodu x ve směru ω_o k pozorovateli,
- $L_e(x, \omega_o)$ je zář vyzářena samotným objektem z bodu x do směru ω_o,
- $f_r(x, \omega_i \to \omega_o)$ je BRDF (příp. BSDF, nebo jiná funkce viz sekce 2.2),
- θ_i je úhel mezi směrovým vektorem ω_i a normálou povrchu,
- Ω je množinou všech směrů v hemisféře nad bodem x.



Obrázek 2.12: Znázornění veličin použitých v zobrazovací rovnici[6].

[5] Převzato z https://courses.washington.edu/arch481/1.TapestryReader/2.LightSources/5.Falloff/0.default.html.

[6] Převzato z https://commons.wikimedia.org/wiki/File:Rendering_eq.png.

2.5 Metody zobrazování a vykreslování

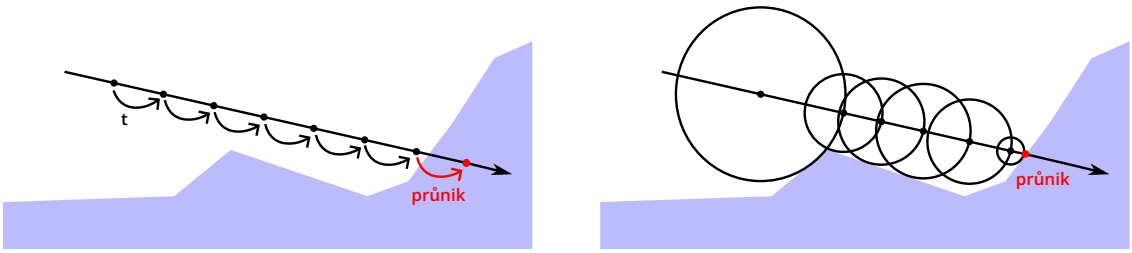
Pro řešení zobrazovací rovnice se v počítačové grafice využívá nespočet metod a jejich úprav. Obecně nejrozšířenější metodou je díky nízké výpočetní náročnosti *rasterizace*. Ta je však pro účely věrné fyzikální simulace zcela nevhodná, jelikož nijak nesleduje jednotlivé světelné interakce a řadu z nich pouze hrubě odhaduje. Využívají se proto různé metody, které světelné paprsky buď přímo simulují, nebo je alespoň věrně aproximují.

Zkreslení metody

Obecně se rozlišují dva typy zobrazovacích metod: *zkreslené* (angl. **biased**) a *nezkreslené* (angl. **unbiased**). Zkreslené metody často vynikají svou rychlostí, ovšem nekonvergují ke skutečnému řešení zobrazovací rovnice – pouze ji (s odchylkou, chybou) aproximují. Nezkreslené metody jsou výpočetně náročnější, ale fyzikálně věrné, a tedy vhodnější pro realistickou simulaci.

Metoda kráčení paprsku (ray marching)

Kráčení paprsku (angl. *ray marching*, viz obr. 2.13) je **zkreslená** metoda založená na naivním posunu paprsku o konstantní krok t [29]. To přináší několik nevýhod – je-li krok příliš velký, může se stát, že průnik s objektem nebude zaznamenán (paprsek jej překročí). Naopak, je-li krok příliš malý, algoritmus bude provádět nespočet „zbytečných“ kontrol podmínek, čímž je následně zastíněna hlavní výhoda metody, její rychlost. Algoritmus navíc v každém daném bodě pouze udává, zdali k průniku došlo, nebo ne. Nehledá konkrétní bod průniku a jde tak o metodu vesměs nepřesnou.



(a) Klasický ray marching

(b) Optimalizace sphere tracing

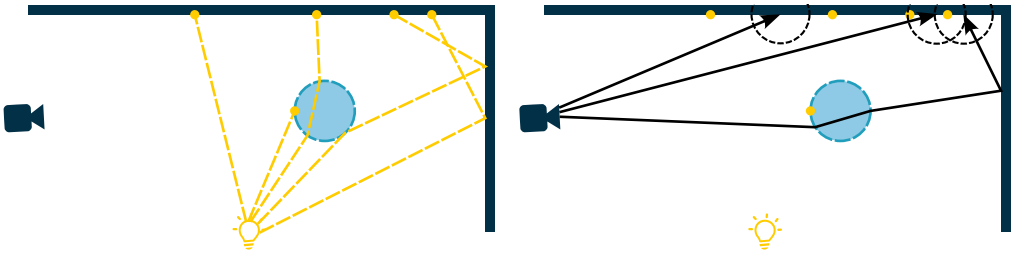
Obrázek 2.13: Vizualizace metody kráčení paprsku.

Problém s přesností kroku řeší optimalizace známá jako *sphere tracing* [9]. Objekty ve scéně se definují *vzdálenostní funkcí* (angl. signed distance function, zkr. **SDF**). SDF je vyhodnocena v každé iteraci algoritmu pro nalezení nejbližšího objektu. Jeho vzdálenost je pak nastavena jako délka kroku, díky čemuž může paprsek efektivně překročit velké prázdné prostory ve scéně a zároveň přesně zjistit konkrétní bod průniku s objektem.

Mapování fotonů (photon mapping)

Další **zkreslenou** metodou je mapování fotonů. Na rozdíl od všech ostatních zmíněných metod probíhá ve dvou krocích [14]. V prvním kroku jsou do scény ze světelných zdrojů vyslány částice světla – fotony (viz obr. 2.14). Při každém průniku s objektem je zaznamenán bod průniku a směrový vektor do tzv. *fotonové mapy*. Následně je na základě BRDF

(kapitola 2.2) rozhodnuto, zdali bude foton odražen, případně zanořen, a dále trasován, nebo dojde k absorpci a trasování skončí. K ukončení trasování může dojít také náhodně, na základě pravděpodobnostní metody známé jako *ruská ruleta* (viz sekce 2.8 níže).



(a) Tvorba fotonové mapy (b) Vykreslení fotonů

Obrázek 2.14: Vizualizace metody mapování fotonů.

V druhém kroku dochází k samotnému trasování světelných paprsků vypuštěných do scény směrem od pozorovatele (viz obr. 2.14). Čím více fotonů se dle fotonové mapy nachází v okolí světelné interakce, tím více váhy je pak k výsledné intenzitě přidáno. Jak už bylo zmíněno, mapování fotonů je zkreslená metoda, která nekonverguje k přesnému řešení. Navzdory tomu se ale ukazuje, že jde o efektivní metodu pro zobrazování některých skutečných optických jevů (např. *kaustika*), se kterými mohou mít některé nezkreslené metody (kvůli „jednosměrnému“ přístupu) problém.

Sledování paprsku (ray tracing)

Mezi nejstarší přístupy založené na paprskovém popisu světla patří metoda sledování paprsku (angl. ray tracing) [7]. Ta sama o sobě neslouží k fotorealistickému zobrazování a **neřeší zobrazovací rovnici**, položila však základy pokročilejším metodám, jako např. *path tracing*. Staví na tzv. *ray castingu* – metodě, při které jsou od pozorovatele vysílány paprsky, které jsou následně testovány na průnik se scénou.

Samotný *primární paprsek* pouze testuje, co je viditelné z pozice kamery, a je tak vizuálně podobný rasterizaci. Ray tracing však po tomto prvotním dopadu vysílá tzv. *sekundární paprsky*, které získávají o scéně více informací. Sekundární paprsky jsou tři [7]:

- **Stínový paprsek** je vyslán z bodu dopadu směrem ke všem světelným zdrojům ve scéně. Pokud paprsek zablokuje nějaký neprůhledný objekt, v místě dopadu se vykreslí stín. Pokud ne, intenzita zasáhnutého světelného zdroje v daném bodě je započítána do výpočtu osvětlení.
- **Odrazový paprsek** je vysílán v případě, že primární paprsek narazil na nějakou odrazivou plochu, např. zrcadlo. Slouží k tomu, aby bylo ve výsledném snímku možné vidět v odrazu další objekty ve scéně.
- **Průhlednostní paprsek** (také **refrakční**) je vysílán z bodu dopadu dovnitř objektu, má-li daný materiál schopnost lámat nebo propouštět světlo.

Na moderních grafických kartách je algoritmus ray tracingu vysoce optimalizovaný, a je tak možné klasickou rasterizaci s jeho pomocí obohacovat o různé efekty, a to s relativně nízkou časovou náročností. Např. v počítačových hrách se tak dnes ray tracing využívá při vykreslení stínů, odrazů, případně i dalších efektů jako *zastínění okolím* [6] (angl. ambient occlusion) nebo již zmíněné *kaustiky* [32].

Metoda sledování cesty (path tracing)

Sledování cesty, běžně označované svým anglickým názvem *path tracing*, představuje **nezkreslený** přístup k řešení zobrazovací rovnice. Vychází přímo z fyzikálního popisu světelného přenosu a světelné interakce a efekty tak simuluje přirozeně. Pro fotorealistické fyzikální simulace je tak tato metoda častou a obtížně nahraditelnou volbou.

2.6 Path tracing

Metodu path tracing představil přímo James Kajiya [16] spolu se svou zobrazovací rovnicí coby způsob jejího řešení. Při definici vycházel z *ray tracingu* – path tracing taktéž sleduje světelné paprsky, sleduje však cestu pouze jednoho z nich a s výjimkou stínového (tzv. *přímé osvětlení*) tak nevyužívá žádné sekundární paprsky. To, zdali se paprsek odrazí, nebo zanoří, je pak založeno na náhodě (deterministické generování pseudonáhodných čísel). Pro každý pixel výsledného snímku je takových cest simulováno několik a na konci algoritmu dochází k jejich zprůměrování. Jde tak o tzv. **Monte Carlo metodu**.

Algoritmus 1: Path tracing (iterativní)

```
Vstup: maxSamples, maxBounces
sumColor = Black;
i = 0;
while i ≠ maxSamples do
    color = White;
    j = 0;
    ray = generateRay(...);
    while j ≠ maxBounces do
        if intersect(ray) = false then
            break;
        end
        light = traceLight();                          /* přímé osvětlení */
        brdf = getBRDF(ray.hit.pos, ray.dir, light.dir);
        ray.origin = ray.hit.pos;
        ray.dir = randInHemisphere(ray.hit.pos);    /* plně difúzní odraz */
        cosθ = dot(ray.dir, ray.hit.normal);
        color = color · f_r · light.val · cosθ;          /* zobrazovací rovnice */
        j += 1;
    end
    sumColor += color;
    i += 1;
end
return sumColor/maxSamples;
```

Hlavní výhodou je, že tento přístup vede na velmi jednoduché, ale přirozené a fyzikálně věrné vykreslení řady optických jevů, které ostatní metody musí aproximovat, dodatečně trasovat apod. Path tracing věrně zobrazuje odraz světla, lom, měkké stíny, okolní zastínění nebo nepřímé osvětlení. Path tracing může být řešen rekurzivně, podobně jako původní ray tracing, v praxi se však mnohem častěji využívá **iterativní** přístup, který je mj. vhodnější pro realistické simulace na grafické kartě.

Naopak nevýhodou je výpočetní náročnost. Byť je path tracing ve své základní podstatě jednodušším a rychlejším algoritmem než ray tracing, každý vzorek je výsledkem pouze jediné cesty, která je navíc vysoce ovlivněna náhodou. Z toho důvodu mají jednotlivé vzorky výrazně větší rozptyl a ve výsledném snímku tak vzniká více šumu. Pro kompletní fotorealistickou simulaci je potom vyžadováno mnohem více vzorků, a tedy i času a výpočetních prostředků než u ray tracingu.

Optimalizace řešící tento problém se vyvíjejí rychle, a tak se např. ve videoherním průmyslu objevují první hry využívající path tracing v reálném čase. V současnosti jde však stále o velmi složité výpočty, které vyžadují špičkový hardware. Proto ve videoherním průmyslu a dalších odvětvích, kde je třeba podávat výsledky v reálném čase, stále dominuje rasterizace, případně obohacená o efekty (odrazy, stíny, ...) vykreslené pomocí ray tracingu. V tomto případě se však v žádném případě nejedná o věrnou fyzikální simulaci.

2.7 Akcelerační struktury

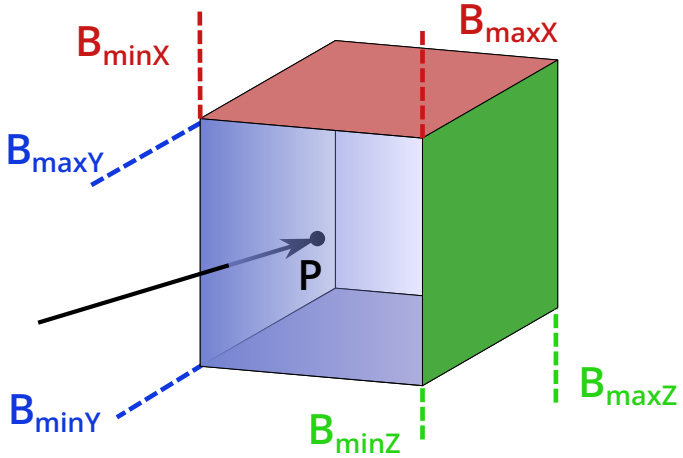
Rozsáhlé, komplexní scény se mohou skládat z velkého množství objektů, přičemž každý z nich se může dále skládat z tisíců trojúhelníků. Jelikož je výpočet průniku paprsku s trojúhelníkem relativně náročný, je v zájmu každé zobrazovací metody počet kandidátů na průnik co nejvíce snížit. Za tímto účelem vznikají tzv. *akcelerační struktury*.

Průnik paprsek–kvádr

Nejjednodušším typem průniku je **bod–kvádr**. Při této interakci se často využívá algoritmus **AABB** (*axis–aligned bounding boxes*, tedy „ohraničující kvádry zarovnané s osami“, viz obr. 2.15). Ten definuje průnik jako logický výraz [19]

$$\begin{aligned} f(P,B) = &(P_x \geq B_{minX} \wedge P_x \leq B_{maxX}) \\ &\wedge (P_y \geq B_{minY} \wedge P_y \leq B_{maxY}) \\ &\wedge (P_z \geq B_{minZ} \wedge P_z \leq B_{maxZ}), \end{aligned} \tag{2.20}$$

kde P je testovaný bod, B je testovaný kvádr a hodnoty *min* a *max* reprezentují minimální, respektive maximální hranice v jednotlivých osách.



Obrázek 2.15: Algoritmus AABB.

Pro metody jako path tracing je však výrazně důležitější průnik **paprsek–kvádr**. Když je totiž paprsek vyslán do scény, jeden konkrétní bod průniku nejprve není znám. Určen je pouze původ paprsku a jeho směr. Pro řešení těchto průniků se nejčastěji používá tzv. *metoda desek* (angl. *slabs method*), kterou představili Timothy Kay a James Kajiya [17], případně její úpravy. Paprsek je možné zapsat parametricky

$$p(t) = o \ + td, \tag{2.21}$$

kde $p(t)$ je konkrétní bod na trase paprsku ve vzdálenosti t, o je původ paprsku a d je jeho směr. Z rovnice lze vyjádřit vzdálenost t jako

$$t = \frac{p - o}{d}. \tag{2.22}$$

Po rozdělení ohraničujícího kvádru dle jednotlivých rozměrů i, je možné za bod p dosadit minima a maxima takto vyhrazených ohraničujících obdélníků („desek“), a zjistit tak vzdálenosti t do těchto extrémů pro každý daný rozměr i

$$\begin{aligned} t_i^{min} &= \frac{B_{minI} - o_i}{d_i} \\ t_i^{max} &= \frac{B_{maxI} - o_i}{d_i}. \end{aligned} \tag{2.23}$$

Bližší (1) a vzdálenější (2) vzdálenost průniku je pak jednoduše

$$\begin{aligned} t_i^1 &= \min\{t_i^{min}, t_i^{max}\} \\ t_i^2 &= \max\{t_i^{min}, t_i^{max}\}. \end{aligned} \tag{2.24}$$

Po získání těchto hodnot napříč všemi rozměry i lze vyjádřit jejich extrémy:

$$\begin{aligned} t^1 &= \min_i\{t_i^1\} \\ t^2 &= \max_i\{t_i^2\}, \end{aligned} \tag{2.25}$$

a s jejich pomocí definovat, že průnik existuje právě tehdy, kdy $t^1 \leq t^2$. Pro získání konkrétních bodů průniku pak stačí tyto hodnoty dosadit do původní definice paprsku:

$$\begin{aligned} p^1 &= o \ + t^1 d \\ p^2 &= o \ + t^2 d. \end{aligned} \tag{2.26}$$

Průnik paprsek–trojúhelník

Značně komplexnější je průnik typu **paprsek–trojúhelník**, klíčový pro zobrazování většiny tradičních trojrozměrných modelů. V počítačové grafice se nejčastěji vychází z algoritmu Möller-Trumbore [20]. Výhodou tohoto algoritmu je, že pro své fungování nepotřebuje počítat tzv. *podpůrnou rovinu*, tedy rovinu, na které testovaný trojúhelník leží, a která s ním má shodný normálový vektor. Algoritmus počítá s definicí bodu na trojúhelníku

$$\begin{aligned} P(u, v) &= (1 - u - v)V_0 + uV_1 + vV_2 \\ P(u, v) &= V_1 + u(V_2 - V_1) + v(V_3 - V_1), \end{aligned} \tag{2.27}$$

kde (u, v) jsou *barycentrické souřadnice* a (V_0, V_1, V_2) jsou jednotlivé body testovaného trojúhelníku. Bod průniku mezi paprskem a trojúhelníkem lze nalézt vztahem $p(t) = P(u, v)$

$$\begin{aligned} o\ + td &= V_1 + u(V_2 - V_1) + v(V_3 - V_1) \\ o\ - V_1 &= -td + u(V_2 - V_1) + v(V_3 - V_1). \end{aligned} \tag{2.28}$$

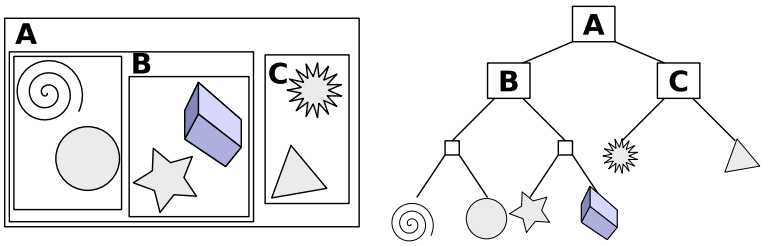
S touto rovnicí o třech neznámých (t, u, v) se nejčastěji pracuje ve vektorové formě za účelem co nejvyšší efektivity, zejména u simulací na grafických kartách.

$$\begin{bmatrix} -d, V_1 - V_0, V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O\ - V_0 \tag{2.29}$$

Po jejím vyřešení[7] proběhne kontrola, zdali jsou koeficienty u a v v intervalu $\langle 0, 1 \rangle$, a zdali $u + v < 1$. Pokud ano, bod průniku je získán dosazením do rovnice 2.27.

Objektové dělení scény (BVH)

Akcelerační struktury mají nejčastěji za úkol nahradit kvanta průniků paprsek–trojúhelník jedním velkým průnikem paprsek–kvádr a výrazně tak snížit počet kandidátů na průnik. Mezi nejčastěji používané akcelerační struktury patří *hierarchie hranic* (angl. bounding volume hierarchy, zkr. **BVH**) [26]. Jak lze vidět na obrázku 2.16, jde o stromovou strukturu, jejíž listy představují samotné objekty. Uzly poté reprezentují ohraničující kvádr pro své poduzly a listy.



Obrázek 2.16: BVH strom[8].

V ilustraci výše reprezentuje uzel A celou scénu. Paprsek, který touto scénou prochází, pak zprvu nemusí uvažovat jako kandidáty na průnik všechny trojúhelníky všech objektů, ale prozatím jen jejich ohraničující kvádry, tedy uzly B a C. Pokud tedy např. paprsek zaznamená průnik s kvádrem C, zmenšil tak seznam potenciálních kandidátů ze šesti objektů na pouhé dva. V komplexních scénách mohou být tyto stromy velmi rozsáhlé a časová náročnost se pak snižuje až na $O(\log(n))$, kde n je počet objektů ve scéně. Pro maximální efektivitu BVH je třeba zajistit, aby:

1. U sousedních uzlů byl minimální přesah.
2. Každý uzel měl co nejmenší objem.
3. Strom byl vyvážený co do počtu uzlů i objektů.

[7] Pozor na situaci, kdy je paprsek rovnoběžný s trojúhelníkem, a tedy kdy je determinant roven nule.
[8] Převzato z https://commons.wikimedia.org/wiki/File:Example_of_bounding_volume_hierarchy.svg.

Prostorové dělení scény (k-d stromy)

Další akcelerační strukturou jsou tzv. *k-d stromy*. Na rozdíl od BVH se zde scéna nedělí dle jednotlivých objektů, ale prostor scény je dělen rekurzivními řezy v k dimenzích. Může se tak stát, že vznikne uzel bez jakékoliv podléhající geometrie. Výsledné stromy dále můžou být výrazně hlubší a jejich paměťová náročnost tak mnohem vyšší. Při simulacích na grafických kartách se také nejčastěji využívá struktur BVH proto, že je jejich průchod hardwarově akcelerovaný.

2.8 Metody optimalizace

Jelikož je řada výpočtů v rámci path tracingu a dalších algoritmů, které zahrnuje, velice náročná, je vhodné uplatnit optimalizace všude, kde je to možné. Mezi ty nejčastější, relativně přímočaré implementace patří:

- **Ruská ruleta** [23] je metoda využívaná při path tracingu, díky které jsou cesty s minimálním přínosem předčasně ukončeny. Ukončení závisí na náhodě a platí, že čím menší je přínos dané cesty, tím vyšší je pravděpodobnost jejího ukončení. Ruská ruleta optimalizuje výkon, protože předchází kompletnímu průchodu „zbytečných“ cest.
- **Subpixelový posun** (angl. *subpixel jitter*) slouží jako základní forma antialiasingu. Každý nový vzorek má původ paprsku náhodně posunut v rámci svého pixelu, čímž ve výsledném snímku dochází k vyhlazení hran. Jde tak hlavně o vizuální optimalizaci.

Vzorkování dle důležitosti (importance sampling)

Metoda vzorkování dle důležitosti (angl. *importance sampling*) slouží ke snížení rozptylu Monte Carlo metod, a tedy i path tracingu. Je založená na předpokladu, že některé vzorky jsou pro zdárnou aproximaci důležitější než jiné, a proto může být výhodnější volit náhodná čísla z jiného než rovnoměrného rozdělení. Obecný vztah pro aproximaci integrálu funkce $f(x)$ pomocí metody Monte Carlo je možné zapsat následovně [22]:

$$E_{f(x)} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \tag{2.30}$$

kde X_i je náhodné číslo, $p(X_i)$ je pravděpodobnost jeho zvolení a $E_{f(x)}$ je aproximovaná hodnota integrálu $f(x)$. Při rovnoměrném rozložení bude např. při vzorkování difúzního odrazu $p(x)$ konstantní funkcí $p(x) = \frac{1}{2\pi}$. Běžně má však funkce $f(x)$ do rovnoměrného rozložení daleko a každý výsledný vzorek je tak značně odlišný. Zvyšuje se tak rozptyl jednotlivých vzorků [13].

Pokud by však generovaná čísla měla rozložení co nejbližší hodnotám funkce $f(x)$, rozptyl by se snížil. Proto se generovaná čísla mohou váhovat jinými, vhodně zvolenými funkcemi. V kontextu path tracingu jsou to např. BSDF, fázové funkce, nebo specificky navržené *naváděcí funkce*. Tímto postupem je možné získat více vzorků pro „důležitější“ cesty [13].

Kapitola 3

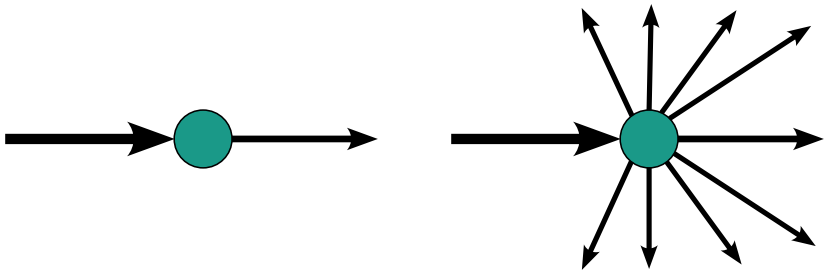
Zúčastněná média

Doposud byly zmíněny světelné interakce výhradně na rozhraní materiálů. Řada materiálů však světlo částečně, nebo úplně propouští a dále jej ovlivňuje také pod povrchem. Materiály, které světlo ovlivňují počínaje momentem, kdy se po interakci na základě BSDF paprsek zalomí dovnitř objektu, a konče okamžikem, kdy z něj opět vystoupí, se v angličtině označují jako *participating media* [13]. Přeneseně tak jde o materiály (příp. prostředí), které se světelných interakcí taktéž „účastní“ tím, že jej rozptylují nebo absorbují.

Pro účely simulací se média běžně považují za shluk mikroskopických částic. Vzhledem k tomu, že jsou částice mikroskopické a náhodně rozmístěné, není nutné simulovat každou částici samostatně. Místo toho se s médiem pracuje jako s celkem a světelné interakce probíhají na bázi pravděpodobností.

3.1 Světelné interakce v médiích

Když cestuje paprsek médiem, může veškeré částice minout a pokračovat beze změny, nebo může s některými částicemi interagovat. Když k interakci dojde, mohou nastat dvě události, při kterých dojde k útlumu vstupní záře (viz obr. 3.1): Část světelného paprsku bude absorbována, nebo bude paprsek rozptýlen do jiného směru. Tyto jevy se nazývají **absorpce** (angl. *absorption*) a **výstupní rozptyl** (angl. *out-scattering*) [13].



(a) Absorpce (b) Výstupní rozptyl

Obrázek 3.1: Interakce utlumující vstupní zář.

Jev absorpce je možné popsat vztahem (všechny rovnice v sekci 3.1 jsou převzaty z disertační práce *Efficient Monte Carlo Methods for Light Transport in Scattering Media*, W. Jarosz, 2008 [13]):

$$(\vec{\omega} \cdot \nabla_a) L(x, \vec{\omega}) = -\sigma_a(x) L(x, \vec{\omega}), \tag{3.1}$$

kde ∇_a je absorpční gradient, $(\vec{\omega} \cdot \nabla_a)$ počítá směrovou derivaci ve směru paprsku $\vec{\omega}$, $\sigma_a(x)$ je **absorpční koeficient** a $L(x, \vec{\omega})$ je vstupní zář.

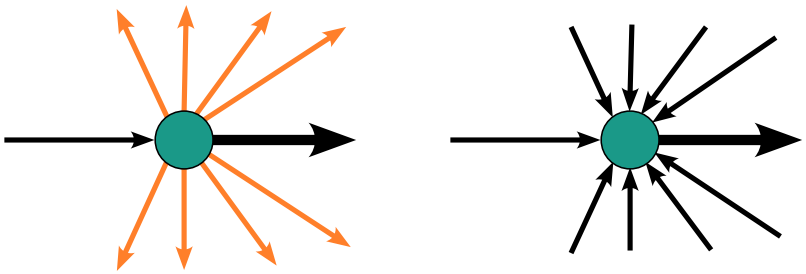
Velmi podobně lze popsat také jev výstupního rozptylu:

$$(\vec{\omega} \cdot \nabla_o) L(x, \vec{\omega}) = -\sigma_s(x) L(x, \vec{\omega}), \tag{3.2}$$

kde jediným rozdílem je veličina σ_s, známá jako **rozptylový koeficient**, a ∇_o, tedy gradient výstupního rozptylu. Celkový úbytek záře vlivem jednoho z těchto jevů se nazývá **zánik** (angl. *extinction*) a může být vyjádřen jako:

$$\begin{aligned}(\vec{\omega} \cdot \nabla_t) L(x, \vec{\omega}) &= -(\sigma_a(x) + \sigma_s(x)) L(x, \vec{\omega}) \\ &= -\sigma_t(x) L(x, \vec{\omega}).\end{aligned} \tag{3.3}$$

V některých případech může dojít také ke zvýšení záře (viz obr. 3.2). Když je vstupní paprsek rozptýlen – dojde k útlumu jeho záře, v jiném bodě média však dojde naopak k **navýšení** o stejnou hodnotu. Materiál může být dále sám o sobě **emisivní** a průchozím paprskům tak přidávat zář (např. oheň). Proto se v zúčastněných médiích uvažují další dva jevy – **emise** (angl. *emission*) a **vstupní rozptyl** (angl. *in-scattering*) [13].



(a) Emise (b) Vstupní rozptyl

Obrázek 3.2: Interakce navyšující vstupní zář.

Jev emise je možné popsat vztahem:

$$(\vec{\omega} \cdot \nabla_e) L(x, \vec{\omega}) = \sigma_a(x) L_e(x, \vec{\omega}), \tag{3.4}$$

kde L_e je zář spontánně vydávaná médiem. Obdobně se zapisuje i poslední jev, vstupní rozptyl:

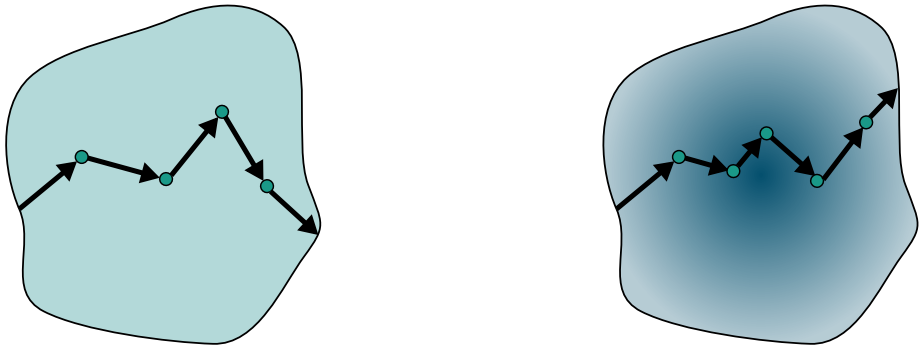
$$(\vec{\omega} \cdot \nabla_i) L(x, \vec{\omega}) = \sigma_s(x) L_i(x, \vec{\omega}), \tag{3.5}$$

kde L_i je vstupní zář, nebo přesněji zářivý tok ze všech okolních prostorových úhlů obdržený daným bodem interakce.

3.2 Typy a kategorie médií

Při výpočtech výše bylo využíváno absorpčního σ_a a rozptylového σ_s koeficientu. Když jsou tyto hodnoty konstantní napříč celým médiem, lze takové médium označit jako **homogenní**. Většinou jsou však skutečná média **heterogenní** a koeficienty se tak mohou více či méně lišit napříč médiem (viz obr. 3.3). Ve vztahu 3.3 došlo mj. k substituci absorpčního a rozptylového koeficientu za **koeficient zániku** $\sigma_t = \sigma_a + \sigma_s$ [13].

Další významnou veličinu lze vyjádřit poměrem $\frac{\sigma_s}{\sigma_t}$. Jde o tzv. *rozptylovou odrazivost* (angl. **scattering albedo**) – veličinu udávající pravděpodobnost, že v daném místě v médiu



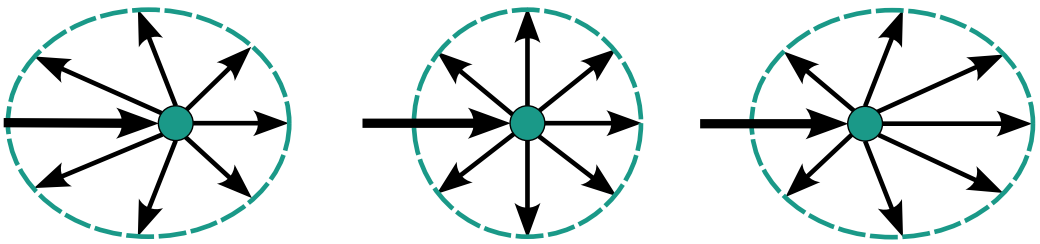
(a) Homogenní (b) Heterogenní

Obrázek 3.3: Typy médií v závislosti na proměnlivosti koeficientů. V „hustějším“ středu heterogenního média může např. k interakcím docházet častěji.

dojde k rozptylu světla. Je-li tato hodnota nulová, částice daného média vůbec nerozptylují světlo. Naopak, je-li hodnota rovna jedné, částice světlo neabsorbují [13].

Izotropie a anizotropie

Média lze rozdělit ještě jedním způsobem, a to na **izotropní** a **anizotropní**. Tuto vlastnost vyjadřuje **parametr asymetrie** $g \in \langle -1, 1 \rangle$ (viz obr. 3.4). Ve své fyzikální podstatě jde o průměrnou hodnotu kosinů všech rozptýlených směrů. V praxi potom platí, že izotropní média, tedy ta s $g = 0$, rozptylují světlo do všech směrů rovnoměrně. Anizotropní média pak mohou mít buď hodnotu $g > 0$ a rozptylovat **dopředně** (angl. *forward-scattering*), nebo $g < 0$ a rozptylovat **zpětně** (angl. *back-scattering*). Obdobně, jako u klasických světelných interakcí udává směr putování paprsku BRDF (či BSDF, BSSRDF, viz kap. 2.2), slouží k určení směru rozptýlení tzv. **fázová funkce** [13].



(a) Zpětný rozptyl ($g < 0$) (b) Izotropní rozptyl ($g = 0$) (c) Dopředný rozptyl ($g > 0$)

Obrázek 3.4: Rozptyl v závislosti na parametru asymetrie.

3.3 Fázové funkce

Stejně jako existuje nespočet funkcí pro řešení klasických světelných interakcí, existuje i mnoho fázových funkcí popisujících rozptyl uvnitř média. Každá taková funkce musí být [23]:

1. **Reciproká**. Musí dodržovat Helmholtzův zákon o reciprocitě, a tedy musí platit vztah

$$p(x, \omega_i \to \omega_o) = p(x, \omega_o \to \omega_i). \tag{3.6}$$

2. **Normalizovaná** čili její integrál napříč všemi směry musí být roven právě jedné [13]:

$$\int_\Omega p(x, \omega_i \leftrightarrow \omega_o) d\omega_i = 1, \forall \omega_o. \tag{3.7}$$

Stejně jako média mohou být i fázové funkce izotropní nebo anizotropní. Ovšem vzhledem k tomu, že jsou fázové funkce normalizované, existuje a může existovat právě jedna **izotropní** fázová funkce [23]:

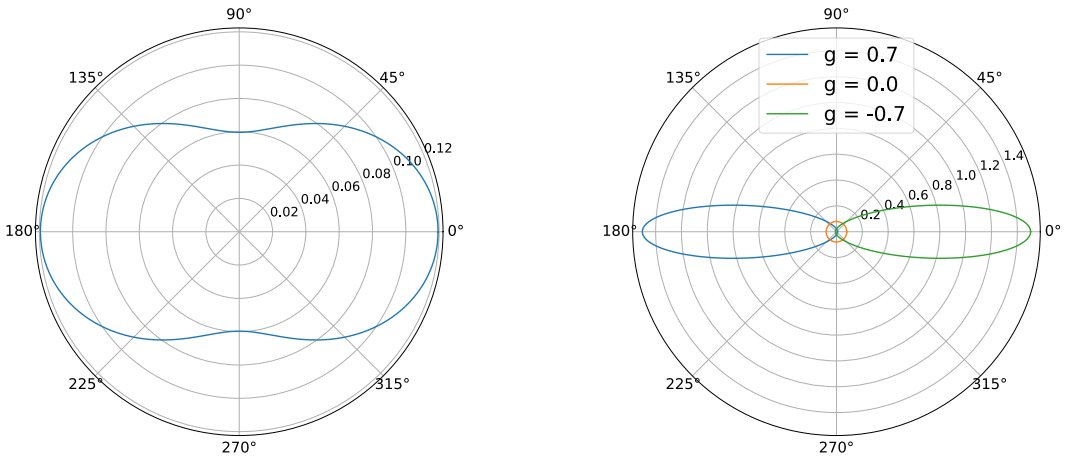
$$p(x, \omega_o \to \omega_i) = \frac{1}{4\pi}. \tag{3.8}$$

Rayleighova fázová funkce

Rayleighova fázová funkce je založena na světelných interakcích s jednotlivými částicemi, které jsou mnohem menší než vlnová délka světla. Své využití nachází především v simulacích rozptylu světla v atmosférách planet. Typicky se zapisuje vztahem [4]:

$$p(\theta) = \frac{1}{4\pi}\frac{3}{4}(1 + \cos^2\theta), \tag{3.9}$$

kde θ je úhel rozptylu, tedy úhel mezi ω_o a ω_i. Její polární graf je možné vidět v obr. 3.5.



(a) Rayleighova fázová funkce

(b) Henyey-Greensteinova fázová funkce

Obrázek 3.5: Srovnání polárních grafů fázových funkcí.

Henyey-Greensteinova fázová funkce

Svou fázovou funkci představili také Louis G. Henyey a Jesse L. Greenstein roku 1941 [11]. Ve svém výzkumu se zabývali rozptylem světla v mezihvězdném prachu a jejich model mj. zahrnuje i výše zmíněný parametr asymetrie g (viz obr. 3.5). Matematicky lze zapsat jako:

$$P(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}}. \tag{3.10}$$

3.4 Zobrazovací rovnice uvažující zúčastněná média

V sekci 2.4 byla uvedena zobrazovací rovnice. Její zápis (rovnice 2.19) však neuvažuje interakce v zúčastněných médiích. Spojením rovnic 3.3, 3.4 a 3.5 vznikne vztah pro celkovou změnu záře v médiu, známý také jako **rovnice přenosu záření** [13]:

$$(\vec{\omega} \cdot \nabla)L(x, \vec{\omega}) = -\sigma_a(x)L(x, \vec{\omega}) - \sigma_s(x)L(x, \vec{\omega}) + \sigma_a(x)L_e(x, \vec{\omega}) + \sigma_s(x)L_i(x, \vec{\omega}). \tag{3.11}$$

Po integraci rovnice přenosu záření je možné s použitím klasické zobrazovací rovnice vyjádřit čistě integrální vztah pro výpočet záře za přítomnosti zúčastněného média [13]:

$$\begin{aligned} L(x, \vec{\omega}) = {} & T_r(x \leftrightarrow x_s)L(x_s \rightarrow \vec{\omega}) + \\ & \int_0^s T_r(x \leftrightarrow x_t)\sigma_a(x)L_e(x \rightarrow \vec{\omega})\ dt + \\ & \int_0^s T_r(x \leftrightarrow x_t)\sigma_s(x_t)L_i(x_t \rightarrow \vec{\omega})\ dt, \end{aligned} \tag{3.12}$$

kde:

- $L(x, \vec{\omega})$ je zář přicházející do bodu x ze směru $\vec{\omega}$,
- x_s je bod na povrchu nacházejícím se za médiem,
- s je hloubka média z pohledu bodu x ve směru $\vec{\omega}$,
- T_r je tzv. **propustnost** média (angl. *transmittance*).

Propustnost udává pravděpodobnost, že médium propustí světelnou částici mezi dvěma danými body, v přímé linii, a modeluje tak úbytek záře po dobu cestování médiem. Částice, které propuštěny nejsou, byly buď absorbovány, nebo rozptýleny. Propustnost obecně udává vztah [13]

$$T_r(x' \leftrightarrow x) = \exp\left(-\int_0^d \sigma_t(x + t\vec{\omega})dt\right), \tag{3.13}$$

kde $(x + t\vec{\omega})$ je parametrický zápis segmentu mezi body x a x'. V případě homogenních médií, kde je σ_t konstantní, je možné zápis zjednodušit:

$$T_r(x' \leftrightarrow x) = \exp\left(-||x' - x||\sigma_t\right). \tag{3.14}$$

Jelikož je emisní část rovnice 3.12 triviální na výpočet, často se ze zobrazovací rovnice vynechává. Finální vztah se pak běžně označuje jako **objemová zobrazovací rovnice** [13]:

$$L(x, \vec{\omega}) = T_r(x \leftrightarrow x_s)L(x_s \rightarrow \vec{\omega}) + \int_0^s T_r(x \leftrightarrow x_t)\sigma_s(x_t)L_i(x_t \rightarrow \vec{\omega})\ dt, \tag{3.15}$$

a při zobrazování scén, kde se uvažuje rozptyl a absorpce zúčastněných médií, doplňuje Immelovu zobrazovací rovnici ze sekce 2.4. Ta je v objemové zobrazovací rovnici reprezentována výrazem $L(x_s \rightarrow \vec{\omega})$.

Vzorkování vzdálenosti interakce

Jak bylo definováno ve vztazích 3.13 a 3.14, propustnost je vždy udávána mezi dvěma body. Aby tedy bylo možné průchod médiem simulovat, je tedy nutné určit, v jaké vzdálenosti od bodu x se onen bod x', ve kterém nastane další interakce, nachází. Jelikož propustnost udává *pravděpodobnost*, že mezi dvěma body světelné částice projdou bez interakce, vzdálenost těchto bodů je možné získat použitím metody *importance sampling*. Vzdálenost d tak lze vyjádřit vztahem [13]:

$$d(x) = -\frac{\log(\xi)}{\overline{\sigma}_t(x)}, \tag{3.16}$$

kde ξ je náhodné číslo z intervalu $(0, 1\rangle$. Odpovídající funkce hustoty pravděpodobnosti je pak:

$$p(d) = e^{-\overline{\sigma}_t(x)d}. \tag{3.17}$$

Výraz $\overline{\sigma}_t(x)$ může být definován jako libovolné nezáporné číslo, pro účely vzorkování vzdálenosti interakce je však vhodné tuto hodnotu zvolit tak, aby výsledné vzdálenosti z rovnice 3.16 měly rozdělení úměrné k propustnosti T_r. Pro **homogenní média** tohoto lze docílit velmi snadno. Stačí za tuto hodnotu přímo dosadit *koeficient zániku* média, tedy $\overline{\sigma}_t(x) = \sigma_t$. Výše zmíněná funkce hustoty pravděpodobnosti tak bude k propustnosti T_r přímo úměrná. V **heterogenních médiích** je zajištění hodnoty přímo úměrné k propustnosti výrazně náročnější a obecně je nutné ji aproximovat [13].

Kapitola 4

Simulační model

Následující kapitola pojednává o tom, jak byla teorie a koncepty z předcházejících kapitol přeneseny do simulačního modelu. Definuje předpoklady a cíle simulačního modelu a dále popisuje metody a algoritmy, jež byly zvoleny pro jeho implementaci.

4.1 Předpoklady a cíle simulace

Hlavním cílem simulace je řešení objemové zobrazovací rovnice, která byla definována v sekci 3.4, a to efektivně, ale zároveň tak, aby byla výsledná implementace dobře srozumitelná. Aby bylo těchto cílů dosaženo, simulační model zavádí celkem tři hlavní předpoklady, které vedou buď na zjednodušení výpočtu, jeho urychlení, nebo na oboje najednou.

Simulační model předpokládá, že nebude použit k simulaci emisivních zúčastněných médií. Taková média v přírodě sice existují (např. oheň), je jich však, vzhledem ke všem ostatním, málo. V simulačním modelu a jeho následné implementaci by přidávala emisivní média vrstvu komplexity, a proto nejsou uvažována.

Další předpoklad se týká vstupního rozptylu. Jeho rovnice 3.5 uvažuje vstupní zář $L_i(x, \vec{\omega})$, tedy zářivý tok přicházející ze všech okolních prostorových úhlů. Aby byl výpočet této hodnoty korektní, pro každý bod po cestě světelného paprsku by bylo třeba vysledovat všechny okolní paprsky, pro které taktéž došlo v daném bodu k interakci.

Takový výpočet je však prakticky velmi obtížně dosažitelný. Proto, zatímco jev výstupního rozptylu je sledován kompletně (angl. jde o tzv. **multiple scattering**, tedy „mnohonásobný rozptyl“), vstupní rozptyl se uvažuje pouze ve směru světelného zdroje (angl. **single scattering**, tedy „jednonásobný rozptyl“). V daném bodě se tedy pouze rozhoduje o tom, zdali je bod vůči světelnému zdroji ve stínu, nebo ne.

Tento předpoklad nemá na výsledné zobrazení až takový vliv, jak by se mohlo zdát. Ukazuje se totiž, že když je u velkého množství paprsků simulován výstupní rozptyl, vstupní rozptyl je tak postupně aproximován.

Třetím a posledním předpokladem je, že simulační model bude využit výhradně pro homogenní média. Heterogenní média vyžadují výrazně složitější způsob reprezentace a dále komplikují řadu výpočtů, které pro homogenní média často ani není nutné uvažovat. Pro zjednodušení výsledné implementace jsou tak heterogenní zúčastněná média ze simulačního modelu vynechána.

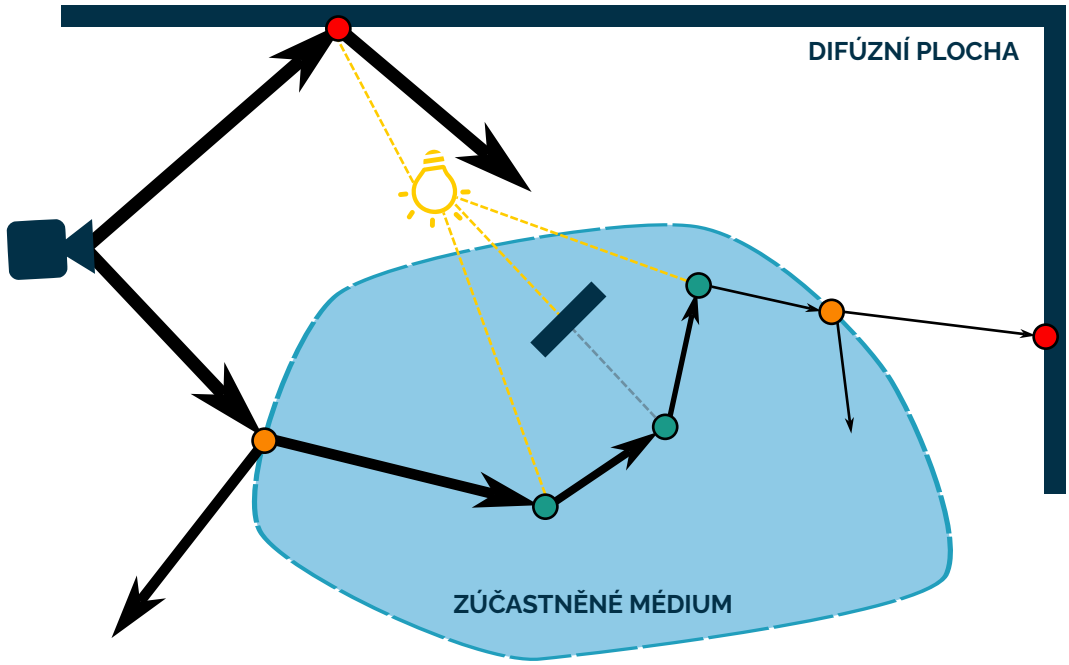
4.2 Základní zobrazovací model

Pro realistické zobrazování byla vybrána metoda **sledování cesty** (path tracing). Při testování na průnik paprsku s geometrií jsou využívány akcelerační struktury, konkrétně hierarchie hranic (BVH). Zobrazovací model je možné rozdělit na dvě hlavní části:

1. **Path tracer** se stará o generování světelných paprsků, jejich sledování a jejich interakce s povrchy (lom a odraz na základě Fresnelových vztahů, vzorkování BRDF). Nakonec se také stará o zpracování výsledků. Pro základní přehled o fungování algoritmu viz sekci 2.6.

2. **Průchod médiem** začíná v momentě, kdy je do něj paprsek zalomen. Model průchodu médiem zodpovídá za výpočet propustnosti. Podrobněji je popsán v sekci níže.

Obrázek 4.1 zobrazuje schéma zobrazovacího modelu se zvýrazněnými interakcemi. Oranžovou barvou jsou zvýrazněny interakce, při kterých path tracer počítá Fresnelovy rovnice a rozhoduje, zdali se paprsek zalomí, nebo odrazí. Červenou barvou jsou zobrazeny plně difúzní interakce řešeny path tracerem pomocí Lambertovy BRDF. Zeleně jsou pak vyznačeny interakce vznikající v rámci průchodu médiem. Scéna uvažuje právě jeden bodový světelný zdroj.



Obrázek 4.1: Schéma zobrazovacího modelu.

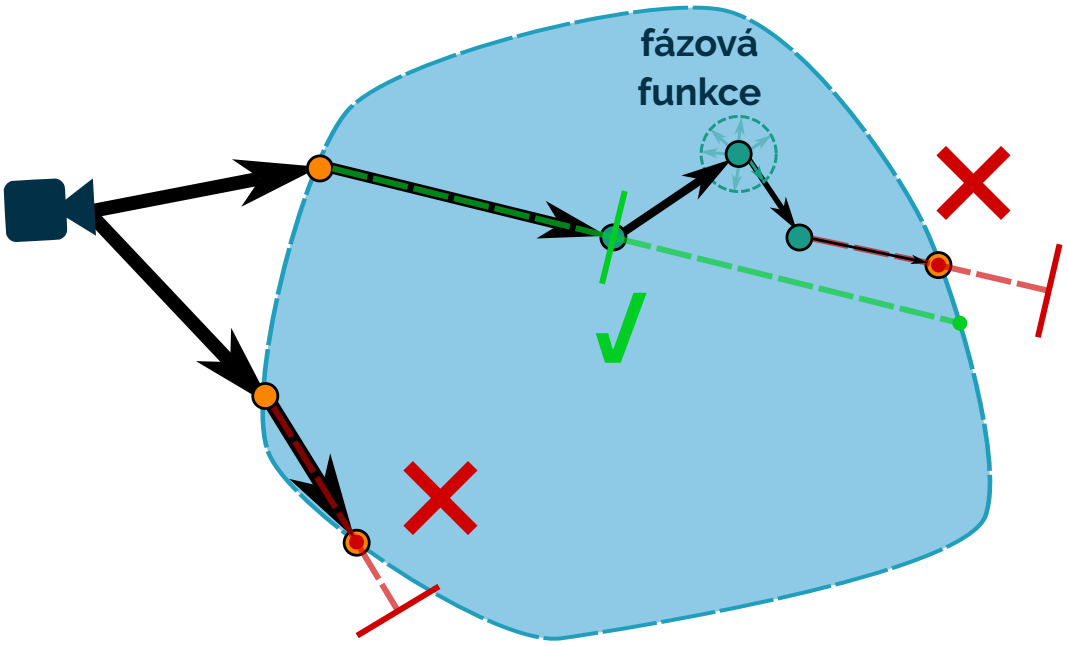
Path tracer si uchovává informaci o celkové propustnosti po průchodu (nebo průchodech) médiem a může s ní tak při výpočtech pracovat. V kontextu objemové zobrazovací rovnice 3.15 zodpovídá za výpočet výrazu

$$T_r(x \leftrightarrow x_s)L(x_s \rightarrow \vec{\omega}), \tag{4.1}$$

kde $T_r(x \leftrightarrow x_s)$ je právě celková propustnost a $L(x_s \rightarrow \vec{\omega})$ je celková zář v daném bodě difúzní interakce, získaná výpočtem Immelovy zobrazovací rovnice (viz rovnice 2.19).

4.3 Model průchodu médiem

Jak už bylo zmíněno, průchod médiem započíná tehdy, kdy se paprsek do média zalomí na základě Fresnelových vztahů. Nejdříve je třeba navzorkovat vzdálenost d první interakce s médiem, tedy z bodu vstupu do média x, do nějakého bodu x_t v médiu. Uplatní se zde vztah 3.16. Zároveň je však třeba provést kontrolu, zdali se tato vzdálenost médium nepřekročí. Proto je vyslán jednorázový paprsek, jehož účelem je zjistit nejbližší průnik ať už s médiem, nebo s jiným objektem. K průniku dojde ve vzdálenosti t. Pokud $d < t$, bod x_t se nachází ve vzdálenosti d od x. Pokud ne, bod x_t je nastaven do vzdálenosti t, tedy na konec média, jak je možné vidět na obrázku 4.2.



Obrázek 4.2: Schéma průchodu médiem.

Je-li určena pozice bodu x_t, není již problém vypočítat propustnost $T_r(x \leftrightarrow x_t)$ za pomoci vztahu 3.14. Tato částečná propustnost je pak aplikována na průběžnou hodnotu $T_r(x \leftrightarrow x_s)$, kterou si, jak bylo zmíněno, uchovává path tracer pro výpočet porce záře z difúzních objektů za médiem.

Po nalezení světelného zdroje metodou single scattering, a tedy získání konkrétní hodnoty $L_i(x_t \rightarrow \vec{\omega})$, je již možné vypočítat částečnou zář pro tuto interakci. Probíhá zde výpočet vzorku L_t integrálu v objemové zobrazovací rovnici:

$$L_t = T_r(x \leftrightarrow x_t)\sigma_s(x_t)L_i(x_t \rightarrow \vec{\omega}). \tag{4.2}$$

Nakonec je třeba určit směr, kam se paprsek dále odrazí. V sekci 3.3 byly vysvětleny fázové funkce, které slouží právě k tomuto účelu. Pro simulaci průchodu médiem byla zvolena **Henyey-Greensteinova** fázová funkce, přičemž hodnota parametru asymetrie g je konstantní součástí definice každého média. Po rozptylu se celý proces opakuje, až dokud paprsek z média nevystoupí.

4.4 Použitá radiometrická data

Simulační model využívá fyzikálně věrná radiometrická data, experimentálně zjištěná na základě reálných materiálů. Jejich měření není předmětem této práce, a proto byla data převzata ze článku *Acquiring Scattering Properties of Participating Media by Dilution* [21]. Ten obsahuje data měřená pro roztoky různých zúčastněných médií a vody, a proto byla pro účely simulace upravena tak, aby hodnoty odpovídaly čistým kapalným médiím (postup pro tento výpočet je součástí článku).

Tato práce byla jako zdroj dat vybrána ze dvou důvodů. V první řadě obsahuje rozptylové koeficienty σ_s a koeficienty zániku σ_t (ze kterých lze snadno vypočítat také koeficient absorpce σ_a), stejně jako hodnoty asymetrického parametru g. Popis zúčastněných médií je zde tak zcela kompletní. Druhým důvodem je, že na rozdíl od většiny zdrojů nejsou data uváděna spektrálně, nýbrž pro jednotlivé RGB složky. Pro účely počítačové grafické simulace jsou tak tyto hodnoty ideální. V tabulce 4.1 jsou data pro několik vybraných materiálů upravená tak, aby mohla být v simulaci použita. **Prvních osm** materiálů pochází ze zmíněného článku a **jsou tak fyzikálně věrné**, **zbylé čtyři** materiály byly vytvořeny **zcela uměle** a jde pouze o velmi vzdálenou imitaci.

Materiál	$\sigma_s(mm^{-1})$			$\sigma_a(mm^{-1})$			g		
	R	G	B	R	G	B	R	G	B
Mléko	18.2052	20.3826	22.3698	0.00153	0.00460	0.01993	0.750	0.714	0.681
Espresso	7.78262	8.13050	8.53875	4.79838	6.57512	8.84925	0.907	0.896	0.880
Sprite	0.00011	0.00014	0.00014	0.00189	0.00183	0.00200	0.943	0.953	0.952
Coca Cola	0.00254	0.00299	0.00000	0.10014	0.16503	0.24680	0.965	0.972	0.000
Jablečný džus	0.00257	0.00311	0.00413	0.01296	0.02374	0.05218	0.947	0.949	0.945
Hroznový džus	0.00138	0.00000	0.00000	0.10404	0.23958	0.29325	0.961	0.000	0.000
Chardonnay	0.00021	0.00033	0.00048	0.01078	0.01186	0.02400	0.914	0.958	0.975
Budweiser pivo	0.00029	0.00055	0.00059	0.01149	0.02491	0.05579	0.917	0.956	0.982
Smaragd	0.18000	0.07000	0.03000	0.97000	0.06100	1.45000	0.943	0.953	0.952
Rubín	0.18000	0.07000	0.03000	0.06100	0.97000	1.45000	0.943	0.953	0.952
Safír	0.18000	0.07000	0.03000	0.97000	1.45000	0.06100	0.943	0.953	0.952
Sklo	0.00011	0.00014	0.00014	0.00189	0.00183	0.00200	0.943	0.953	0.952

Tabulka 4.1: Hodnoty koeficientů zúčastněných médií zahrnutých do výchozího katalogu materiálů simulace. Pro barevné složky, kde $\sigma_s = 0$, není definována hodnota g.

Pro veškeré kapaliny je uvažován index lomu $n = 1.33$. Pro smaragd je $n = 1.52$, pro safír i rubín $n = 1.77$. „Sklo“, byť koeficienty ve skutečnosti Sprite, má oproti Spritu o něco vyšší index lomu, $n = 1.5$.

Kapitola 5

Implementace realistické simulace

V následující kapitole je popsána samotná implementace vybraných metod z teoretických sekcí práce na základě návrhu z předchozí kapitoly.

Implementace je rozdělena na dvě části. První, psána v jazyce C++, je určena pro běh na procesoru (CPU) a slouží k nastavení parametrů scény, načtení vstupu, nastavení API Vulkan atd. Druhou částí je samotná implementace simulace formou tzv. *shaderu* v jazyce GLSL, určeného pro běh na grafickém procesoru (GPU).

5.1 Použité technologie

Nejdůležitější technologií, kterou implementace využívá, je **Vulkan API**. Jde o rozhraní poskytující přímou kontrolu nad GPU se zaměřením na maximální výkonnost. Mezi klíčové výhody patří [8]:

- **Nezávislost na platformě.** Vulkan běží nativně na OS Windows (7 a vyšší), Linux, BSD Unix, Raspberry Pi a řadě dalších. Jako nadstavba nad Metal API od Applu lze spustit i na zařízeních s macOS, iOS, nebo tvOS.
- **Nízké využití CPU.** Vulkan automaticky využívá dávkování a nízkoúrovňové optimalizace pro co nejnižší zátěž na CPU. Oproti starším API také dokáže pracovat s více vlákny procesoru.
- **Předkompilované shadery.** Ovladače Vulkanu přijímají shadery v jednotném mezijazyce SPIR-V, do kterého lze zkompilovat různé jazyky vyšších úrovní, jako např. GLSL, HLSL nebo MSL. Ovladače na grafické kartě tak provádí jen optimalizace a generování kódu specificky pro danou GPU a nemusí implementovat vlastní překladač.

Vulkan SDK[1] ve verzi **1.3.250.1**. Jde o sadu celé řady programů užitečných při vývoji grafických programů v prostředí Vulkan API. Implementace z SDK používá:

- **Vulkan API** ve verzi 1.3.0.
- **Glslang** ve verzi 12.2.0, coby validátor jazyka GLSL a zároveň kompilátor zdrojových souborů v jazyce GLSL do mezijazyka SPIR-V – reprezentace používané rozhraním Vulkan. Nespornou výhodou mezijazyka SPIR-V je nativní reprezentace všech grafických primitiv. Samotný jazyk GLSL je použit ve **verzi 4.6**.

[1] Ke stažení na webu https://www.lunarg.com/vulkan-sdk/.

- **VVL** (Vulkan Validation Layers) ve verzi 1.3.255. Rozhraní Vulkan dává vývojáři přímou kontrolu nad GPU, s minimální kontrolou chyb pro maximální výkon. Aplikace tak plně zodpovídají za korektní používání Vulkan API a jakákoliv chyba může vést k pádu programu bez chybové hlášky. Validační vrstvy dokážou některé nekorektní používání zachytit a pomáhají tak s vývojem.
- **Další komponenty**, které Vulkan API využívá na pozadí.

nvpro_core[2], konkrétně moduly *nvvk* a *nvh*. Jde o výběr ze sady Nvidia DesignWorks obsahující lehké struktury a funkce nad rozhraním Vulkan API. Jejich účelem je minimalizovat boilerplate kód, ale zároveň zachovat co nejvíce přímé kontroly.

tinyobjloader[3] ve verzi 2.0 (Release Candidate). Program sloužící k načítání 3D modelů a scén ve formátu Wavefront OBJ a jejich materiálů ve formátu Wavefront MTL.

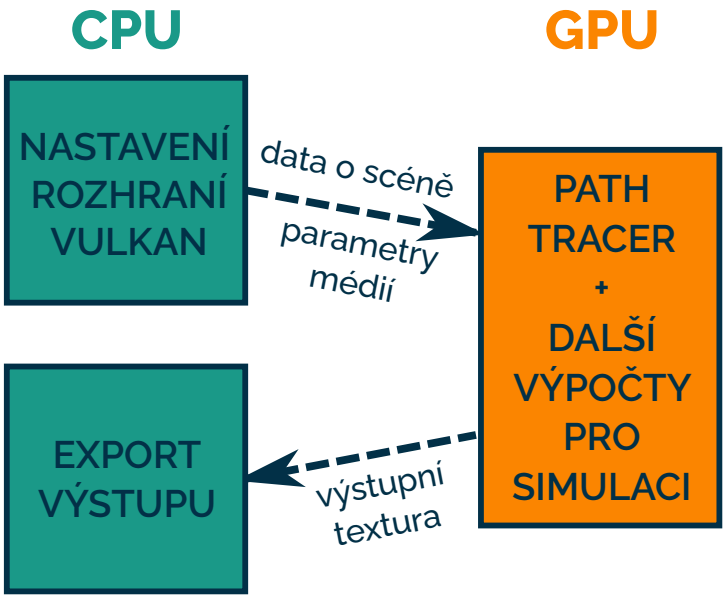
JSON for Modern C++[4] ve verzi 3.10.4. Program používaný pro načítání JSON souborů s definicemi zúčastněných médií pro danou scénu.

Jazyk C++ dle standardu ISO C++17, kompilovaný pomocí **MSVC 14.38.33130**.

Jazyk Python ve verzi 3.10.0. Využit pro usnadnění vytváření souborů s definicemi médií.

5.2 Architektura implementace na CPU a GPU

Jak bylo zmíněno v úvodu kapitoly, implementace je rozdělena na dva programy – tzv. „hostitelský“ CPU kód psaný v jazyce C++ a tzv. kód „zařízení“ čili shader v jazyce GLSL, určený pro běh na GPU. Oba programy mezi sebou nepřímo komunikují a na nejvyšší úrovni je možné architekturu rozdělit do tří fází, znázorněných na obrázku 5.1.



Obrázek 5.1: Diagram fází implementace na CPU a GPU.

[2]GitHub nvpro_core: https://github.com/nvpro-samples/nvpro_core/.
[3]GitHub tinyobjloader: https://github.com/tinyobjloader/tinyobjloader.
[4]GitHub JSON for Modern C++: https://github.com/nlohmann/json.

Nastavení rozhraní Vulkan

Detailnější diagram první fáze je na obrázku 5.2. Nastavování rozhraní Vulkan začíná tvorbou reprezentace zařízení – GPU – v kódu. V této fázi probíhá také žádání o rozšíření potřebná pro běh simulace. Jsou to rozšíření:

- `VK_KHR_RAY_QUERY` – umožňuje shaderům využívat volání funkcí *ray query* pro vysoce efektivní hardwarově optimalizované trasování paprsků. Rozšíření vyžaduje vcelku moderní grafické procesory, které tuto hardwarovou akceleraci podporují.
- `VK_KHR_DEFERRED_HOST_OPERATIONS` – nutná závislost `VK_KHR_RAY_QUERY`. Umožňuje shaderu využívat paralelizace a vícevláknového zpracování.
- `VK_KHR_ACCELERATION_STRUCTURE` – nutná závislost pro `VK_KHR_RAY_QUERY`. Přidává do shaderu podporu akceleračních struktur.

Po zavolání inicializační funkce se Vulkan pokusí najít v počítači takový grafický procesor, který podporuje všechna žádaná rozšíření. Pokud se mu to povede, hostitelský program může se zařízením začít komunikovat. V další fázi dochází k alokování zdrojů, načítání souborů scény a jejich převod na akcelerační struktury (zkr. AS) a dále také k načítání shaderu. Pro všechny zdroje jsou dále vytvořeny tzv. *deskriptory*. Tento proces podrobně popisuje sekce 5.3 níže.

Obrázek 5.2: Detailnější diagram fáze nastavení rozhraní Vulkan.

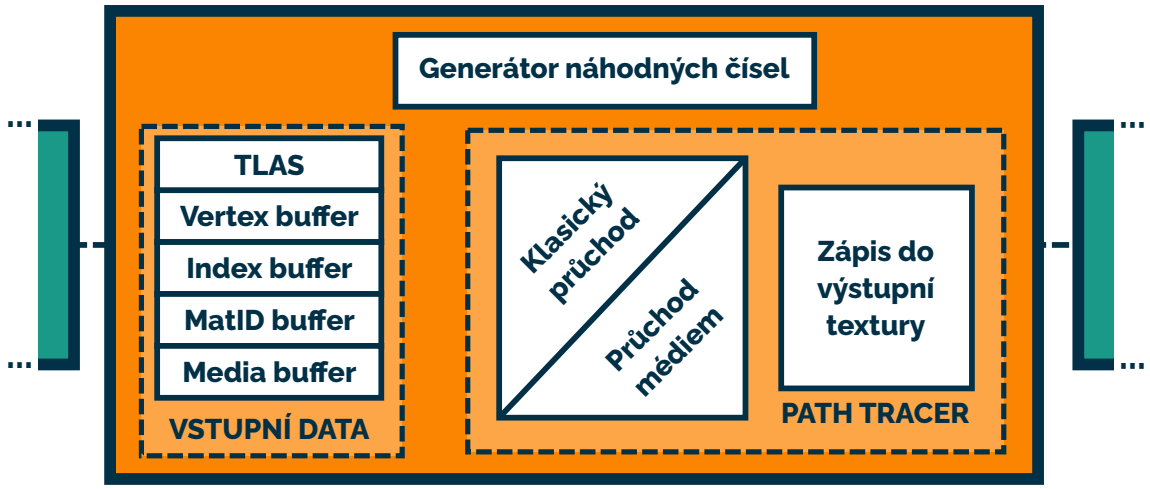
Všechny zdroje jsou v závěru první fáze „spjaty“ (angl. *bind*) s tzv. **příkazovým bufferem**. Z hlediska optimalizace je vhodné co nejvíce volání spojit do jednoho příkazového bufferu a ten pak odeslat s optimálním nastavením paralelizace. Vulkan umožňuje ve funkci `vkCmdDispatch` přímo nastavit, jak velká pracovní skupina se má danému příkazovému bufferu věnovat.

V implementaci byl zvolen postup rovnoměrného rozložení dle rozměrů výstupního obrázku. Jedna pracovní skupina grafických čipů Nvidia je schopná zpracovat 32 paralelních operací, u čipů AMD je to 64 operací. Skupiny se indexují souřadnicemi x, y, z.

Byť nemusí být v daném čase dostupná skupina celá (např. ji využívá jiný program), tyto hodnoty k obstojnému rozložení poslouží. Implementace ve výchozím nastavení pracuje s kartou Nvidia, v rozlišení Full HD, a tak se příkazový buffer odesílá s hodnotami $x = \lceil 1920/32 \rceil = 60$, $y = \lceil 1080/32 \rceil = 34$ a $z = 1$.

Simulace na GPU

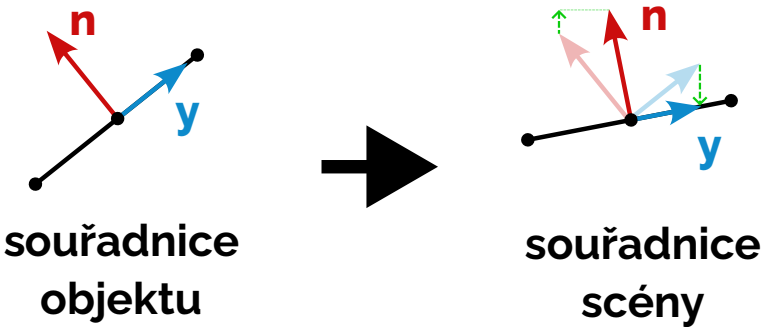
Poté, co je shader nahrán spolu s potřebnými daty na GPU, může začít samotná simulace. V shaderu je naimplementován samotný path tracer s podporou zúčastněných médií, který blíže popisují sekce 5.5 a 5.6, jsou zde ale i různé pomocné funkce a struktury. Blokové schéma je možné vidět na obrázku 5.3.



Obrázek 5.3: Detailnější diagram fáze GPU simulace.

Mezi nejdůležitější patří struktura `HitInfo` a s ní související funkce `getObjectHitInfo` zodpovídající za zpracování výstupu z trasování paprsků pomocí *ray query* a jejich interpretaci v závislosti na vstupních datech. Rozšíření ray query umožňuje získat identifikátor grafického primitivu (v rámci implementace jde o trojúhelník), který paprsek protnul. S tímto identifikátorem je možné z *vertex bufferu* získat všechny tři vrcholy, které jej tvoří.

Ray query dále umožňuje získat barycentrické souřadnice u a v konkrétního průniku. S použitím vztahu 2.27 je pak snadné vyjádřit konkrétní souřadnice průniku, ovšem v objektovém prostoru (angl. *object space*). Pro převedení do prostoru scény (angl. world space) je třeba použít transformační matici, kterou taktéž může poskytnout rozšíření ray query (viz vektor y na obr. 5.4).



Obrázek 5.4: Srovnání transformace normály a vrcholů trojúhelníku (jen jedna osa).

Z vrcholů je možné vypočítat také tzv. *stěnovou normálu*, a to jednoduše pomocí pravidla pravé ruky – je totiž třeba určit směr, který je na trojúhelník kolmý. Směr normály proto lze vyjádřit za pomocí vektorového součinu následovně:

$$\vec{n} = (V_1 - V_0) \times (V_2 - V_0), \tag{5.1}$$

kde V_0, V_1, V_2 jsou vrcholy trojúhelníku. Také směr $\vec{n}$ je nutné převést do souřadnic scény. Protože je však normála na stěnu kolmá a při převodu se posouvá na opačnou stranu (viz vektor n na obr. 5.4), používá se zde oproti převodu vrcholů invertovaná matice. Shader dále zahrnuje implementaci kongruentního generátoru (pseudo)náhodných čísel, jenž je blíže popsán v sekci 5.4.

Zpracování a export výstupu

V závěrečné fázi programu je *device-only* výstupní textura překopírována do *host-only* textury k výstupu. Tento postup je zaveden primárně pro ulehčení možného budoucího rozšíření pro zobrazování v reálném čase. Textura ze zařízení tak totiž nikdy GPU neopustí, pouze je krátce zpřístupněna pro čtení na *přenosové úrovni*. Grafický procesor se tak zbytečně nezpožďuje přenosem na CPU a zase nazpět a teoreticky by mohl v real-time aplikaci již počítat další rámec.

Obrázek 5.5: Detailnější diagram exportní fáze programu.

Překopírovaná textura je pak hostovským programem už pouze vyexportována ve formátu `.hdr`, postupně se korektně dealokují všechny použité zdroje a program končí (viz obr. 5.5).

5.3 Reprezentace a načítání scény

Pro vykreslení scény jsou třeba dohromady tři soubory:

1. **Wavefront OBJ** (.obj) obsahující jednotlivé 3D objekty k vykreslení.
2. **Wavefront MTL** (.mtl) – sesterský soubor k OBJ. Běžně slouží k definici *PBR materiálů*, v rámci implementace je však využíván čistě jako způsob, jak přiřadit jednotlivým stěnám objektu identifikátor.
3. **JSON** (.json) ideálně vytvořený přiloženým skriptem `mat_parser.py`. Přiřazuje identifikátory materiálů z MTL souboru ke koeficientům zúčastněných médií.

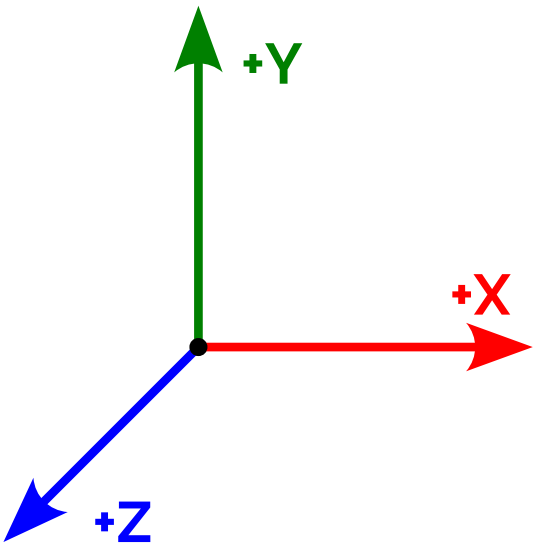
Soubory OBJ a MTL jsou zpracovány pomocí knihovny `tinyobj` a uloženy do tří bufferů. Vrcholy jsou načteny do `vertexBufferu`, indexy pro sestavení trojúhelníků do `indexBufferu` a indexy materiálů do `materialIdBufferu`. Čtvrtý a poslední načtený buffer `mediaDefinitionsBuffer` je vytvořen ze souboru JSON. V hlavičce bufferu je celkový počet definovaných médií a pak již následují jednotlivá data ve formátu dle tabulky 5.1.

Název	**Význam**
`id`	Číslo odkazující na materiál dle souboru MTL.
`sigma_s`	Rozptylový koeficient (3 čísla typu float za sebou).
`sigma_a`	Absorpční koeficient (3 čísla typu float za sebou).
`g`	Parametr asymetrie (3 čísla typu float za sebou).
`ior`	Index lomu média.

Tabulka 5.1: Paměťové rozložení bufferu `mediaDefinitionsBuffer`.

Souřadnicový systém scény

Scéna v rámci implementace přebírá stejný souřadnicový systém, který je definován pro soubory Wavefront OBJ, známý jako **pravoruký kartézský souřadnicový systém**. V tomto systému směřuje kladná osa x doprava, kladná osa y nahoru a kladná osa z směrem do kamery. Vizualizaci jednotlivých os je možno vidět na obrázku 5.6.



Obrázek 5.6: Pravoruký souřadnicový systém.

Formát JSON souboru pro definici scény a médií

Soubor JSON pro definici scény a médií je možné vytvořit přiloženým skriptem, nebo manuálně. Obecně nejvhodnější postup je poprvé soubor vytvořit pomocí skriptu a případně menší úpravy již dělat manuálně. Formát je dobře čitelný, a tak menší úpravy nejsou problém. Soubor následuje formát naznačený v tabulce 5.2

Název	Vnořený název	Význam
`scene`	`camera`	Pozice kamery v souřadnicích scény.
	`cameraLookAt`	Pozice bodu v souřadnicích scény, na který je kamera zaměřena.
	`fov`	Úhel záběru kamery.
	`lightPos`	Pozice světelného zdroje v souřadnicích scény.
	`lightColor`	Barva světelného zdroje ve složkách RGB.
	`lightIntensity`	Intenzita světelného zdroje.
	`scale`	Měřítko scény sloužící ke škálování koeficientů médií.
`{id}`	`sigma_s`	Rozptylový koeficient.
	`sigma_a`	Absorpční koeficient.
	`g`	Parametr asymetrie.
	`ior`	Index lomu média.

Tabulka 5.2: Formát souboru JSON definujícího scénu a zúčastněná média.

Sekce `scene` tento soubor uvozuje a obsahuje veškerou podporovanou konfiguraci scény jako takové. Soubor dále obsahuje libovolný počet sekcí označených identifikátorem daného materiálu dle souboru MTL. Jak lze vidět, formát definice média je velmi podobný bufferu z tabulky 5.1 a to proto, že se na něj přímo mapuje.

V tabulce, a tedy i souboru se nachází mj. také hodnota `scale`. Ta vychází ze skutečnosti, že koeficienty médií jsou definovány v jednotkách mm^{-1} (viz tabulka 4.1). Bez jakýchkoliv úprav je tedy scéna simulována tak, že jedna jednotka rozměru scény j je rovna jednomu milimetru v reálném světě. Jelikož trojrozměrné scény obvykle pracují s výrazně větším měřítkem, umožňuje program tyto hodnoty škálovat, a to právě pomocí nastavení `scale`. Hodnota tohoto parametru může být libovolné desetinné číslo, pro převod na reálné jednotky však platí následující vztahy:

- Pokud je `scale = 1.0`, platí vztah $1\ j = 1\ mm$.
- Pokud je `scale = 10.0`, platí vztah $1\ j = 1\ cm$.
- Pokud je `scale = 1000.0`, platí vztah $1\ j = 1\ m$.

Akcelerační struktury ve Vulkan API

Aby bylo možné využít rozšíření ray query a efektivně trasovat paprsky ve scéně, je nejprve nutné vytvořit z bufferů scény akcelerační struktury (dále zkráceně AS, viz sekce 2.7). Ty se při práci s rozhraním Vulkan používají hned dvě, obě typu BVH:

1. **Bottom-level AS** (zkr. BLAS, česky „spodní AS“) zaobaluje samotnou geometrii. Pro její tvorbu se využívá `vertexBuffer` a `indexBuffer`.

2. **Top-level AS** (zkr. TLAS, česky „svrchní AS“) představuje instanci BLAS. Rozhraní Vulkan umožňuje každé TLAS přidělit např. vlastní pozici, materiál a další vlastnosti. Právě TLAS je nahrávána na GPU a následně při vykreslování procházena.

Samotnou tvorbu AS obstarává Vulkan na pozadí – program mu pouze musí dát všechna potřebná data a zvolit nutná nastavení. Jedno z nejdůležitějších ovládá, jak bude výsledný BVH strom vypadat. Implementace používá nastavení `PREFER_FAST_TRACE`, které znamená, že není podstatné, jak dlouho se bude AS tvořit, a hlavní je co nejrychlejší průchod. Toto nastavení tak obecně vede na hlubší, ale lépe oddělené BVH stromy. Samozřejmě existuje také opačné nastavení `PREFER_FAST_BUILD` i řada dalších, které výslednou AS ovlivňují.

Deskriptory zdrojů

Každý zdroj (buffer, AS, textura apod.) potřebuje v rozhraní Vulkan svůj tzv. *deskriptor*. Jedná se o datovou strukturu zaobalující zdroje a zároveň specifikující jejich typ a jiná metadata. Každý typ zdroje má svůj deskriptor mírně odlišný.

Všechny deskriptory se po vytvoření ukládají pod unikátním identifikátorem do tzv. *setů*. Těch může být hned několik, nebo může být jeden – obecně se často dodržuje, že jeden set odpovídá jednomu vykreslovacímu průchodu. Pokud by například aplikace v jednom průchodu vykreslovala scénu rasterizací a ve druhém přidávala efekty s pomocí ray tracingu, může být vhodné potřebná data rozdělit do dvou setů. Vše však závisí na konkrétních potřebách dané aplikace. Tato implementace využívá pouze jeden set deskriptorů se všemi zdroji. Tak, jak jsou zdroje do setů vloženy, je možné je později interpretovat v shaderu.

Načítání shaderu a tvorba pipeline

Podobným procesem načítání musí projít i soubory s kódem pro zařízení, tedy shadery. Ze souboru je shader načten do tzv. *modulu*. Každý modul pak musí být přiřazen některé fázi procesu vykreslování. V klasickém případu rasterizace by se dalo mluvit o fázi vertex shaderu a fragment shaderu. V této implementaci se však využívá tzv. **compute shader** (česky „výpočetní shader“), a proto se k modulu přidává příznak `VK_SHADER_STAGE_COMPUTE_BIT`.

Shader se ještě před nahráním na GPU řadí do struktury `VkPipeline`. Ta popisuje celou pipeline zobrazování, v tradičním smyslu počítačové grafiky. Pro její vytvoření je třeba ještě `VkPipelineLayout`, ve které jsou specifikovány všechny sety deskriptorů a kam je možné přidat například tzv. *push konstanty* – Vulkan poskytuje velmi malý (garantováno je 128 bytů) buffer pro konstantní hodnoty, které jsou v shaderu potřeba, ale tvorba celého dedikovaného bufferu by pro ně byla zbytečná.

Pipeline se nakonec spolu se sety deskriptorů spojí s příkazovým bufferem a odešle na zařízení, jak již bylo popsáno v sekci 5.2.

Paměťové bariéry

Aby byl výkon co nejvyšší, Vulkan na pozadí používá vedle hlavní RAM paměti také výrazně rychlejší cache. To ale v praxi může znamenat, že např. výstup z GPU zůstane v cache paměti, kam se k němu CPU běžnými způsoby nedostane. Z toho důvodu zavádí Vulkan tzv. **paměťové bariéry**.

Paměťová bariéra ve zkratce zajišťuje, že se cache paměť obnovuje a že jsou data z jedné strany bariéry dostupné na straně druhé. Jedním příkladem z implementace je paměťová

bariéra na samém konci programu, kdy CPU čeká na výstupní texturu z GPU. Už bylo zmíněno, že výstupní textura je překopírována na přenosové úrovni (viz obr. 5.5). Proto je v programu implementována bariéra z `PIPELINE_STAGE_TRANSFER` na `PIPELINE_STAGE_HOST`, která dělá všechny zápisy na přenosové úrovni přístupné pro čtení na úrovni hosta.

5.4 Generování náhodných čísel

Generování náhodných čísel je pro Monte Carlo metodu klíčové, nicméně na GPU žádný generátor náhodných čísel běžně není. Proto je nutné nějaký naimplementovat. Pro implementaci byl zvolen algoritmus **PCG** (*permuted congruential generator*). Generátor pracuje se stavem typu `uint` (v GLSL 32bitové číslo) a za výchozí hodnotu – seed – je pro každý pixel dosazen výraz:

$$resolution.x \cdot pixel.y + pixel.x, \tag{5.2}$$

kde $pixel.x$ a $pixel.y$ jsou souřadnice právě vykreslovaného pixelu a $resolution.x$ je šířka výstupního renderu. V jádru PCG je kongruentní generátor, na výstupu se však dělá řada dalších operací. Implementace používá generátor `pcg_output_rxs_m_xs_32_32`[5], kde:

- **RXS** symbolizuje operaci *xorshift* o náhodný počet bitů (závislý na vstupu).
- **M** značí pronásobení konstantou.
- **XS** znázorňuje *xorshift* hodnoty sebou samotnou.
- **32** jsou bitové velikosti vstupu a výstupu.

Jeden krok kongruentního generátoru udává výraz:

$$state = state \cdot 747796405 + 1, \tag{5.3}$$

a celý proces generování náhodného čísla, včetně převodu na `float` hodnotu v intervalu $\langle 0, 1\rangle$, popisuje následující pseudokód. Veškeré konstanty jsou zde navrženy specificky tak, aby byla perioda generátoru co nejdelší.

Algoritmus 2: Generátor náhodných čísel PCG rxs_m_xs_32_32

Vstup: **uint** state
$state = step(state)$;
uint $word = ((state \gg ((state \gg 28) + 4)) \wedge state)$; `/* rxs */`
$word = word \cdot 277803737$; `/* m */`
$word = (word \gg 22) \wedge word$; `/* xs */`
return $word$ / 4294967295.0;

5.5 Path tracer

Implementace path traceru obohacuje původní kód ze sekce 2.6 o větev programu zpracovávající průchod médiem, Fresnelovy vztahy (viz rovnice 2.9) a ruskou ruletu (viz sekce 2.8). V následujícím pseudokódu je proto znázorněn jen hlavní cyklus.

[5]Kód převzat z: https://github.com/imneme/pcg-c/blob/master/include/pcg_variants.h.

Algoritmus 3: Hlavní cyklus path traceru s podporou zúčastněných médií

```
throughput = White, color = ...;
while j ≠ maxBounces do
    if ray.hit.medium = true then
        r = getFresnel(...);                          /* Fresnelovy vztahy */
        if random() < r then
            reflect();
        else
            refract();
        end
        traceMedium(...);                  /* průchod médiem viz sekce 5.6 */
    else
        traceLambertian(...);               /* difúzní odraz viz sekce 2.6 */
    end
    j += 1;
    if j > rrStart then
        q = min(max(throughput), 0.95);                  /* ruská ruleta */
        if random() > q then
            break;
        end
    end
end
```

Funkce $getFresnel(...)$ umožňuje v rámci implementace na základě přepínače zvolit, zda bude využita rychlejší Schlickova aproximace (viz rovnice 2.12), nebo se korektně spočítají oba koeficienty r_s a r_p (dle rovnic 2.9 a 2.10). V tomto případě funkce vrací průměrnou hodnotu dle vztahu 2.11, jelikož implmentace neuvažuje polarizované světlo.

Sekce 2.8 pojednávající o optimalizaci zmiňovala také subpixelový posun jako jednu z metod antialiasingu. Tato metoda byla do implementace taktéž zahrnuta a uplatňuje se při prvotním vyslání paprsku pro nastavení směru tohoto paprsku. Subpixelový posun lze popsat vztahem:

$$\vec{p} = \vec{p_0} + (\xi_1, \xi_2), \tag{5.4}$$

kde $\vec{p_0}$ je pozice pixelu v souřadnicích obrazovky a (ξ_1, ξ_2) je vektor ze dvou náhodných čísel z intervalu $\langle 0, 1\rangle$. Pro určení pozice tohoto bodu na zobrazovací ploše se poté používají vztahy:

$$\vec{s_x} = \frac{2\vec{p_x} - r_x}{r_y}, \tag{5.5}$$

$$\vec{s_y} = \frac{2\vec{p_y} - r_y}{r_y}, \tag{5.6}$$

kde r_x a r_y jsou složky rozlišení výstupní textury. Finální pozice pixelu je poté upravena dle zorného pole kamery a prvotní paprsek je vyslán směrem od kamery k pozici pixelu na zobrazovací ploše.

5.6 Průchod zúčastněného média

Aby mohl průchod zúčastněného média vůbec začít, musí nejdříve path tracer zjistit, že došlo k jeho zásahu. Tato informace se plní do již výše zmíněné struktury `HitInfo` za pomoci `materialIdBufferu` a `mediaDefinitionsBufferu` velmi jednoduše – `materialIdBuffer` obsahuje pro každý primitiv identifikátor jeho příslušeného materiálu. Tento identifikátor se pak dá propojit s definicí média v `mediaDefinitionsBufferu` (viz tabulka 5.1). Pokud žádná definice média na tento materiál neodkazuje, jsou daný materiál a daná interakce považovány za difúzní.

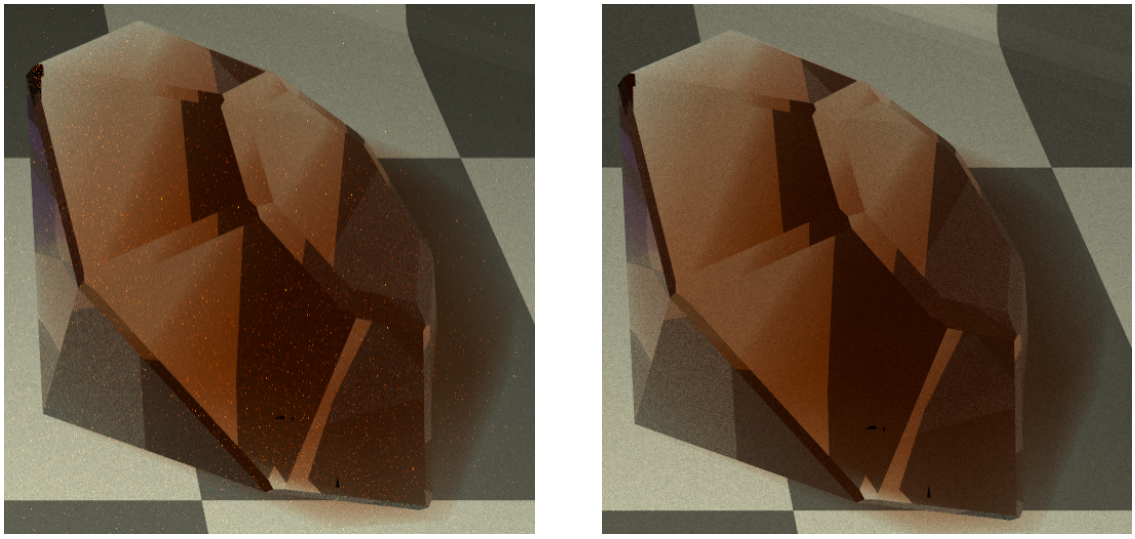
Ihned po zalomení do média se vzorkuje první vzdálenost interakce dle vztahu 3.16. Pro určení vzdálenosti ke konci média se používá nový jednorázový paprsek. Implementace v pseudokódu vypadá následovně:

Algoritmus 4: Vzorkování vzdálenosti interakce

```
Vstup: medium, distToEnd
vec3 σ_t = medium.σ_a + medium.σ_s;
float density = min(extinction);
float w = max(medium.σ_s / σ_t);                    /* albedo viz sekce 3.2 */
if w > 0 then
    w = max(w, 0.5);                          /* alespoň 50 % na interakci */
end
float ξ = random();
float sampledDist;
if ξ < w then
    ξ = ξ / w;
    sampledDist = −log(1 − ξ) / density;              /* viz rovnice 3.16 */
end
bool success = true;
if sampledDist > distToEnd then
    sampledDist = distToEnd;
    success = false;
end
medium.T_r = exp(σ_t · (−sampledDist));               /* viz rovnice 3.14 */
return success;
```

Pokud je vzorkovací pravděpodobnost daného média nenulová, a médium tedy nějaké světlo rozptyluje, je tato pravděpodobnost případně navýšena alespoň na 50 %. Tato úprava má za cíl snížení prostorového šumu ve výsledném snímku, který by mohl vznikat vlivem vysokého rozptylu jednotlivých vzorků.

Tento problém nastává zejména u médií, které sice světlo do nějaké míry rozptylují, ale rozptylují jej pouze velmi málo oproti své vysoké absorpci, což má za příčinu velmi nízkou hodnotu *scattering albeda*. Na obrázku 5.7 je možné vidět, že tento způsob kompenzace eliminuje nechtěný šum a zároveň nemá žádný negativní vliv na výsledný vykreslený snímek, ani jeho věrnost.



(a) Bez navýšení (b) S navýšením

Obrázek 5.7: Význam kompenzace navýšením pravděpobobnosti interakce.

Po získání vzdálenosti interakce se cesta posouvá do daného bodu a dochází k několika událostem:

1. Hodnota propustnosti $T_r(x \leftrightarrow x_t)$ ze vzorkování vzdálenosti je započítána do průběžné celkové propustnosti $T_r(x \leftrightarrow x_s)$.
2. Je vyslán jednorázový paprsek ke světelnému zdroji pro získání hodnoty L_i.
3. Je vyhodnocena fázová funkce pro importance sampling.
4. Do vzorků je připočítána barva:

$$color = T_r(x \leftrightarrow x_s) \cdot L_i \cdot phaseEval. \tag{5.7}$$

5. Z fázové funkce je navzorkován směr rozptylu.

Průchod média pak pokračuje až do doby, kdy se paprsek z média opět nevynoří ven.

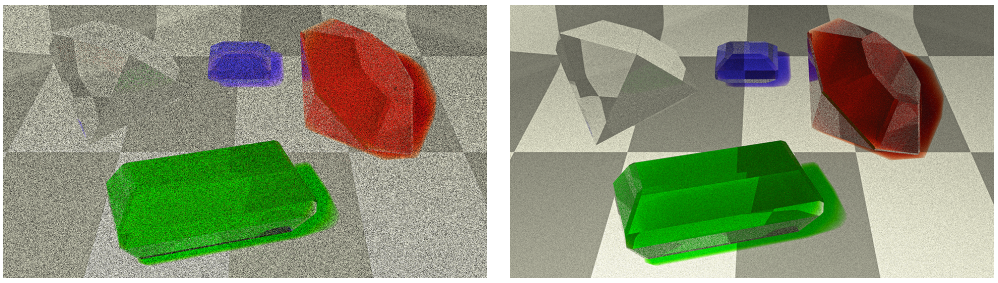
5.7 Dosažené výsledky

Implementovaná demonstrační aplikace obsahuje několik různých materiálů a scén zobrazujících různé jevy, které je možné u zúčastněných médií pozorovat. Jelikož program generuje snímky ve formátu HDR a barvy jsou tak v rozsahu mimo běžné spektrum RGB, byly všechny snímky dodatečně *tónovány* v programu Adobe Photoshop metodou lokálního přizpůsobení.

Všechny snímky byly vygenerovány v rozlišení Full HD. Uvedený čas vykreslení je čistý čas strávený výpočty na grafickém procesoru. Čas strávený zpracováním scény na CPU se pro všechny snímky pohybuje okolo půl vteřiny a není v uvedeném čase pro vykreslení zahrnut. Vykreslení proběhlo na stroji s grafickým procesorem Nvidia GeForce RTX 3060 Mobile.

Vliv počtu vzorků

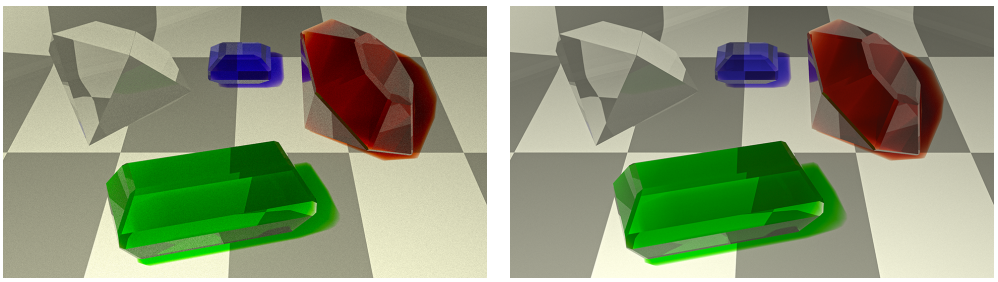
U první sady snímků je simulace provedena s různým počtem vzorků. Pro testování byla použita scéna se čtyřmi různými médii, a tedy spoustou rozmanitých interakcí – jde o média smaragd, rubín, safír a sklo. První dva snímky 5.8 mají velmi nízký počet snímků. Takové snímky jsou sice velmi rychle vygenerované, ale jen těžko použitelné.



(a) 1 vzorek, vykresleno za 14 ms (b) 8 vzorků, vykresleno za 61 ms

Obrázek 5.8: Snímky s velmi nízkým počtem vzorků.

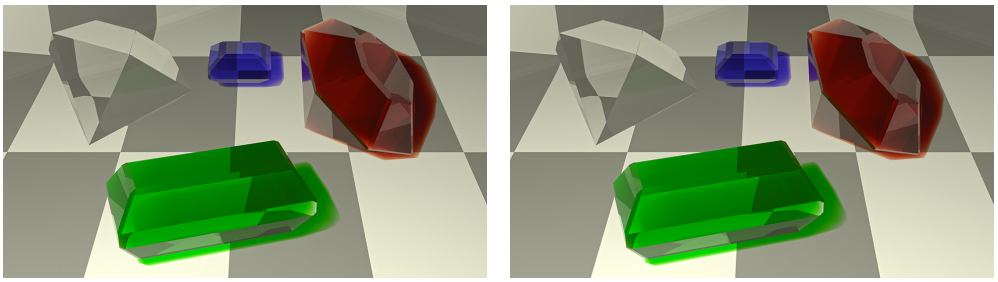
Následující snímky 5.9 mají 64 a 512 vzorků a jsou tak výrazně náročnější na vykreslení, jejich vizuální kvalita již však může být v některých případech dostačující.



(a) 64 vzorků, vykresleno za 445 ms (b) 512 vzorků, vykresleno za 3 619 ms

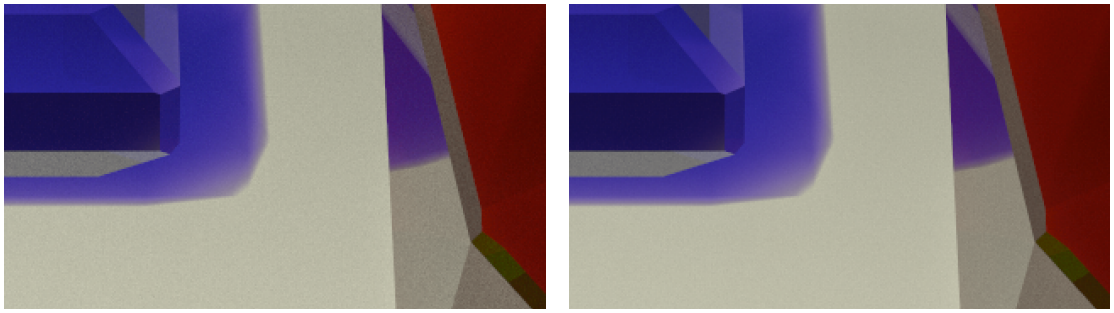
Obrázek 5.9: Snímky s dostatečným počtem vzorků.

Velmi kvalitních snímků lze nejčastěji dosáhnout s počtem vzorků 1 024 nebo 2 048. Oba snímky 5.10 jsou na první pohled velmi kvalitní, liší se až po přiblížení (viz obr. 5.11).



(a) 1 024 vzorků, vykresleno za 7 289 ms (b) 2 048 vzorků, vykresleno za 14 580 ms

Obrázek 5.10: Snímky s velmi vysokým počtem vzorků.

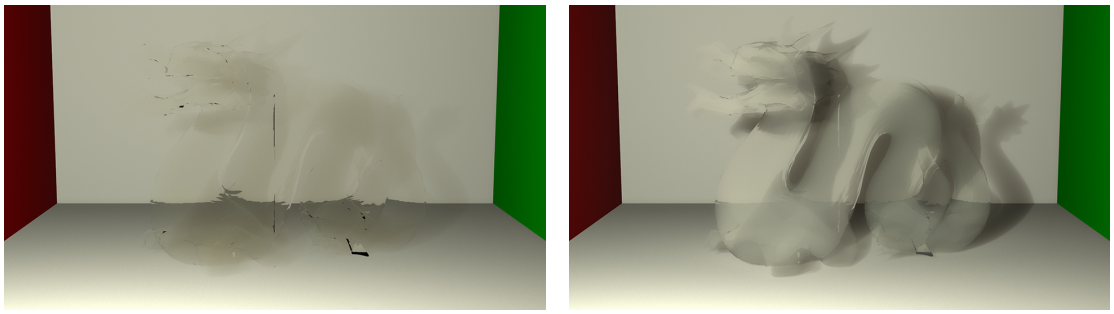


(a) 1 024 vzorků, přiblíženo (b) 2 048 vzorků, přiblíženo

Obrázek 5.11: Snímky s velmi vysokým počtem vzorků, 6× zoom.

Vliv rozptylového koeficientu

Další snímky simulují, jaký vliv má rozptylový koeficient na procházející světlo. Pro tyto ukázky bylo využito zúčastněné médium *mléko* se sníženým indexem lomu, aby byl efekt na průchozí světlo více zřejmý. První dva snímky 5.12 mají rozptylový koeficient dělený 1 000 [obr. (a)] a 100 [obr. (b)]. Lze pozorovat, že ve snímku s minimálním rozptylovým koeficientem prochází světlo takřka beze změny – jasně jde vidět pozadí a model je pouze mírně zbarven nízkým absorpčním koeficientem mléka. Na snímku (b) již lze pozorovat mírné zakalení pozadí.

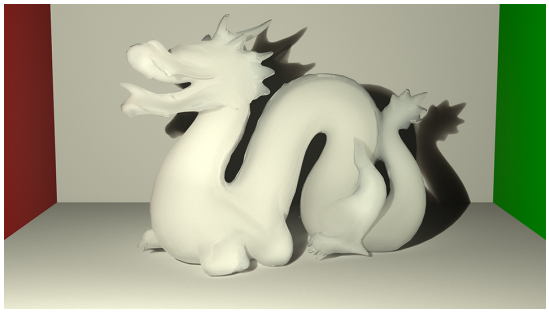


(a) Hodnota $\sigma_s = (0.0182, 0.0204, 0.0224)$
(2 048 vzorků, vykresleno za 15,5 s)

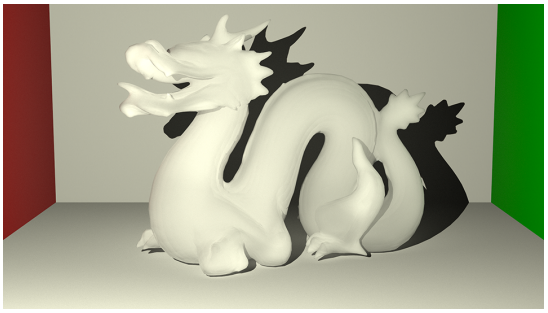
(b) Hodnota $\sigma_s = (0.1821, 0.2038, 0.2237)$
(2 048 vzorků, vykresleno za 23,3 s)

Obrázek 5.12: Dopad rozptylového koeficientu na vzhled média.

Následující dva snímky 5.13 už se příliš neliší. Obr. (a) má rozptylový koeficient dělen 10, obr. (b) už je plnohodnotné zúčastněné médium mléko. Rozdíl je patrný především u okrajů média – ty jsou v obr. (a) o něco průhlednější. Jak bylo již několikrát zmíněno, vyšší rozptylový koeficient přímo souvisí s vyšším počtem vzorkovaných interakcí – tato skutečnost se projevuje také na době potřebné k vykreslení snímku, která se, až na poslední dva snímky, které jsou velmi podobné, postupně zvyšuje.



(a) Hodnota $\sigma_s = (1.8205, 2.0383, 2.2370)$
(2 048 vzorků, vykresleno za 36,5 s)

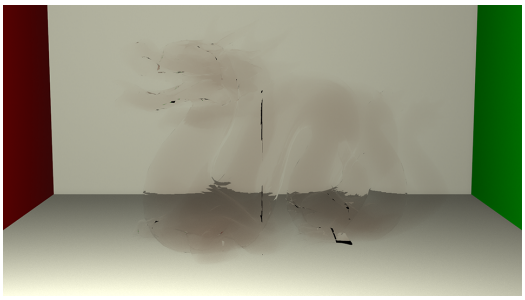


(b) Hodnota $\sigma_s = (18.2052, 20.3826, 22.3698)$
(2 048 vzorků, vykresleno za 38 s)

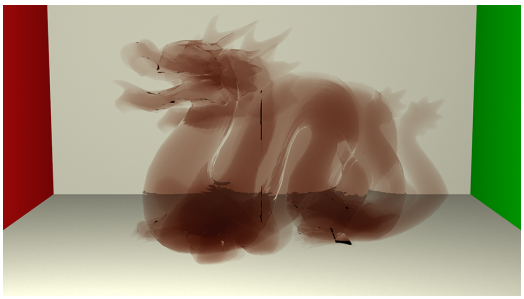
Obrázek 5.13: Dopad vyššího rozptylového koeficientu na vzhled média.

Vliv absorpčního koeficientu

Pro ukázku vlivu absorpčního koeficientu bylo vybráno médium *hroznový džus* a index lomu byl opět snížen pro lepší viditelnost vlivu absorpce. Jelikož jsou absorpční koeficienty obecně výrazně nižší, pro první snímek byl absorpční koeficient dělen 10, zatímco druhý snímek už je samotné médium. Výsledky, kdy je světlo zabarveno do vínova, jsou na obr. 5.14.



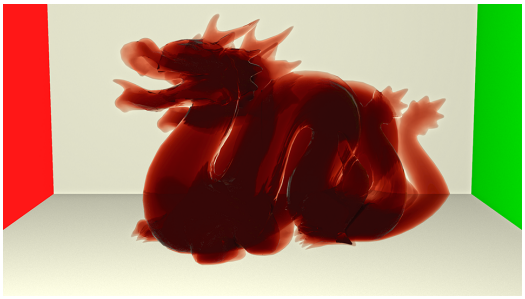
(a) Hodnota $\sigma_a = (0.0104, 0.0240, 0.0293)$
(2 048 vzorků, vykresleno za 14,5 s)



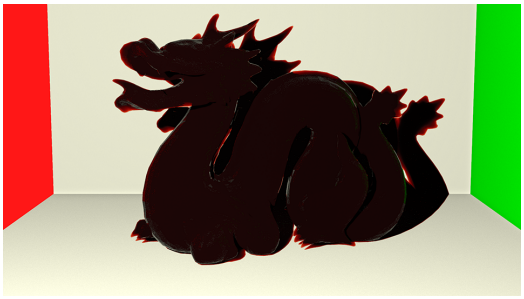
(b) Hodnota $\sigma_a = (0.1040, 0.2399, 0.2933)$
(2 048 vzorků, vykresleno za 16,7 s)

Obrázek 5.14: Dopad absorpčního koeficientu na vzhled média.

Efekt vysoké absorpce znázorňují snímky 5.15. Jak lze vidět, absorpce a rozptyl mají prakticky opačné efekty. Zatímco při rozptylu se paprsek v médiu odráží tolikrát, že se vytratí původní světelná informace a výsledek je takřka zcela bílý, vysoká absorpce veškeré světlo pohltí. Výsledný snímek je tak velmi tmavý, až téměř úplně černý.



(a) Hodnota $\sigma_a = (1.0404, 2.3958, 2.9325)$
(2 048 vzorků, vykresleno za 21,9 s)

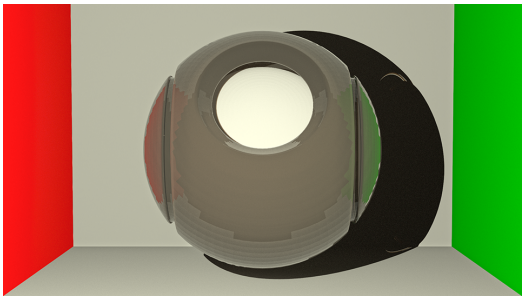


(b) Hodnota $\sigma_a = (10.4040, 23.9580, 29.3250)$
(2 048 vzorků, vykresleno za 24,3 s)

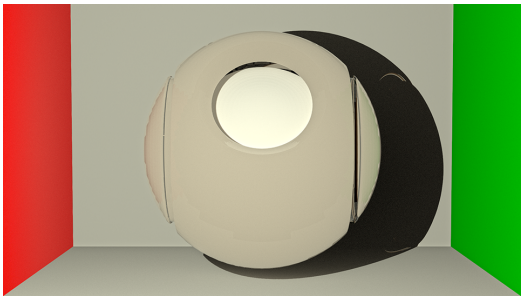
Obrázek 5.15: Dopad vyššího absorpčního koeficientu na vzhled média.

Vliv asymetrického parametru

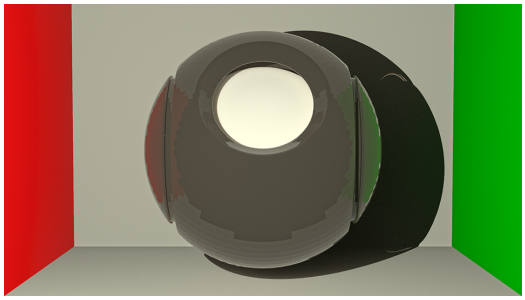
Pro zobrazení vlivu asymetrického parametru g se nejlépe hodí vysoce rozptylující média. Pro reprezentaci je z dostupných dat tak nejvíce vhodné mléko, nebo espresso. Espresso má výraznější barvu, a proto bylo využito pro demonstrační snímky 5.16.



(a) Hodnota $g = (0.907, 0.896, 0.880)$
(2 048 vzorků, vykresleno za 25,8 s)



(b) Hodnota $g = (0.0, 0.0, 0.0)$
(2 048 vzorků, vykresleno za 15,1 s)



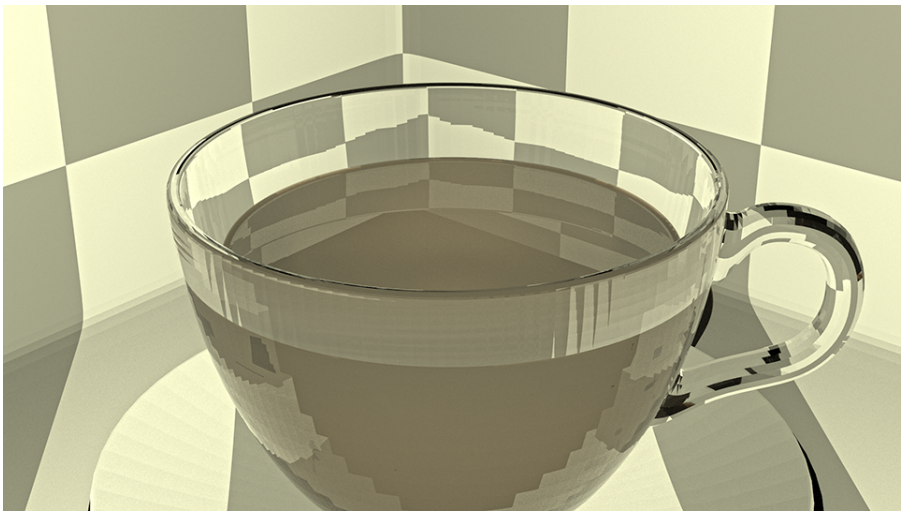
(c) Hodnota $g = (-0.907, -0.896, -0.880)$
(2 048 vzorků, vykresleno za 23,3 s)

Obrázek 5.16: Vliv asymetrie na vzhled média.

Obr. (a) zobrazuje dopředně asymetrické anizotropní zúčastněné médium, tedy fyzikálně věrné espresso. V případě izotropního média (obr. (b)) se světlo rozptyluje mnohem více rovnoměrně, což vede na obecně více prosvětlený objekt. Poslední, zpětně asymetrické anizotropní médium zvyšuje pravděpodobnost, že se paprsek již krátce po zalomení odrazí zpět a nestihne se tak tolik rozptýlit. Výsledná barva je tak o něco tmavší.

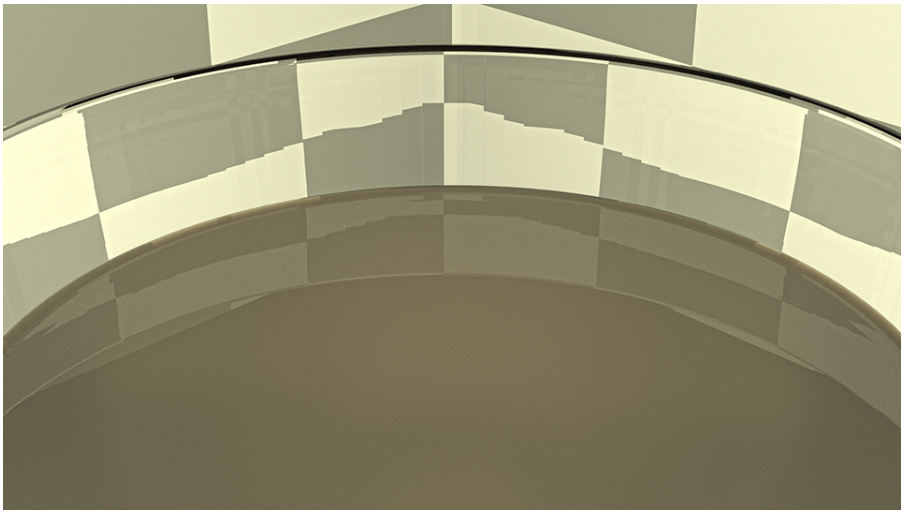
Rozhraní dvou médií

Snímek 5.17 vykresluje rozhraní dvou zúčastněných médií – fyzikálně věrného espressa v hrnku z fyzikálně nepřesného skla. Na snímku je možné pozorovat především korektní chování světla s ohledem na Frenselovy vztahy – na skleněném hrnku, zejména ve spodní části snímku, je jasně vidět, kde se světlo více zalamuje a kde se více odráží.



Obrázek 5.17: Snímek fyzikálně přesného zúčastněného média *espresso*. Pro ilustrační účely ve fyzikálně nepřesném „skleněném“ hrnku.
(1 024 vzorků, vykresleno za 14,5 s)

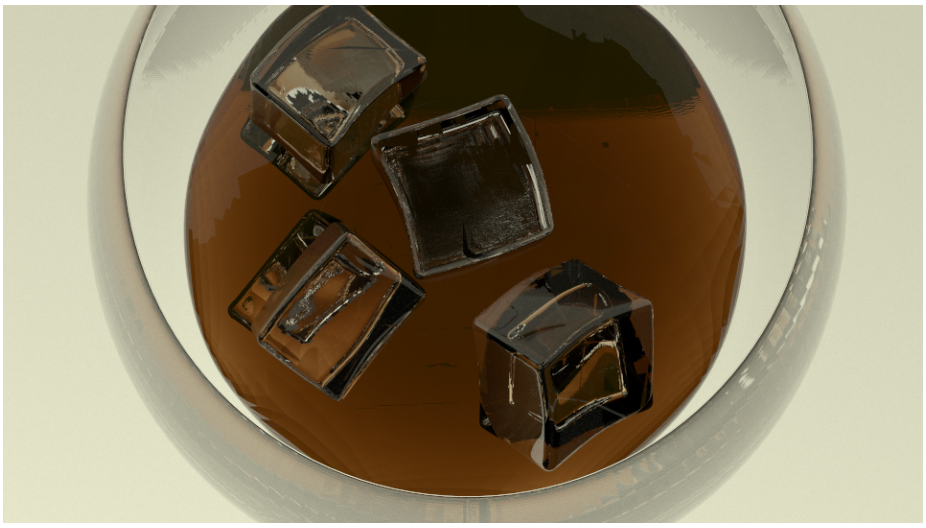
Pro vykreslení dalšího snímku 5.18 byla kamera přiblížena, aby se do snímku dostalo co nejvíce detailů. U samého rozhraní médií je možné vidět, že vykreslená káva místy prosvítá – jde o body, kde se světelný paprsek nestihl tolik rozptýlit nebo absorbovat, aby byla káva úplně neprůhledná. Byť je sklo jen uměle vytvořené a nemá podklady ve fyzikálních měřeních, velmi věrohodně působí také hrana hrnku.



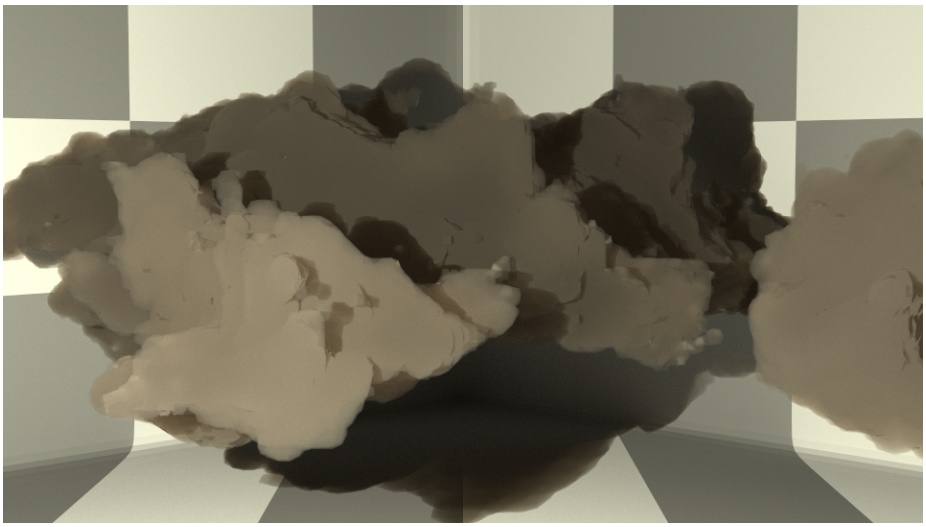
Obrázek 5.18: Detailní snímek zobrazující rozhraní médií zblízka.
(1 024 vzorků, vykresleno za 18,5 s)

Jak je možné vidět v popiscích snímků, vysoce detailní a věrně simulované snímky zvládá program vykreslit v rámci několika sekund, a to i s vysokým počtem vzorků. Implementaci podobné metody čistě na CPU by vykreslení zabralo minuty, spíše hodiny. Ukazuje se zde tak obrovská síla GPU akcelerace. Program má navíc spoustu prostoru pro optimalizace.

Dále v některých snímcích lze pozorvat, že program je náchylný na drobnou vizuální vadu v podobě občasných černých fragmentů. Ty jsou s největší pravděpodobností způsobeny nepřesností floating point čísel, která může vznikat kvůli zvolené metodě prevence sebeprůniku světelných paprsků. Tato vada však nemá zásadní vliv na korektnost simulace.



Obrázek 5.19: Médium Coca Cola s ledem (médium sklo).
(1 024 vzorků, vykresleno za 27,2 s)



Obrázek 5.20: Kouřový mrak.
(1 024 vzorků, vykresleno za 51,3 s)

Kapitola 6

Závěr

Cílem této práce bylo prozkoumat různé metody používané pro realistickou simulaci opticky složitých materiálů a některý z těchto přístupů prezentovat v demonstrační aplikaci. Pro aplikaci mělo být mj. diskutováno využití GPU a Vulkan API. Navržením a implementací metody sledování cest (path tracingu) s podporou zúčastněných médií, určenou pro GPU a rozhraní Vulkan, byly tyto cíle splněny.

Poznatky nabyté studiem technik realistického zobrazování jsou spolu s vysvětlením základních fyzikálních vlastností světla shrnuty v Kapitole 2. Kapitola dále zahrnuje popis nejpoužívanějších metod určených pro fotorealistické zobrazování v počítačové grafice a více do hloubky popisuje metodu sledování cest (path tracing). Právě tato metoda totiž byla zvolena pro finální implementaci. Kapitola 3 podrobněji prozkoumává vlastnosti jednoho typu komplexních materiálů – zúčastněných médií. Popisuje, jaké jevy je možné v těchto materiálech pozorovat, jak lze média klasifikovat a jak je nutné upravit tradiční zobrazovací metody z počítačové grafiky tak, aby média zahrnovaly.

Výsledný program následuje návrh specifikovaný v Kapitole 4. Část programu pracující na procesoru načítá uživatelem definovanou scénu spolu se souborem typu JSON obsahujícím všechny důležité koeficienty pro simulaci zúčastněných médií. Samotná simulace je pak implementována s využitím GPU a grafického rozhraní Vulkan API. O implementaci pojednává Kapitola 5, ve které jsou popsána úskalí programování v rozhraní Vulkan, zvolená metoda generování náhodných čísel pro *Monte Carlo path tracing* a v neposlední řadě také různé optimalizace, které řešení obsahuje.

Práce mi dala vcelku komplexní vhled do světa nejen počítačových simulací světelného přenosu, ale i graficky akcelerovaného programování obecně. Tato zkušenost je uplatnitelná napříč řadou různých odvětví informačních technologií, kde je výkonnost na prvním místě.

Jako první se z možných rozšíření nabízí vytvoření grafického rozhraní pro usnadnění práce s programem, ideálně obsahujícího náhled generovaného snímku s jednoduchým stínováním před spuštěním procesu vykreslování. Pro vylepšení samotné simulace zúčastněných médií by dalším logickým krokem byla implementace heterogenních médií s pomocí volumetrických dat (např. 3D textura, mřížka, ...) a postupů, které byly v této práci taktéž prozkoumány. Celkový vizuální dojem by zlepšila také implementace dalších funkcí BRDF, případně techniky mapování prostředí.

Literatura

[1] Bartell, F. O.; Dereniak, E. L. a Wolfe, W. L. The Theory And Measurement Of Bidirectional Reflectance Distribution Function (Brdf) And Bidirectional Transmittance Distribution Function (BTDF). In: SPIE. online. 1981, sv. 257, s. 154–160. ISBN 9780892522866. Dostupné z: https://doi.org/10.1117/12.959611.

[2] Blinn, J. F. Models of light reflection for computer synthesized pictures. New York, NY, USA: Association for Computing Machinery, jul 1977, sv. 11, č. 2, s. 192–198. ISSN 0097-8930.

[3] Born, M. a Wolf, E. *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light*. Elsevier, 2013.

[4] Bucholtz, A. Rayleigh-scattering calculations for the terrestrial atmosphere. *Applied optics*. Optica Publishing Group, 1995, sv. 34, č. 15, s. 2765–2773.

[5] Gambetta, G. *Computer Graphics from Scratch* online. 1. vyd. Gabriel Gambetta, 2021. ISBN 9781718500761. Dostupné z: https://www.gabrielgambetta.com/computer-graphics-from-scratch/. [cit. 2024-04-11].

[6] Gautron, P. Real-time ray-traced ambient occlusion of complex scenes using spatial hashing. In: *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks*. 2020, s. 1–2.

[7] Glassner, A. S. *An introduction to ray tracing*. Morgan Kaufmann, 1989.

[8] Group, T. K. V. W. *Vulkan® 1.3.283 - A Specification* online. 1. vyd. Beaverton, USA, duben 2024, revidováno 18. 4. 2024. Dostupné z: https://registry.khronos.org/vulkan/specs/1.3/html/vkspec.html. [cit. 2024-04-24].

[9] Hart, J. C. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*. Springer-Verlag Berlin/Heidelberg, 1996, sv. 12, č. 10, s. 527–545.

[10] Hecht, E. *Optics, 5e*. Pearson Education India, 2002.

[11] Henyey, L. G. a Greenstein, J. L. Diffuse radiation in the galaxy. *Astrophysical Journal, vol. 93, p. 70-83 (1941).*, 1941, sv. 93, s. 70–83.

[12] Immel, D. S.; Cohen, M. F. a Greenberg, D. P. A radiosity method for non-diffuse environments. New York, NY, USA: Association for Computing Machinery, aug 1986, sv. 20, č. 4, s. 133–142. ISSN 0097-8930. Dostupné z: https://doi.org/10.1145/15886.15901.

[13] Jarosz, W. *Efficient Monte Carlo Methods for Light Transport in Scattering Media.* La Jolla, CA, USA, 2008. Disertační práce. University of California, San Diego. ISBN 978-0-549-72071-3.

[14] Jensen, H. W. *Realistic image synthesis using photon mapping.* AK Peters/crc Press, 2001.

[15] Johnson, B. Lighting: Falloff. *TAPESTRY: The Art of Representation and Abstraction* online. University of Washington, duben 2014. Dostupné z: https://courses.washington.edu/arch481/1.Tapestry%20Reader/2.Light%20Sources/5.Falloff/0.default.html. [cit. 2024-27-03].

[16] Kajiya, J. T. The rendering equation. *SIGGRAPH Comput. Graph.* New York, NY, USA: Association for Computing Machinery, aug 1986, sv. 20, č. 4, s. 143–150. ISSN 0097-8930. Dostupné z: https://doi.org/10.1145/15886.15902.

[17] Kay, T. L. a Kajiya, J. T. Ray tracing complex scenes. *ACM SIGGRAPH computer graphics.* ACM New York, NY, USA, 1986, sv. 20, č. 4, s. 269–278.

[18] Kwan, A.; Dudley, J. a Lantz, E. Who really discovered Snell's law? *Physics World.* IOP Publishing Ltd, apr 2002, sv. 15, č. 4, s. 64. Dostupné z: https://dx.doi.org/10.1088/2058-7058/15/4/44.

[19] MDN, P. 3D collision detection. *MDN Web Docs* online. 3. listopadu 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection. [cit. 2024-04-14].

[20] Möller, T. a Trumbore, B. Fast, minimum storage ray/triangle intersection. In: *ACM SIGGRAPH 2005 Courses.* 2005, s. 7–es.

[21] Narasimhan, S. G.; Gupta, M.; Donner, C.; Ramamoorthi, R.; Nayar, S. K. et al. Acquiring scattering properties of participating media by dilution. In: *ACM SIGGRAPH 2006 Papers.* 2006, s. 1003–1012.

[22] Novák, J.; Georgiev, I.; Hanika, J. a Jarosz, W. Monte Carlo Methods for Volumetric Light Transport Simulation. *Computer Graphics Forum*, 2018, sv. 37, č. 2, s. 551–576. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13383.

[23] Pharr, M.; Jakob, W. a Humphreys, G. *Physically based rendering: From theory to implementation: Third edition.* 3. vyd. Morgan Kaufmann, 2016. ISBN 978-0128006450.

[24] Pharr, M.; Jakob, W. a Humphreys, G. Lambertian Reflection. *Physically Based Rendering: From Theory To Implementation* online. FIT VUT v Brně, listopad 2008. Dostupné z: https://www.pbr-book.org/3ed-2018/Reflection_Models/Lambertian_Reflection. [cit. 2024-03-27].

[25] Phong, B. T. Illumination for computer generated pictures. *Commun. ACM.* New York, NY, USA: Association for Computing Machinery, jun 1975, sv. 18, č. 6,

s. 311–317. ISSN 0001-0782. Dostupné z: https://doi-org.ezproxy.lib.vutbr.cz/10.1145/360825.360839.

[26] Rubin, S. M. a Whitted, T. A 3-dimensional representation for fast rendering of complex scenes. In: *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. 1980, s. 110–116.

[27] Schlick, C. An inexpensive BRDF model for physically-based rendering. In: Wiley Online Library. *Computer graphics forum*. 1994, sv. 13, č. 3, s. 233–246.

[28] Stratton, J. A. *Electromagnetic theory*. Hoboken: John Wiley and Sons, 2007. IEEE Press series on electromagnetic wave theory. ISBN 0-470-13153-5.

[29] Tuy, H. K. a Tuy, L. T. Direct 2-D display of 3-D objects. *IEEE Computer Graphics and Applications*. IEEE, 1984, sv. 4, č. 10, s. 29–34.

[30] Vyšín, I. a Říha, J. *Paprsková a vlnová optika*. 1. vyd. Univerzita Palackého v Olomouci, 2012. ISBN 978-80-244-3334-9.

[31] Wikipedie. *Intenzita osvětlení — Wikipedie: Otevřená encyklopedie*. 2023. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Intenzita_osv%C4%9Btlen%C3%AD&oldid=23355989. [Online; navštíveno 11. 04. 2024].

[32] Yang, X. a Ouyang, Y. Real-time ray traced caustics. *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Springer, 2021, s. 469–497.

Příloha A

Obsah přiloženého CD

Přiložené CD následuje tuto adresářovou strukturu:

- `complex_materials_renderer/`
  - `source/` – zdrojové kódy v jazyce C++ pro hostitelský program
  - `source/shaders` – zdrojové kódy v jazyce GLSL pro program zařízení (implementace využívá `volpath.comp.glsl`, zbylé jsou pro účely debugování)
  - `resources/scenes/` – několik scén, předpřipravených pro okamžité použití
  - `resources/scenes/mat_parser.py` – skript pro přípravu vlastních scén
  - `README.md` – postup instalace závislostí a kompilace
  - `CMakeLists.txt` – nastavení pro sestavení projektu pomocí CMake
- `bin_x64/Release`
  - `complex_materials_renderer_source.exe` – zkompilovaný binární soubor
  - do tohoto adresáře se také ukládají rendery