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

PHYSICALLY BASED RENDERING

PHYSICALLY BASED RENDERING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JIŘÍ HERRGOTT**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ STARKA**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Herrgott Jiří**
Program: Informační technologie
Název: **Physically Based Rendering**
Physically Based Rendering
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte techniky Physically Based Renderingu (PBR).
2. Navrhněte knihovnu realizující PBR.
3. Vytvořte demonstrační aplikaci používající danou knihovnu.
4. Vytvořte krátké video demonstrující práci.

Literatura:

- Po dohodě s vedoucím.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2 a část bodu 3.

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

Vedoucí práce: **Starka Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato bakalářská práce se věnuje vykreslování 3D modelů v reálném čase, kde je kladen důraz na fyzikální vlastnosti simulovaných světelných paprsků. Práce pojednává o technikách, které se touto problematikou zabývají a následně jejich využití pro vykreslení fyzikálně aproximovaného modelu.

Abstract

This bachelor thesis is focused on rendering of 3D models in real time, where emphasis is placed on physical properties of simulated light rays. The work is about techniques that deal with this issue and then their use for drawing a physically approximated model.

Klíčová slova

PBR, PBS, fyzikálně založený model, mikrofasety, BRDF, IBL, GGX, Trowbridge-Reitz, Vulkan, glTF

Keywords

PBR, PBS, physically-based model, mikrofacet, BRDF, IBL, GGX, Trowbridge-Reitz, Vulkan, glTF

Citace

HERRGOTT, Jiří. *Physically Based Rendering*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka

Physically Based Rendering

Prohlášení

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

.....
Jiří Herrgott
9. května 2021

Obsah

1 Úvod **3**

2 Techniky PBS **4**
2.1 Základní vzorec odrazu 4
2.2 BRDF 5
2.3 Povrchová odrazivost 6
2.3.1 Fresnel 7
2.3.2 Distribuční funkce 8
2.3.3 Geometrická funkce 8
2.4 Podpovrchová odrazivost 9
2.5 Model osvětlení 10
2.5.1 Přesné zdroje světla 10
2.5.2 Ambientní osvětlení 11
2.5.3 Image-Based Lighting 11
2.6 Skybox 12
2.7 Povrchové nerovnosti 12
2.7.1 Normal mapping 12
2.7.2 Parallax mapping 13

3 Návrh **14**
3.1 BRDF 14
3.2 Osvětlovací model 15
3.3 Knihovna 16
3.3.1 Načítaní modelů 16
3.4 Vizualizační aplikace 18
3.4.1 GUI 18

4 Implementace **20**
4.1 Využité technologie 20
4.2 BRDF 21
4.3 Osvětlovací model 23
4.4 Normal mapping 24
4.5 Knihovna 25
4.5.1 Zapouzdření Vulkan struktur 25
4.5.2 Model 27
4.5.3 Vykreslení modelů 28
4.6 Skybox 29
4.7 Vizualizační aplikace 29

5 Možná rozšíření **31**
5.1 Vylepšení knihovny 31
5.2 Optimalizace IBL 31
5.3 Pokročilejší vykreslení povrchových nerovností 32

6 Závěr **33**

Literatura **35**

A Ovládání aplikace **37**

Kapitola 1

Úvod

Bakalářská práce se zabývá vykreslováním texturovaných objektů, které využívají fyzikálních modelů pro co nejpřesnější simulování světelných paprsků ze skutečného světa. Jinak také nazývané PBR (Physically Based Rendering). K tomuto vykreslování se používá řada technik PBS (Physically Based Shading). Jedná se o sadu technik, které mají za úkol co nejefektivněji aproximovat fyzikální model pro vykreslování fotorealistických snímků.

Cílem této práce je vytvoření programu pro vykreslení daného objektu s definovanými fyzikálními vlastnostmi v reálném času. Bude taktéž možné v reálném čase tyto vlastnosti měnit společně i např. s osvětlením tak, aby se mohly pozorovat změny vlivu osvětlení na tento texturovaný objekt.

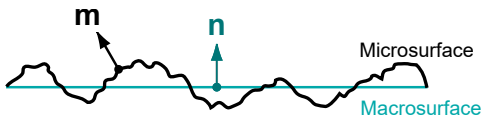
Počátky PBR technik se datují již v 60. až 80. létech 20. století, avšak rozšířené využití našli až v posledních 15 letech, kdy došlo ke skokovému nárustu výkonu hardwaru a taktéž rozšíření tohoto hardwaru v podobě stolních počítačů a notebooků běžnými uživateli. Tato technologie přešla od vědeckých simulací osvětlení přes profesionální animace filmových studií až po běžné každodenní využití, například v počítačových hrách.

V první kapitole se budu zabývat technikami PBS. Nejprve představím základní vzorec odrazu, poté vysvětlím princip funkce BRDF. Dále představím některé nejpopulárnější modely a následně je rozdělím dle typu. Poté se budu věnovat jednotlivým částem funkce BRDF dopodrobna. Na konci této kapitoly vypíši osvětlovací modely především následně využité v této práci. V druhé kapitole se zaměřím na návrh knihovny a programu dle zadaných požadavků. Jako první navrhnu jednotlivé části funkce BRDF a osvětlovací model z technik představených v kapitole 2. U knihovny ukáži, jakým způsobem si představuji její rozhraní a definuji, jak má napomoci pro účely této práce. Následně u programu zvolím způsob, kterým bude s touto knihovnou komunikovat a také navrhnu grafické rozhraní. Ve třetí kapitole se budu snažit popsat implementaci, kterou jsem provedl dle předchozí kapitoly 3. Nejprve se budu věnovat funkci BRDF a osvětlovacímu modelu, které lze považovat za jádro mé práce. Podrobně popíši, jakým způsobem jsem implementoval funkcionality knihovny a jak jsem rozdělil tuto problematiku do jednotlivých tříd. Na konci této kapitoly popíši implementaci vizualizační aplikace, která využívá třídy z knihovny pro plnění svého účelu. V poslední čtvrté kapitole podotknu věci, které by se daly vylepšit, dodělat nebo opravit.

Kapitola 2

Techniky PBS

Existuje mnoho druhů technik PBS, které se zabývají fyzikálnímu modely na úrovní mikro fasetů. Mikro fasety jsou miniaturní části povrchu, které jsou ploché a každá z nich disponuje normálou. A právě pro tyto mikro fasety lze definovat, jak se od povrchu každého z nich má rozptýlit světlo při kontaktu s povrchem tohoto mikro fasetu. Na následujícím obrázku 2.1 můžeme vidět rozdíl mezi mikro-povrchem a makro-povrchem. Za mikro fasetu se považuje část mikro-povrchu s normálou m.



Obrázek 2.1: Obrázek znázorňující rozdíl mezi makro-povrchem a mikro-povrchem s vyznačenými normálami n pro makro-povrch a m pro část mikro-povrchu [14].

V podstatě se především používá BSDF (Bidirectional Scattering Distribution Function), která má za úkol popsat, jakým způsobem se světlo rozptýlí od povrchu. Tato funkce se může rozdělit na dvě podfunkce, které po sečtení jsou ekvivalentní se zmíněnou BSDF. Jedná se o BRDF (Bidirectional Reflactance Distribution Function) a BTDF (Bidirectional Transmission Distribution Function). První z těchto funkcí řeší odraz světla od povrchu a druhá řeší šíření světla povrchem. Jak jsem se již zmínil, dohromady dávají BSDF. V první řadě je potřeba se zmínit o *reflectance equation* neboli „vzorci odrazu“.

2.1 Základní vzorec odrazu

To jest vzorec definující množtsví světla odraženého od povrchu. Vzorec je následující [9]:

$$L_o(v) = \int_\Omega f(l,v) \otimes L_i(l)(n * l) d\omega_i \tag{2.1}$$

Symbol $\otimes$ značí matematickou operaci pronásobení jednotlivých složek RGB vektorů.

V podstatě je množství světla obsaženého ve směru vektoru v rovno integrálu příchozího světla vynásobeného s funkcí BRDF a s kosinovým faktorem v hemisféře Ω[1], definované dle normály n, kde funkce $f(l,v)$ je funkcí pro výpočet podílu odraženého světla na úrovni mikrofasetů (neboli obvykle funkce BRDF). Dále $L_i(l)$ je množství příchozího světla ve

[1] Dalo by se říci, že jde o všechny možné směry nad povrchem

směru vektoru l vynásobeného skalárním součinem dvou vektorů n a l, které reprezentují již zmíněný kosinový faktor [9].

Nutno dodat, že funkce $f(l, v)$ by měla mít fyzikálně přijatelné vlastnosti [9]:

- vzájemnost – aneb výsledek funkce f je stejný při přehození vektorů

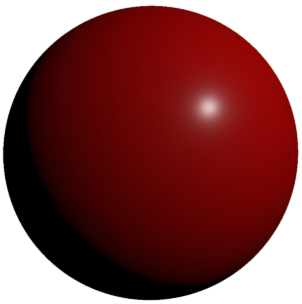
$$f(l, v) = f(v, l) \tag{2.2}$$

- zachování energie – výsledné množství odraženého světla nemůže být větší než množství světla směrujícího k povrchu

$$\forall l, \int_{\Omega} f(l, v)(n * v) d\omega_o \leq 1 \tag{2.3}$$

2.2 BRDF

Pro vykreslení odrazu světla se často používá model využívající funkce BRDF, neboli *Dvousměrová distribuční funkce odrazu*. Je to zjednodušený model, který celkem dobře aproximuje fyzikální vlastnosti odrazu světla od povrchu s poměrně rozumnými nároky na výpočetní výkon (viz obrázek 2.2). Jeho míra aproximace se může výrazně lišit. Některé modely dávají přednost výpočetnímu výkonu a některé se zaměřují na kvalitní aproximaci fyzikálních vlastností.



Obrázek 2.2: Obrázek znázorňující vykreslený BRDF model koule s jedním lokálním zdrojem světla[3].

Existuje mnoho modelů s různými vlastnostmi jako například:

- Phong
- Blinn-Phong
- Torrance-Sparrow
- Cook-Torrance
- GGX (Trowbridge-Reitz)
- Disney
- ...

Modely se dají dělit na fyzikální a empirické. Třeba v případě Phong [11] nebo Blinn-Phong [1] se jedná o empirické modely a takovým modelům se ve své práci věnovat nebudu. Zaměřím se především na fyzikální modely, které mají z hlediska této práce smysl.

[3]Obrázek byl vygenerován s pomocí webové aplikace dostupné z: https://oneshader.net/

2.3 Povrchová odrazivost

Ve své práci se především zaměřím na Cook-Torrance model, který si dal za úkol, co nejpřesněji aproximovat fyzikální model za rozumnou cenu hardware výkonu a to na úrovni mikro fasetů. Tímto modelem [4] je definovaná BRDF funkce následovně:

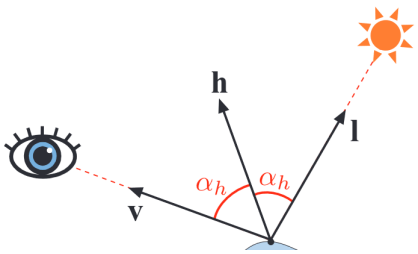
$$R_s = \frac{FDG}{\pi(n * l)(n * v)} \tag{2.4}$$

kde:

- R_s je výsledný dvousměrový zrcadlový odraz
- F je odraz hladkého povrchu
- D je distribuční funkce sklonu fasety
- G je faktor geometrického útlumu
- n je normálový vektor k povrchu objektu
- l je vektor ke světelnému zdroji
- v je vektor ke kameře, neboli k „oku“ uživatele

Pro budoucí použití je potřeba si zadefinovat tzv. half-angle vektor v budoucích vzorcích znám jako h. To jest vektor mezi světelným paprskem l a vektorem ke kameře v, znázorněný na obrázku 2.3. Je definován jako:

$$h = \frac{l + v}{|l + v|} \tag{2.5}$$



Obrázek 2.3: Obrázek vyobrazující half-angle vektor h vzniklý z vektoru světelného paprsku l a vektoru směřujícímu ke kameře v [10].

V tomto vzorci 2.4 se často nahrazuje π číslem 4. Toto číslo je tam kvůli transformaci z half-angle prostoru do jiných prostorů.

Vzorec 2.4 lze rozdělit na tři samostatné části [8]. První část také známou jako funkci viditelnosti, která působí jako maska, která určuje, jaké části povrchu budou viditelné pro kameru a do jaké míry. Vzorec je následující:

$$V(l, v, h) = \frac{G}{(n * l)(n * v)} \tag{2.6}$$

Druhou část doprovází určité změny, a to konkrétně π v čitateli vzorce. Dle [8] by se mělo násobit π v případě, když jsou využita bodová světla pro osvětlení modelu. Tato část určuje

vlastnosti odlesku, například jeho tvar nebo velikost. Více o distribuční funkci se lze dočíst v kapitole 2.3.2. Po úpravách se druhá část počítá jako:

$$D_{pl} = \frac{\pi}{4} D \tag{2.7}$$

A třetí částí je zbývající Fresnelova funkce odrazu. Tato funkce určuje, jaké množství světla je odraženo od plochy materiálu. Více se lze dočíst v následující kapitole 2.3.1.

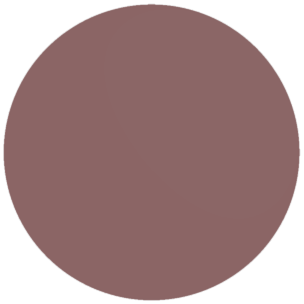
2.3.1 Fresnel

Funkce sloužící pro výpočet podílu odraženého světla od povrchu ku lomenému. Závisí na úhlu mezi normálou a světelným paprskem a na indexu lomu daným materiálem. V BRDF výpočtu mikro fasetů se místo úhlu mezi normálou a světelným paprskem, využívá uhlu mezi světelným paprskem a half-angle vektorem h. Index lomu v podstatě popisuje, o jaký druh materiálu se jedná. Na základě této vlastnosti se určuje množství odraženého světla od povrchu materiálu. Obecně jsou materiály děleny na metalické a nemetalické, přičemž speciální, většinou polovodičové materiály, už v praxi nejsou tolik řešeny kvůli jejich malému zastoupení. U metalických materiálů se tento parametr většinou pohybuje mezi 50% až 98%. Nemetalické materiály se pohybují v řádu jednotek procent. V praxi se využívá Schlickovi aproximace Fresnelovy funkce [12] pro snížení nároků na výpočetní výkon. Ukázku výstupu této funkce pro model koule s 5% metalicitou, je možné vidět na obrázku 2.4. Vzorec po úpravě pro mikro BRDF vypadá následovně:

$$F(F_0, l, h) = F_0 + (1 - F_0)(1 - (l * h))^5 \tag{2.8}$$

Pro ještě lepší optimalizaci lze využít *Sférické Gaussovské aproximace* pro nahrazení exponenciály [7]. Vzorec pak vypadá následovně:

$$F(F_0, v, h) = F_0 + (1 - F_0)2^{(-5.55473(v*h) - 6.98316)(v*h)} \tag{2.9}$$



Obrázek 2.4: Ukázka Fresnelovy funkce BRDF, vycházející z modelu z obrázku 2.2 s hodnotou metalicity 5%[5].

[5]Obrázek byl vygenerován s pomocí webové aplikace dostupné z: https://oneshader.net/

2.3.2 Distribuční funkce

Distribuční funkce definuje podíl fasetů, orientovaných ve směru half-angle vektoru h [4]. V podstatě by se dalo říci, že se jedná o funkci, která nám popisuje vlastnosti odlesku světla od povrchu. Konkrétně definuje tvar, velikost nebo jas odrazu. Různé modely využívají různé funkce. Obvykle funkce distribuční a geometrická bývají k sobě vázány tak, aby jejich vlastnosti po vynásobení ve vzorci 2.4 odpovídaly co nejvíce požadovanému fyzikálnímu modelu. Obvykle se jako parametr využívá vlastnosti povrchu nazvané *roughness*, neboli drsnost r. Na rozdíl od Fresnelovy funkce, nebo geometrické funkce, nemusí být výsledná hodnota mezi 0 a 1, ale nemůže být záporná. Příklad výsledku distribuční funkce lze spatřit na obrázku 2.5. Mezi nejvíce využívané se považují:

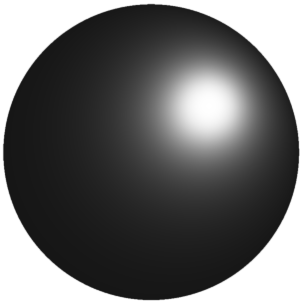
Beckmannova distribuční funkce [10]:

$$D(h) = \frac{\underline{(n * h)}}{\pi r^2 (n * h)^4} e^{\frac{(n*h)^2 - 1}{r^2 (n*h)^2}} \tag{2.10}$$

GGX (Trowbridge-Reitz) distribuční funkce [10]:

$$D(h) = \frac{\underline{(n * h)} r^2}{\pi (1 + (n * h)^2 (r^2 - 1))^2} \tag{2.11}$$

Podtržené $\underline{(n * h)}$ znamená, že výraz musí mít nezápornou hodnotu.



Obrázek 2.5: Ukázka distribuční funkce BRDF, vycházející z modelu z obrázku 2.2 s drsností povrchu 25%[7].

2.3.3 Geometrická funkce

Geometrická funkce je v podstatě maska tvořená pravděpodobností, že bod s normálou n bude dosažitelný, jak pro světelný paprsek l, tak pro vektor ke kameře v. Příkladem této masky pro model koule je obrázek 2.6. V BRDF se opět někde zaměňuje normála n za half-angle vektor h. Protože se jedná o masku, hodnoty se musí pohybovat mezi 0 a 1.

Různé modely využívají opět různé geometrické funkce. V případě Cook-Torrance modelu [4] se užívá funkce:

$$G(l, v, h) = min\left\{1, \frac{2(n * h)(n * v)}{(v * h)}, \frac{2(n * h)(n * l)}{(v * h)}\right\} \tag{2.12}$$

Zmíněná funkce využívá i normálu n i half-angle vektor h. Ze vzorce lze vypozorovat, že může docházet k vykrácení jmenovatele vzorce 2.4 touto funkcí. V praxi se ale často používá

[7] Obrázek byl vygenerován s pomocí webové aplikace dostupné z: https://oneshader.net/

Smithova aproximace, která dělí dvousměrovou masku na dvě masky jednosměrové. Vzorec pak vypadá následovně:

$$G(l, v) = G_1(l, n)G_1(v, n) \tag{2.13}$$

Funkce G_1 je funkce derivovaná z distribuční funkce D. Opět různé modely využívají různé funkce. Zpravidla ale tato funkce musí být s distribuční funkcí vázána vlastnostmi. Součástí této funkce je taktéž parametr drsnosti povrchu r. Například následující funkce odvozená od původního vzorce 2.12 [12].

$$G_1(v) = \frac{v * n}{(1 - k)(v * n) + k} \tag{2.14}$$

V této funkci můžeme spatřit parametr k, který je definován modelem a je založen na drsnosti povrchu. Z již zmíněné [12] parametr k odpovídá následovně:

$$k = \sqrt{\frac{2r^2}{\pi}} \tag{2.15}$$

Pro některé modely, například GGX (Trowbridge-Reitz) [14], se používají jiné parametry k podle potřeby. V [7] se lze setkat se dvěma vzorci, přičemž první $k = \frac{(r+1)^2}{8}$ se používá pro přímé osvětlovací modely a druhý $k = \frac{r^2}{2}$ se používá pro IBL (Image-Based Lightning), které jsou podrobně popsané v kapitole 2.5.3.

Obrázek 2.6: Ukázka geometrické funkce BRDF, vycházející z modelu z obrázku 2.2 s drsností povrchu 25%[9].

2.4 Podpovrchová odrazivost

Ve většině modelů se využívá tzv. Lambertovy diffusní složky [7]. Jedná se o konstantní hodnotu vzniklou ze vzorce:

$$f_{lambert} = \frac{c}{\pi} \tag{2.16}$$

kde c je většinou diffusní hodnota. Tato diffusní hodnota je právě podělena π, kvůli tomu, aby tato hodnota byla normalizována vzhledem ke světlu, protože BRDF ze vzorce 2.4 je násobena π.

[9]Obrázek byl vygenerován s pomocí webové aplikace dostupné z: https://oneshader.net/

2.5 Model osvětlení

Obvykle pro PBS model nestačí pouze využití matematického modelu BRDF, ale využívá se i osvětlovacího modelu, přičemž se obvykle tyto dva modely kombinují. Některé osvětlovací modely dokáží být extrémně náročné na výpočetní výkon, a to až do úrovně, kdy jeden snímek trvá vykreslit v řádů minut v závislosti na množství objektů, atd. Pro účely své práce jsem zvažoval modely, které by měly být přiměřeně náročné pro relativně plynulé vykreslování v reálném čase.

2.5.1 Přesné zdroje světla

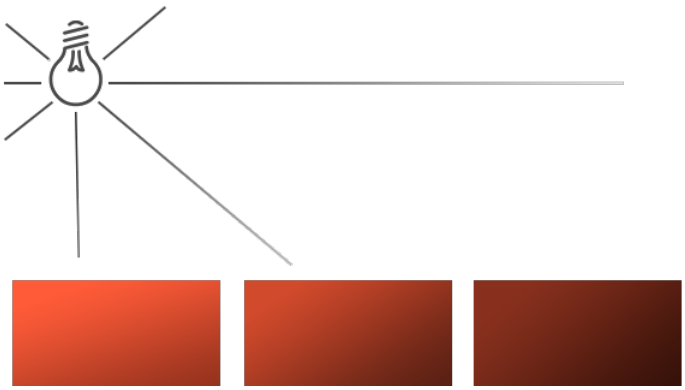
Jedná se o empyrické zdroje světla, které jsou nekonečně malé a nekonečně zářivé [9]. Většinou se uvažují modely směrového, bodového, nebo dokonce kuželovitého typu. Každý zdroj světla má vektorem definovanou barvu vyzařovaného světla.

O směrové světlo se jedná, když jsou všechny paprsky nekonečně dlouhé a jsou k sobě rovnoběžné ve směru definovaném vektorem. Osvětlují celou scénu a hodí se pro zdroje světla, které jsou extrémně daleko od povrchu (např. Slunce).

Dále existuje bodové světlo, které od určitého definovaného bodu vyzařuje světelné paprsky do všech směrů. Za příklad z reálného světa lze považovat žárovka. Aby klesala intenzita světelného záření se vzdáleností od světelného zdroje, součástí bodového světla je vlastnost zvaná útlum, kterou je možné vidět na obrázku 2.7. Útlum lze počítat několika způsoby. Dle [10] lze využít tzv. „útlum metodou inverzních čtverců“, kde výsledná hodnota světla c_l je vypočítána ze zdrojové hodnoty c_{l_0} za pomoci vzdálenosti od světelného zdroje r a tzv. „pevné referenční vzdálenosti“ r_0 následovně:

$$c_l(r) = c_{l_0}(\frac{r_0}{r})^2 \tag{2.17}$$

Lze si povšimnout, že ve jmenovateli vzorce 2.17 může hodnota r dosáhnout 0. To se dá vyřešit přičtením pevné hodnoty ϵ nebo využitím funkce max v shaderu s definovanou minimální hodnotou.



Obrázek 2.7: Znázornění bodového světla a jeho vlivu na povrch včetně útlumu[10].

Jako poslední uvedu kuželovité světlo, které se pouze od bodového světla liší tím, že je omezeno množství paprsků definovaným kuželem. Světelné paprsky jsou pak vyzařovány ze světelného zdroje pouze ve směrech tak, aby byly uvnitř tohoto kuželu.

[10]Obrázek převzat z webové stránky [navštíveno 29.04.2021]: https://learnopengl.com/Lighting/Light-casters

Hlavní výhodou těchto zdrojů světla je dle [9] zjednodušení vzorce 2.1 na následující:

$$L_o(v) = \pi f(l_c, v) \otimes c_l \underline{(n * l_c)} \tag{2.18}$$

Podtržené $\underline{(N * L_c)}$ znamená, že tato hodnota nesmí být záporná. Hlavní výhodou je optimalizace díky nahrazení integrálu. V praxi se proto využívají hlavně tyto zdroje světla pro osvětlení. Pro více světel stačí výsledné hodnoty L_o sečíst. Kvůli výraznému zjednodušení modelu na tyto zdroje světla, byly zanedbány ostatní, méně výrazné zdroje světla, o kterých je třeba taktéž uvažovat. Je proto vhodné tento model kombinovat s nějakými dalšími, například s modelem ambientního osvětlení, nebo s modelem IBL (Image-Based Lighting) z kapitoly 2.5.3.

2.5.2 Ambientní osvětlení

Jedno z nejprimitivnějších řešení osvětlovacího modelu. Není založeno na fyzikálních vlastnostech, ale na domnění, že v žádné částí modelu nemůže dojít k úplné absenci světla, především toho nízkofrekvenčního. Model uvažuje, že na každý bod dopadá stejné množství tohoto nízkofrekvenčního světla ze všech směrů. Toto ambientní osvětlení je možno ignorovat v případě, když využíváme nějakého hodně přesného fyzikálního modelu, který dokáže aproximovat přesně i tyto nízkofrekvenční paprsky světla.

2.5.3 Image-Based Lighting

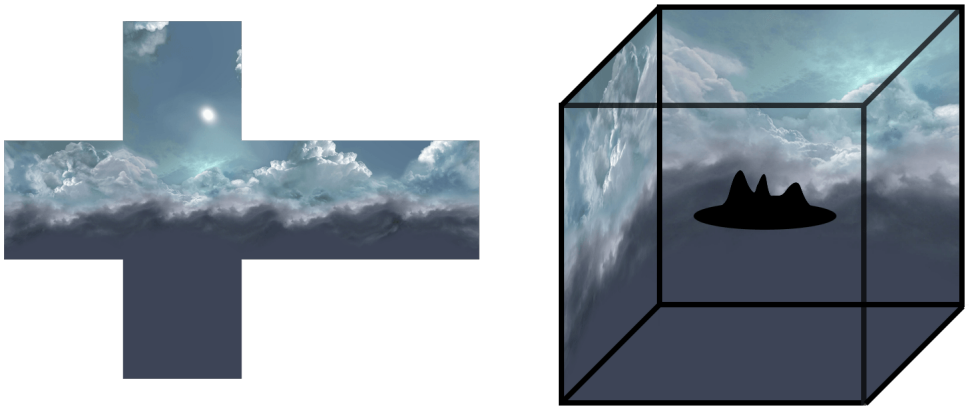
Využívá tzv. *Evironment map* neboli „textury prostředí“ pro simulování vzdálených zdrojů světla. Dříve se tato technika využívala pro vykreslení extrémně reflektivních objektů, jako například zrcadel. Později se tyto mapy prostředí začaly používat i pro méně reflektivní, především metalické, povrchy. Často se využívá metoda Monte Carlo, kdy pro co nejpřesnější aproximaci odrazu je potřeba co nejvíce vzorků. Ovšem je nutné si zvolit vhodnou chybovostní odchylku dle požadovaných nároků na kvalitu aproximace, jinak může dojít ke zbytečnému vytížení hardwaru. Ke snížení nároků na množství vzorků se využívá tzv. *Importance sampling*, neboli „vzorkování dle důležitosti“ [9]. Rovnice aproximace integrálu vypadá následovně [7]:

$$\int_H L_i(l) f(l, v) cos\theta_l dl \approx \frac{1}{N} \sum_{k=1}^{N} \frac{L_i(l_k) f(l_k, v) cos\theta_{l_k}}{p(l_k, v)} \tag{2.19}$$

I přes značné snížení nároků na množství vzorků díky *Importance sampling*, nebo i přes využití mip-map optimalizace, stále je tato technika náročná na výpočet a pro plynulé využití v reálném čase. Proto se suma ze vzorce 2.19 může aproximovat do dvou sum, které se za určitou cenu kvality dají předvypočítat pro dané textury.

2.6 Skybox

Skybox je technikou pro simulaci vzdáleného okolí [5]. V podstatě se jedná o krychli, na jejíž vnitřní strany se nanáší textury tak, aby se z pohledu z vnitřku krychle zdálo, že je kolem nás prostředí. Sada šesti textur, určených pro skybox, se nazývá *cubemap* a je znázorněna na obrázku 2.8. Skybox má většinou stálou pozici vzhledem ke kameře a vytváří tak iluzi, že je prostředí nekonečně daleko. V některých případech můžeme chtít, aby se kamera pohybovala vzhledem k prostředí, a proto skybox musí být dostatečně velký, aby se při posunu nedostala kamera mimo hranice krychle. V praxi se lze setkat s konceptem skyboxu, kde se ale místo krychle využívá koule. Tato obdoba se nazývá *skydome*.



Obrázek 2.8: Ukázka rozložené cubemapy a složené do skyboxu[11].

2.7 Povrchové nerovnosti

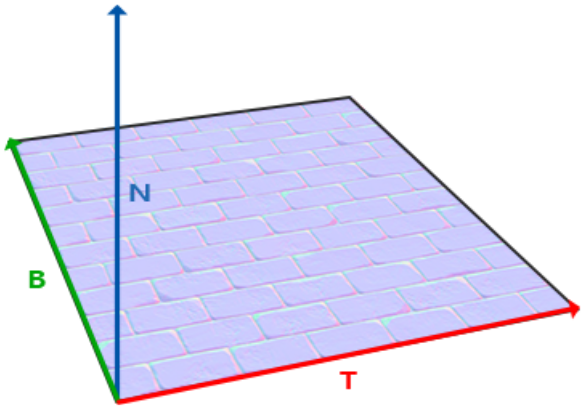
Povrchové nerovnosti přidávají modelům výrazně na realitě obzvláště, když se tato technika kombinuje s physically-based osvětlením. Existuje mnoho technik, realizujících simulaci nerovností, počínaje *Bump mapping* [2]. Zaměřím se především na *Normal mapping* a na *Parallax mapping* [6], které patří mezi klasické metody využívané ve většině případů. Za zmínku taktéž stojí metoda *Displacement mapping* [3]. Jedná se o pokročilejší techniku, která vizuálně nesimuluje hloubku, ale skutečně upravuje množství a pozice vrcholů a umožňuje tak tvarovat model.

2.7.1 Normal mapping

Normal mapping je technika, umožňující simulaci nerovností na základě normálových vektorů, které jsou definovány pro každý bod povrchu pomocí textury. Normálový vektor je v rámci textury reprezentován barvou RGB, kde každá složka slouží jako souřadnice. Jelikož většinou vektor směřuje vzhůru od plochy, normálová textura bývá zbarvená do odstínu modré až fialové. To ale taktéž značí, že normály jsou reprezentovány v tangent prostoru (Tangent space).

[11] Obrázek převzat z webové stránky [navštíveno 26.04.2021]: https://www.pngegg.com/en/png-ikotn

Tangent space je definován třemi vektory tangent T, bitangent B a normal N. Vektory T a B značí plochu kolmou k vrcholu ve směrech vektorů UV[12]. Takže se jedná o vektory kolmé k normále N a zároveň tečící plochu s vrcholem, odkuď normála N pochází. Tyto vektory lze společně s normálovou texturou spatřit na následujícím obrázku 2.9.



Obrázek 2.9: Ukázka vektorů T, B, N na ploše znázorňující normálovou texturu[13].

Pospolu nám tyto tři vektory dávají matici pro transformaci v rámci tangent space. Často se převádí vektory z world space do tangent space, protože je rychlejší převést vektory určené pro jeden vrchol, než pak přepočítávat každý fragment. Hlavní nevýhodou této techniky je, že při pohledu v nízkém úhlu vzhledem k povrchu, je značně vidět falešnost simulované hloubky nerovností, protože technika neřeší výšku nerovností.

2.7.2 Parallax mapping

Parallax mapping [6] je technikou, která za pomoci výškové textury řeší problém normal mappingu, zmíněný v minulé kapitole 2.7.1. Principem techniky je změna UV souřadnic dle výškové mapy. Při pohledu v nízkém úhlu na povrch, se na základě výškové mapy dopočítá posun těchto souřadnic a nerovnosti se pak zdají reálnější. Avšak i tato technika se setkává s problémy. Technika neřeší okluzi a při nižších úhleh se stále můžeme setkat s nesrovnalostmi. Proto se lze setkat s rozšiřujícími technikami, jako například s *Steep parallax mapping* nebo *Parallax occlusion*. Tyto techniky jsou však iterativními a tudíž vyžadují více výpočetních zdrojů.

[12]UV vektory jsou 2D souřadnice určeny pro techniku *UV mapping*, která slouží k nanesení 2D textury na 3D model.

[13]Obrázek převzat z webové stránky [navštíveno 29.04.2021]: https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Kapitola 3

Návrh

Podle zadaní se praktická část této práce skládá ze dvou částí. První je knihovna realizující PBR dle technik představených v kapitole 2. Druhou je program, který využívá zmíněnou knihovnu pro vykonání vykreslení objektů s definovanými fyzikálními vlastnostmi v reálném čase. Mezi hlavní požadavky, které by měla tato práce umožňovat patří:

- Načtení/odstranění modelů
- Úprava pozice a vlastností modelů
- Změna textur materiálů jednotlivých modelů
- Úprava pozice a vlastností lokálních světel
- Změna skybox cubemap textury
- Modifikace pohledu kamery
- Zobrazení informací o běhu programu (např. FPS)

Začnu tedy první s návrhem PBS technik představených v kapitole 2, které budou využívány knihovnou aplikace.

3.1 BRDF

V této podkapitole vyberu jednotlivé složky BRDF funkce z kapitoly 2.3 a objasním, proč jsem si právě tyto techniky vybral. Odrazil jsem se od prvotního BRDF vzorce 2.4, navrženého původním modelem Cook-Torrance [4]. Nahradil jsem π za 4 a taktéž jsem si BRDF rozdělil na tři části, kvůli důvodům popsaným v této kapitole. Především jsem se inspiroval modelem GGX (Trowbridge-Reitz) [14], který se řadí mezi nejefektivnější a nejvyužívanější physically-based model dnešní doby. Za rozumnou cenu hardwaru slibuje poměrně kvalitní aproximaci odrazu světla od povrchu. Ještě než začnu s návrhem částí povrchové odrazivosti, nesmím zapomenout na podpovrchovou odrazivost. Jak se píše v kapitole 2.4, i já jsem se rozhodl využít Lambertovy diffusní složky.

První částí je Fresnelova funkce, kterou jsem vybíral dle kapitoly 2.3.1. V praxi se většinou využívá Schlickovi aproximace [12], ale já jsem se rozhodl využít o trochu více optimalizovanou variantu Schlickovi aproximace se vzorcem 2.9. Hlavním parametrem je index lomu F_0, který se pro každý druh materiálu liší. Jelikož se Fresnelova odrazivost

pohybuje většinou kolem hodnoty funkce pro 0°, považujeme tuto hodnotu za „charakteristickou zrcadlovou odrazivost“ materiálu [9] a přibližně odpovídá barvě materiálu o určité odrazivosti světla. Často se pro výpočet této hodnoty využívá „lineárního míchání“[1] malé konstantní hodnoty a barvy materiálu na základě metalicity materiálu.

V druhé části, která se věnuje vlastnostem odlesku, je potřeba vybrat vhodnou distribuční funkci. Rozhodl jsem se využít distribuční funkci z modelu GGX. Tento model má jasně definovanou distribuční funkci, jejíž vzorec 2.11 lze nalézt v kapitole 2.3.2. Distribuční funkce pak vypadá následovně:

$$D_{pl}(h) = \frac{(n * h)r^2}{4(1 + (n * h)^2(r^2 - 1))^2} \tag{3.1}$$

Třeba si všimnout, že po substituci distribuční funkce D ve vzorci 2.7, ze vzorce 3.1 kompletně zmizelo π, neboť se nacházelo v čitateli i jmenovateli a došlo k jeho vykrácení. Na namísto π v původním vzorci 2.11 ve jmenovateli přišla 4.

Poslední částí je funkce viditelnosti, která disponuje geometrickou funkcí. Opět po vzoru modelu GGX jsem vybral Smithovu aproximaci ze vzorce 2.13, o které je možné se více dočíst v kapitole 2.3.3. Součástí Smithovy aproximace je funkce G_1 se vzorcem 2.14, která disponuje parametrem k. Model GGX definuje pro různá osvětlení jiné parametry k. Jak se lze v následující kapitole 3.2 dočíst, budu využívat IBL osvětlení (zmíněné v kapitole 2.5.3), pro které model GGX stanovuje parametr $k = \frac{r^2}{2}$. Funkce G_1 je následující:

$$G_1(v) = \frac{v * n}{(1 - \frac{r^2}{2})(v * n) + \frac{r^2}{2}} \tag{3.2}$$

3.2 Osvětlovací model

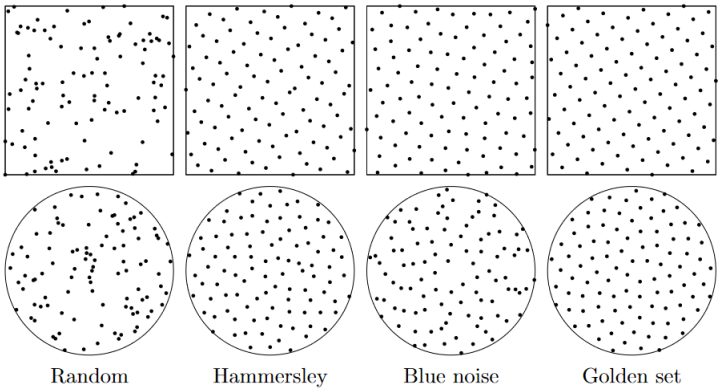
Podle zmínky v kapitole 2.5, se většinou kombinuje více druhů osvětlení. Rozhodl jsem se, že využiji všech třech zdrojů světla z této kapitoly. V následujících odstavcích se dovíme, co mě k tomu vedlo.

Jako lokální zdroj světla jsem se, z hlediska účelu programu, rozhodl využít bodového světla definovaného v kapitole 2.5.1. Mezi hlavní důvody patří využití právě tohoto zdroje v Základním vzorci odrazu 2.1. A právě díky využití lokálního osvětlení je možná optimalizace na vzorec 2.18, jak je popsáno v kapitole 2.5.1. Odraz světla se bude věnovat pouze těmto umělým lokálním zdrojům světla. Bodová světla mají s rostoucí vzdáleností od povrchu útlum intenzity svitu na základě metody inverzních čtverců, zmíněné v kapitole 2.5.1. Hodnota útlumu se vzdáleností by měla být nastavitelná v rámci grafického rozhraní.

Samozřejmě to není z hlediska PBR dostačující a je potřeba využít i dalšího druhu osvětlení. Jedná se složitý o Image-Based Lighting z kapitoly 2.5.3. Tento zdroj světla bude sloužit pro vzdálené, méně významné zdroje světelných paprsků, které pocházejí z okolí definovaného skyboxem. Metoda je daleko výpočetně náročnější a slouží pro aproximaci poměrně důležité části osvětlení, které bylo v rámci optimalizace pomocí lokálních osvětlení z kapitoly 2.5.1 vyřazeno. Jelikož se jedná o techniku založenou na metodě *Quasi Monte Carlo*, která se od běžné metody Monte Carlo liší v použití generátoru čísel s pravidelnými skoky[2] namísto generátoru pseudonáhodných čísel, využívá se specifických sekvencí čísel. Většinou jsem se setkal s využitím *Hammersley* sekvence. Rozhodl jsem se dle [13] využít

[1] V jazyku GLSL je tato technika známá jako funkce *mix*.
[2] Lze hovořit o generátorech s nízkým rozporem.

sekvence založené na zlatém řezu. Technika s využitím zlatého řezu není příliš populární. Na obrázku 3.1 vidíme, že je o trochu přívětivější pro metodu *Quasi Monte Carlo* a v případě IBL může dojít k mírným rozdílům. Rozdíl není tolik markantní a techniku založenou na zlatém řezu, jsem si vybral spíše z experimentálních důvodů.



Obrázek 3.1: Porovnání generátorů pro metodu *Quasi Monte Carlo* [13].

Obvykle využití těchto dvou metod aproximuje většinu potřebných zdrojů osvětlení, ale stále jsou ignorovány drobné, nízkofrekvenční zdroje světla. Proto jsem využil i ambientní osvětlení z kapitoly 2.5.2. Jak je zmíněno v této kapitole, pokud námi zvolený model není schopen aproximovat nízkofrekvenční zdroje světla, je vhodné využít tohoto primitivního řešení.

Tyto druhy osvětlení spolu tvoří celkem slušný osvětlovací model, který jsem se rozhodl využít ve své práci. V kombinaci s BRDF funkcí ve vzorci odrazu, můžeme dosáhnout vlastností, které zdánlivě aproximují světelné paprsky fyzikálního modelu reálného světa. Samozřejmě v reálném světu existují materiály, u kterých by se vizuální podobnost hledala jen stěží (např. kůže, svíčka, krystal, látka), neboť při aproximacích došlo k ignorování vlastností, které u některých materiálů mohou být klíčové. Především se jedná o *Subsurface Scattering*, který řeší energii pohlcenou materiálem. Řešení pohlcení energie a její přenos materiálem, jsem považoval nad rámec své práce a osobně to považuji za možné budoucí rozšíření.

3.3 Knihovna

Podkapitola slouží k navržení knihovny dle požadavků na počátku této kapitoly 3. Knihovna má sloužit pro ulehčení práce s načtením objektů a s jejich následným vykreslením. Knihovnu jsem rozdělil na dvě podčásti. První část má za úkol zakrýt a usnadnit práci s grafickým API. Druhá část slouží k načtení objektů a textur z fyzického úložiště.

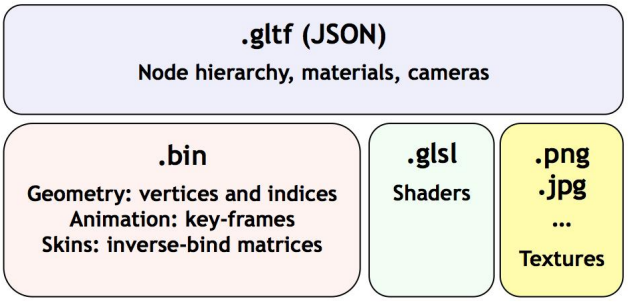
Knihovna by měla poskytovat struktury pro reprezentaci physically-based modelu a měla by disponovat funkcemi pro načtení a vykreslení tohoto modelu.

3.3.1 Načítaní modelů

Načtení modelů do datové struktury se z hlediska návrhu neobejde bez zvolení formátu zdrojového fyzického souboru. Rozhodl jsem se kvůli jednoduchosti a také kvůli jeho podporovanému zaměření na PBR, využít standardu glTF. V úvahu také stál formát od společnosti Autodesk s názvem FBX. Tento formát je mnohem složitější a tedy i víceúčelovější.

Vzhledem k obtížnosti implementace tohoto formátu, jsem se rozhodl jej přidat mezi možnosti rozšíření této práce (viz kapitola 5).

Standard *glTF 2.0* je poměrně novým, open-source formátem, vytvořeným Khronos Group. Tento formát definuje dva způsoby, kterými lze data ukládat na disku. Prvním je klasický přístup po vzoru formátu OBJ, kde jsou informace uloženy kódováním ASCII v souboru a textury jsou obsaženy zvlášť mimo tento soubor. V tomto případě má tento soubor příponu „.gltf“. V druhém případu jsou veškeré informace uloženy binárně v jediném souboru a tento soubor má pak příponu „.glb“. Strukturu tohoto binárního formátu glTF můžeme vidět na následujícím obrázku 3.2.



Obrázek 3.2: Vizualizace jednotlivých částí binárního souboru formátu glTF[3].

Zmíněné formáty podporují mnohé vymoženosti, které jsem ve své práci mohl využít. Formát byl založen s vědomím, že bude sloužit především pro objekty určené pro PBR. Ale navíc podporuje i vertex skinning, animace a kamery pro vytvoření kompletní scény. Setkal jsem se s problémem, že tento standard nepodporuje výškové mapy, které lze využít pro parallax mapping nebo třeba i displacement mapping. Jelikož se ale jedná o open-source standard, na oficiálním githubu se již tento problém řeší. Protože se jedná o poměrně nový a nepříliš rozšířený standard, nebyl zatím dostatek vůle ho aktualizovat. Problém lze vyřešit využitím vlastního rozšíření, které tento standard podporuje, ale samozřejmě se nejedná o normalizované řešení. Jelikož má práce umožňuje načítat a nahrazovat textury za běhu aplikace, nejjednodušším řešením by bylo načítat výškovou mapu explicitně pro každý model.

Tedy knihovna by taktéž měla umožňovat načtení jednotlivých textur ze souborů. Kromě klasických formátů JPG a PNG jsem se rozhodl využít i formátu KTX. Tento formát je opět výtvorem Khronos Group. Liší se od normálních bitmapových obrázků tím, že je předem určen pro grafická API (především právě pro OpenGL a Vulkan) a tedy dopředu uvažuje, že se nejedná o pouhý bitmapový obrázek, ale o texturu. Rozdíl je především v tom, že kromě obyčejných 2D textur podporuje i pole textur nebo třeba cubemapy a to i s mipmapami. A právě kvůli cubemapám, které využiji pro skybox, jsem se rozhodl využít i tohoto formátu. Taktéž další výhodou je možnost komprese těchto dat navržená přímo pro grafická API, která následně pak jednodušeji zpracovávají tyto textury na grafické kartě.

[3]Obrázek převzat z webové stránky [navštíveno 02.05.2021]: https://medium.com/hello-meets/aframe-vr-skybox-configurator-515c70df9ae1

3.4 Vizualizační aplikace

V návrhu vizualizační aplikace se především zaměřím na propojení tohoto programu s návrhem knihovny. Zmínka v kapitole 3.3. Zde se budu zabývat návrhem grafického rozhraní pro ovládání scény na základě požadavků zmíněných na začátku kapitoly 3.

Cílem vizualizační aplikace je využít knihovny pro vykreslení scény, kde je možné v reálném čase měnit vlastnosti a pozorovat jejich ovlivnění physically-based modelu. Aplikace umožňuje uživateli načíst vlastní model a skybox. Kromě fyzikálních vlastností může uživatel měnit i osvětlení nebo kameru. V reálném čase lze procházet vykreslovanou scénu.

Mezitím, co knihovna disponuje funkcemi pro vykreslení physically-based modelu v reálném čase, vizualizační aplikace by měla být schopna tento proces řídit a dle vstupu od uživatele i měnit faktory jednotlivých částí vykreslovacího procesu.

3.4.1 GUI

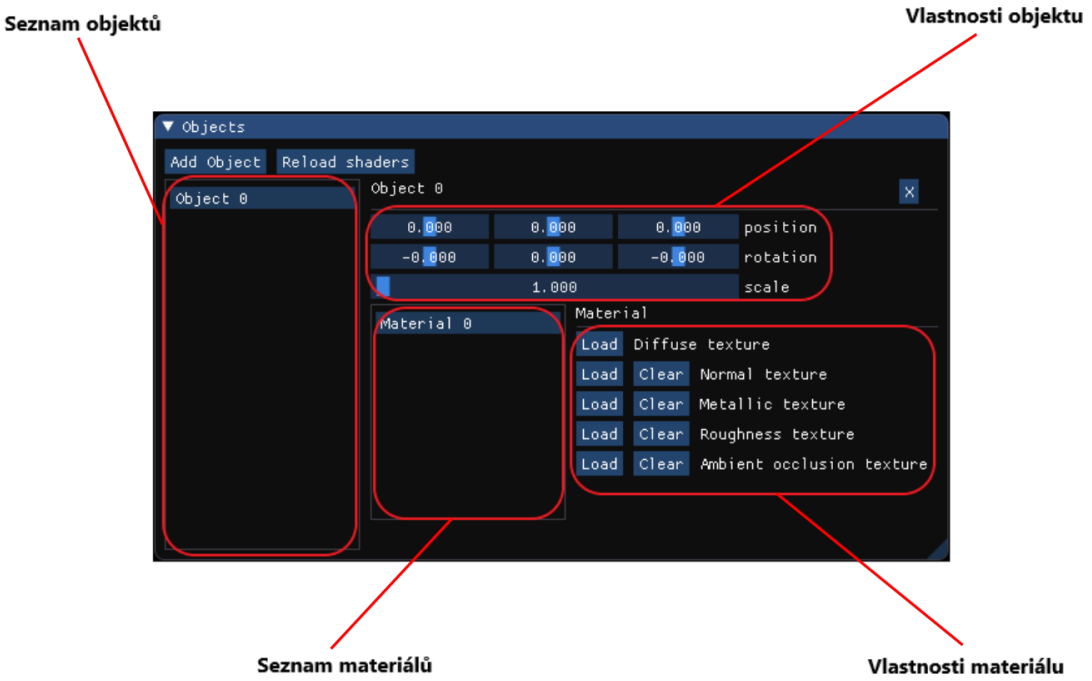
Co se týče grafického rozhraní, tak by mělo být založeno na jednoduchosti a intuitivním zacházení. Na základě těchto požadavků jsem jednotlivé prvky GUI rozdělil do tří částí. Co se týče ovládání kamery, tak to není prováděno v rámci grafického prostředí, ale je využívána klávesnice s myší. Více informací ohledně ovládání programu za pomoci klávesnice a myši lze nalézt v příloze A.

První částí je okno, které spravuje scénu jako celek. Týká se to lokálních osvětlení a skyboxu. Skybox bude možné měnit načtením vlastní cubemapy. Lokálních osvětlení je přesný počet a nacházejí se vlevo v seznamu. Po zvolení je možné měnit jejich pozici, barvu, útlum a zapnutí osvětlení. Tedy v tomto okně se nastavují vnější světelné vlivy na objekty scény. Na následujícím obrázku 3.3 se nachází ukázka tohoto okna.

Obrázek 3.3: Ukázka grafického prostředí spravujícího části scény.

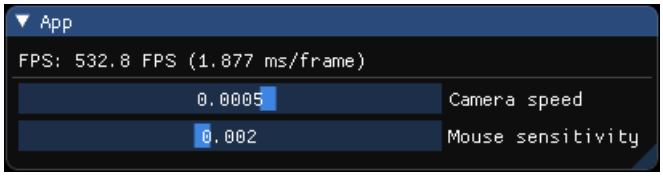
Druhou částí je okno, spravující všechny načtené objekty ve scéně. Dělí se na dvě části. V levé části je seznam všech načtených objektů. Po kliknutí na jeden z těchto objektů dojde

k zobrazení vlastností, pozice a podčásti, která se skládá ze seznamu materiálu a informace o zvoleném materiálu. Tyto informace se zobrazí v pravé části okna. Některé položky mají pouze informativní hodnotu a jiné je možné upravovat, například pozici objektu. Totéž platí i pro jednotlivé materiály, kde je možné měnit textury objektu. Za pomoci tohoto okna je možné měnit jednotlivé objekty z této scény podle vlastních potřeb. Opět se na následujícím obrázku 3.4 nachází ukázka okna s objekty.



Obrázek 3.4: Náhled na grafické prostředí spravující objekty ve scéně.

Poslední částí je okno, které pouze vypisuje aktuální stav o běhu programu a umožňuje částečné nastavení ovládání. Konkrétně, jak je možné spatřit na obrázku 3.5, se zobrazuje množství snímků vykreslených za sekundu, čas potřebný k vykreslení snímku a dále se také, v této části grafického prostředí, nachází ovládání rychlosti pohybu kamery a citlivost myši pro změnu orientace kamery.



Obrázek 3.5: Ukázka grafického prostředí zobrazující informace o vykreslení a spravující nastavení ovládání kamery.

Kapitola 4

Implementace

V této kapitole přesně popíši implementační proces na základě návrhu z kapitoly 3. Jako první představím technologie, kterých jsem využil při vývoji jak knihovny, tak programu, který tuto knihovnu využívá. Týká se to i externích knihoven, které mně především pomáhaly v oblastech načítání modelů a v ulehčení práce s implementací grafického rozhraní. Dále se budu věnovat implementaci funkce BRDF a osvětlovacího modelu. Součástí jsou i části kódu a pseudokódy, které reprezentují jednotlivé funkční složky. Poté bude následovat implementace knihovny a následně i samotné vizualizační aplikace.

4.1 Využité technologie

Program jsem se rozhodl napsat v jazyku C++, protože je to nejběžnější jazyk pro implementaci programů, které využívají grafické aplikační prostředí (API). V této podkapitole popíši všechny vybrané technologie, které jsem při implementaci využil, včetně grafického API, čímž začnu.

Jako grafické API jsem zvolil Vulkan od společnosti Khronos Group, který je považován za nízkoúrovňovou alternativu k OpenGL, a proto implementace za pomoci Vulkan API je náročnější. Zvolil jsem si Vulkan, protože jsem chtěl prozkoumat a lépe pochopit proces vykreslování na nižší úrovni.

Další důležitá technologie je GLFW[1] aplikační prostředí, které poskytuje multiplatformní vytvoření a správu nad okny včetně událostí, jako například vstupu z klávesnice. GLFW API jsem si vybral kvůli jeho podpoře Vulkan API a taktéž kvůli tomu, že je multiplatformní a poskytuje snadnou správu nad událostmi v okně aplikace.

Za zmínku taktéž stojí knihovna ImGui[2]. Jedná se o open-source integraci grafického prostředí, díky kterému lze jednoduše implementovat vykreslení objektů typu tlačítko, posuvník, atd., jež jsou samozřejmě responzivní. Knihovnu jsem si zvolil kvůli zjednodušení práce s GUI, které je potřeba implementovat a protože tato knihovna podporuje Vulkan API. Pro procházení fyzického úložiště zařízení v tomto grafickém rozhraní používám rozšiřující knihovnu ImGui-filebrowser[3]. Tato knihovna využívá standardní knihovnu C++ *filesystem*, která je dostupná od verze C++17, a to je minimální verze jazyka C++, což má práce vyžaduje.

[1] https://www.glfw.org/
[2] https://github.com/ocornut/imgui
[3] https://github.com/AirGuanZ/imgui-filebrowser

Využil jsem i dalších knihoven. Jednou z nich, která je součástí hromady vizualizačních programů, je GLM[4]. GLM je open-source knihovna, která představuje mnoho matematických struktur včetně operací a funkcí. Mezi její hlavní výhody patří, že její struktury a funkce byly převzaty z jazyka shaderu GLSL. K těmto strukturám patří například vektor nebo matice, ale také i kvaterniony, využité při implementaci kamery. Využil jsem i knihovnu od stejných tvůrců s názvem GLI[5]. Ta umožňuje načítání textur formátu KTX a DDS a také obsahuje řadu funkcí pro další práci. K načítání využívám knihoven stb_image[6], tiny gltf[7], JSON for Modern C++[8]. Knihovna pro práci s formátem JSON je zde kvůli tomu, že je využívána knihovnou tiny gltf, neboť právě formát glTF je založen nad JSON (viz kapitola 3.3.1).

4.2 BRDF

Podle návrhu BRDF funkce z kapitoly 3.1 jsem nejprve implementoval jednotlivé části BRDF funkce a poté jsem tyto části spojil do kompletní funkce odrazu.

Fresnelova funkce není nijak implementačně složitá. Dle návrhu Fresnelovi funkce z kapitoly 3.1 jsem vzorec implementoval následně:

```
1  vec3 F_Schlick_opt(vec3 F0, float VdotH)
2  {
3      float ex = (-5.55473 * VdotH - 6.98316) * VdotH;
4      return F0 + (1.0 - F0) * pow(2.0, ex);
5  }
```

Prvním parametrem je index lomu $F0$, jehož výpočet si v této kapitole uvedeme později. Druhým parametrem je skalární součin vektoru ke kameře a half-angle vektoru.

Jak je zmíněno v kapitole 3.1, distribuční funkci jsem převzal z modelu GGX a dle návrhu jsem ji implementoval takto:

```
1  float Dpl_GGX_TrowbridgeReitz(float NdotH2, float roughness2)
2  {
3      float denom = 1.0 + NdotH2 * (roughness2 - 1.0);
4      return roughness2 / (4.0 * denom * denom);
5  }
```

Prvním parametrem *NdotH2* je skalární součin half-angle vektoru s normálovým vektorem, přičemž je tento součin druhé mocniny. Druhým parametrem je hodnota drsnosti povrchu taktéž druhé mocniny.

Poslední částí BRDF funkce je viditelnost, kterou jsem si rozdělil do dvou funkcí. První funkcí je Smithova aproximace pro GGX model, kterou jsem dle návrhu z kapitoly 3.1 implementoval tímto způsobem:

[4] https://github.com/g-truc/glm
[5] https://github.com/g-truc/gli
[6] https://github.com/nothings/stb
[7] https://github.com/syoyo/tinygltf
[8] https://github.com/nlohmann/json

```
float G_GGXSmith(float NdotV, float NdotL, float roughness2)
{
    float k~= roughness2 / 2.0;
    float nk = 1.0 - k;

    float g1 = NdotL / (NdotL * nk + k);
    float g2 = NdotV / (NdotV * nk + k);
    return g1 * g2;
}
```

Prvními dvěma parametry jsou skalární součiny normály s vektorem ke kameře a normály s vektorem ke zdroji světla. Třetím parametrem je opět druhá mocnina drsnosti povrchu.

Geometrickou funkci jsem nechal osamostatněnou od funkce viditelnosti, neboť později bude využita v implementaci osvětlení. Funkce viditelnosti má stejné parametry jako implementovaná geometrická funkce. Jediné co provádí, že výsledek geometrické funkce podělí násobku skalárních součinů, které jsou reprezentovány prvními dvěma parametry. Implementovaná funkce viditelnosti vypadá takto:

```
float visibility(float NdotV, float NdotL, float roughness2)
{
    return G_GGXSmith(NdotV, NdotL, roughness2) / (NdotV * NdotL);
}
```

Tyto tři funkce nám pospolu definují povrchovou odrazivost modelu, avšak nesmíme opomenout i podpovrchovou odrazivost, která se implementuje za pomocí Lambertovy funkce. Nyní po implementování jednotlivých částí BRDF nezbývá nic jiného, než tyto části spojit do jedné funkce. Jako první si vypočítáme parametr F_0 pro Fresnelovu funkci. Jak je zmíněno v kapitole 3.1, využívá se lineárního míchání konstanty 0.4 a barvy, které jsou smíchány na základě metalicity materiálu. Následně se vypočítají jednotlivé složky BRDF a navzájem se vynásobí. Po výpočtu odrazivosti povrchu se vypočítá Lambertova funkce. Nesmíme opomenout, že materiál na základě matelicity má schopnost absorbovat část energie. Tento faktor absorbce je v následujícím kódu označen jako k_D. Výsledek BRDF funkce je součet povrchové a podpovrchové odrazivosti. Pseudokód funkce BRDF je následující:

Algoritmus 1: Pseudokód BRDF funkce

Input: $N, L, V, roughness, metallic, albedo$
Output:

$H = normalize(V + L)$
$F_0 = mix(0.04, albedo, metallic)$
$F = FresnelFnc(F_0, V \cdot H)$
$D_{pl} = DistributionFnc((N \cdot H)^2, roughness^2)$
$vis = VisibilityFnc(N \cdot V, N \cdot L, roughness^2)$
$specular = F * D_{pl} * vis$
$lambert = albedo/\pi$
$k_S = F$
$k_D = 1.0 - k_S$
$k_D = k_D * (1.0 - metallic)$
return $k_D * lambert + specular$

4.3 Osvětlovací model

V této podkapitole se zaměřím na implementaci osvětlovacího modelu dle kapitoly 3.2. Výsledný osvětlovací model se bude skládat ze tří částí, jejichž suma nám dá výsledný odraz simulovaného světla od povrchu.

První nejjednodušším osvětlením k implementaci je ambientní osvětlení. Stačí barvu odraženého světla (taktéž nazývanou „albedo“), která je definovaná pro každý bod texturou, vynásobit konstantní hodnotou. Za tuto konstantní hodnotu jsem zvolil číslo 0.04.

Další osvětlení, které bylo potřeba implementovat, bylo lokální osvětlení. Lokální osvětlení využívá BRDF funkce, jejíž implementace byla popsána v předchozí kapitole 4.2. Pseudokód pro výpočet lokálních osvětlení vypadá následně:

Algoritmus 2: Pseudokód výpočtu vlivu lokálních osvětlení

Input: $N, V, lights, world\_pos$
Output: L_0

$L_0 = 0$
for $i = 0$ to $NumLights$ **do**
    $L = normalize(lights[i].pos - world\_pos)$
    $r = length(L)$
    $attenuation = (lights[i].r0 / max(r, 0.0001))^2$
    $radiance = lights[i].color * attenuation$
    $L_o = L_0 + brdf(N, L, V) * radiance * (N \cdot L)$
end for

V tomto pseudokódu si lze povšimnout, že je lokální osvětlení počítáno pro $NumLights$ počet zdrojů a následně se výsledek jen sečte. Ve smyčce se uvažuje útlum světla vzhledem ke vzdálenosti, který ovlivní intenzitu světla. Nejdůležitější částí je poslední řádek ve smyčce, který je implementací základního vzorce odrazu 2.18 a využívá BRDF funkci.

Poslední využité osvětlení IBL bylo implementačně nejtěžší a nejsložitější. Nejprve bylo zapotřebí implementovat generátor sekvenčních čísel. Jak jsem zmínil v kapitole 3.2, využil jsem sekvence založené na zlatém řezu dle [13]. Funkci jsem implementoval takto:

```
1 vec2 fibonacci_2D(uint i, uint N)
2 {
3     return vec2(float(i+1) * GOLDEN_RATIO, (float(i)+0.5) / float(N));
4 }
```

Funkce má dva parametry. Prvním parametrem je index, aneb kolikátý vzorek se právě zpracovává. Druhým je celkový počet počítaných vzorků.

Jelikož jsem se rozhodl využít i techniku *Importance sampling* pro snížení nároků na množství vzorků, musel jsem implementovat i tuto funkci. Kód této funkce jsem převážně převzal z [7]. Pak už se můžu zapojit do implementace samotné metody IBL, která využívá generátoru a funkci techniky importance sampling. Pseudokód jsem opět převzal z [7] a vypadá následně:

Algoritmus 3: Pseudokód metody výpočtu IBL s využitím Importance Sampling

Input: $spec, roughness, N, V$
$sum \leftarrow 0$
for $i = 0$ to $NumSamples$ **do**
    $X^i = random\_gen(i)$
    $H = ImportanceSample(X^i, roughness, N)$
    $L = normalize(2.0 * (V \cdot H) * H - V)$
    if $(N \cdot L) > 0.0$ **then**
        $sample\_color = texture(env\_texture, L)$
        $G = GeometricFnc(N \cdot V, N \cdot L, roughness^2)$
        $G_{vis} = G * (V \cdot H)/((N \cdot H) * (N \cdot V))$
        $F_c = (1.0 - (V \cdot H))^{5.0}$
        $F = (1.0 - F_c) * spec + F_c$
        $sum = sum + (sample\_color * F * G_{vis})$
    end if
end for
return $sum/NumSamples$

Vstupem této metody jsou barva odlesku *spec*, drsnost materiálu *roughness*, vektor normály N a vektor ke kameře V. Třeba povšimnout, že se jedná o metodu založenou na Monte Carlo. Přesnost metody závisí na předem definovaném množství vzorků *NumSamples*. Ve své práci jsem si množství vzorků nastavil na 16. S pomocí importance sampling to stačí na dosáhnutí přibližných výsledků. Textura prostředí je v pseudokódu reprezentována proměnou *env_texture*. Získání barvy pro vzorek probíhá s pomocí funkce *texture*, která z textury extrahuje barvu bodu dle parametru vektoru L.

4.4 Normal mapping

Implementace techniky normal mapping nejprve spočívala v načtení normálové textury a tangent souřadnic pro jednotlivé vrcholy a poté ve zpracování těchto dat ve grafické pipeline. Konkrétně tedy ve vertex shaderu, kde dochází k výpočtu matice pro převod do tangent space. Jak je popsáno v kapitole 2.7.1 matici je možné vypočítat z normály N a tangent vektoru T, které jsou definované pro daný vrchol a pak je tato matice předána fragment shaderu. Výpočet matice TBN ve vertex shaderu je implementován následně:

```
1 vec3 N = normalize(mat3(ubo.model) * in_normal);
2 vec3 T = normalize(mat3(ubo.model) * in_tangent);
3 vec3 B = normalize(cross(N, T));
4 TBN = mat3(T, B, N);
```

V kódu si lze povšimnout, že vstupní vektory vertex shaderu *in_normal* a *in_tangent* jsou neprve převedeny na potřebný world space a poté jsou počítány složky TBN matice, přičemž bitangent vektor je dopočítán ze zadané normály N a tangent vektoru T.

Ve fragment shaderu se získá výsledná normála z normálové textury, která je převedená do world space za pomoci předané matice z vertex shaderu. Nově získaný normálový vektor se poté využívá ve všech následujících výpočtech.

4.5 Knihovna

Knihovna obsahuje mnoho tříd a funkcí, usnadňujících práci s Vulkan API. V této kapitole představím jednotlivé třídy, reprezentující koncepty, které jsem implementoval za účelem zjednodušení práce s tímto grafickým API. Nejprve se budu věnovat zapouzdření Vulkan struktur dle jednotlivých konceptů, poté se zaměřím na strukturu modelu a nakonec popíši vykreslení physically-based modelu za pomoci zapouzdřených Vulkan struktur.

4.5.1 Zapouzdření Vulkan struktur

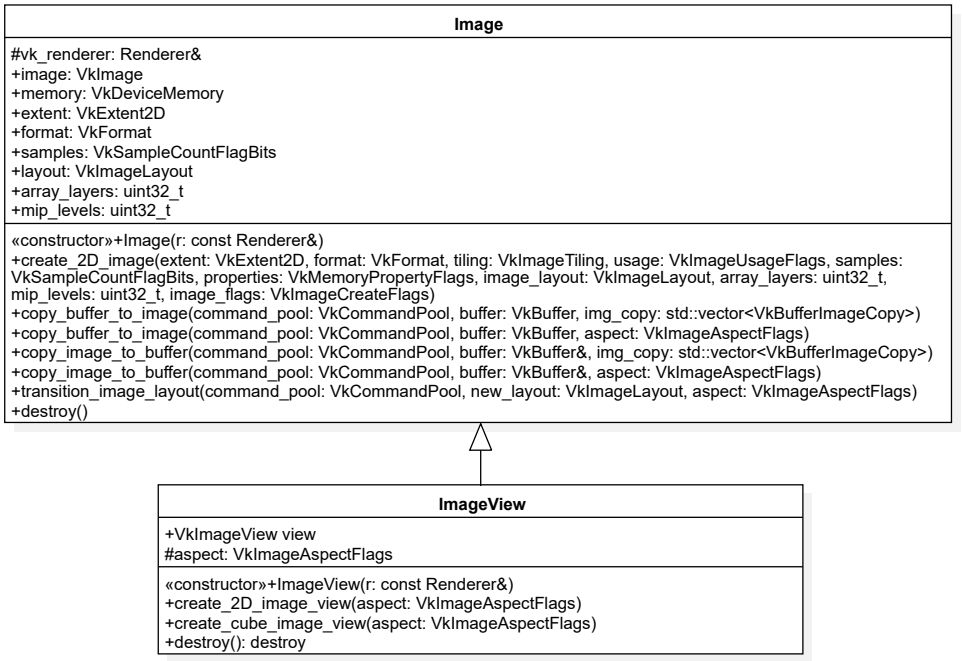
Během implementace mé práce se mi podařilo zapouzdřit několik nejdůležitějších Vulkan struktur z hlediska konceptů. Vzhledem k rozsáhlosti tohoto API, nebylo v mých silách zapouzdřit všechny využívané struktury a proto jsem se zaměřil na ty nejvyužívanější.

Reprezentace obrázku

Jako obrázek lze považovat pole pixelů, které disponují definovaným složením. Zde se budu zabývat konceptem obrázku a jeho reprezentací v prostředí Vulkan. Struktura tříd reprezentující obrázek je ukázána na obrázku 4.1.

Třída *Image* zapouzdřuje obrázek fyzicky uložený na grafické kartě, který je připraven pro zpracování. Třída disponuje vlastnostmi obrázku (např. rozlišení, vrstvy, množství mip-map, formát, atd.) a funkcemi pro kopírovaní obrázku z bufferu nebo pro změnu layout[9].

Druhou třídou je *ImageView*, která je derivovaná od předchozí třídy *Image*. Na základě této třídy tvoří pohled, jenž je následně využit při vykreslovacím procesu. Je možné vytvořit pohled pro 2D povrch a pro cubemapu.



Obrázek 4.1: UML diagram reprezentující třídy *Image* a *ImageView*.

[9] *Image layout* je komprese obrázku v rámci paměti grafické karty pro účely použití v jednotlivých částech grafické pipeline. Při správných změnách layoutu, může doházet ke zmírnění datových toků v rámci GPU.

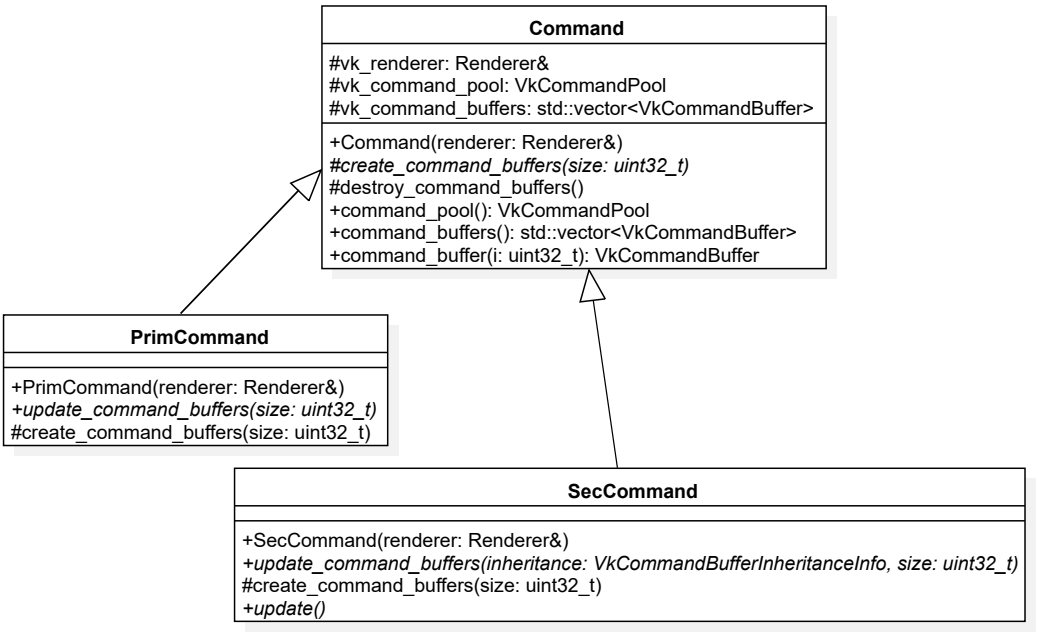
Reprezentace příkazu

Veškeré práce na grafické kartě jsou řízeny jako příkazy v bufferech (*Command buffer*), které jsou součástí nějakého společenství příkazů (*Command pool*). I tyto struktury příkazu z Vulkan API jsem zapouzdřil do tříd.

Třída *Command* se skládá právě z command poolu a z množiny prázdných příkazových bufferů. Třída obsahuje i prázdnou virtualizovanou funkci s názvem *create_command_buffers*, která má dceřiným třídám sloužit k vlastnímu vytvoření příkazových bufferů. Bylo tedy potřeba implementovat i tyto dceřiné třídy dle vlastních potřeb.

Třída *PrimCommand* je derivovaná od této třídy a slouží pro vytváření primárních příkazových bufferů.

Další dceřinou třídou je *SecCommand*, která vytváří sekundární příkazové buffery. Většinou si lze vystačit s primárními příkazy, ale já jsem se rozhodl vykreslovat za pomoci sekundárních příkazů, které budou následně zpracovány příkazy primárními. Na následujícím obrázku 4.2 lze spatřit struktura tříd reprezentující příkazy.



Obrázek 4.2: UML diagram reprezentující třídy *Command*, *PrimCommand* a *SecCommand*.

Pipeline

Nedílnou součástí je také grafická pipeline. V případě Vulkan struktury, jsem se rozhodl ji zapouzdřit do vlastní třídy s názvem *Pipeline*. Jelikož Vulkan umožňuje všelijak upravovat jednotlivé části grafické pipeline, neobešel jsem se bez vytvoření pomocné struktury *PipelineInit* s parametry pro jednotlivé části. Tato struktura je pak parametrem konstruktoru třídy. Pro zjednodušení práce jsem vytvořil funkce pro generování předem připravených struktur *PipelineInit* (např. solid, blend, cubemap, wireframe).

RenderPass

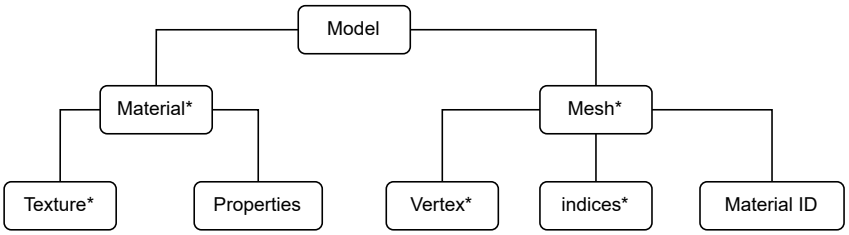
Mezi další zapouzdřující třídy patří *RenderPass*. Třída po vzoru *Pipeline* využívá v konstruktoru strukturu *RenderPassInit*, která opět předá požadované parametry pro vytvoření. Mezi tyto parametry patří tzv. „přílohy“ a popis množiny *subpass*. Většinou se složitější render pass s více subpasses využívají pro změnu image layoutu, k vykreslení hloubky, nebo třeba i k MSAA[10].

Renderer

Renderer je koncept se stejnojmennou třídou, kterou jsem vytvořil k uchování a správě grafického zařízení. Úkolem této třídy je vytvoření instance, vybrání grafické karty, správě zařízení a následném získání fronty z tohoto zařízení. Další součástí této třídy je debug rozhraní schopné logovat procesy s touto instancí. Třída *Renderer* je parametrem konstruktoru téměř ve všech třídách knihovny, neboť většina činností tříd probíhá právě na grafické kartě a *Renderer* je třídou, přes kterou se přistupuje ke grafickému zařízení.

4.5.2 Model

Koncept modelu jsem implementoval několika třídami, které dohromady reprezentují grafický objekt, za který se považuje až několik množin bodů, které dle jednotlivých indexů spolu tvoří povrchy různě tvarovaného tělesa v prostoru. Tyto body reprezentují jednotlivé vrcholy objektu. Pro každou z těchto skupin má model definované vlastnosti, které určují rozptyl světla od povrchu tvořeného body této skupiny. Jelikož se jedná o poměrně složitou strukturu, rozdělil jsem si ji na několik podčástí, přičemž nejsvrchnější třída se nazývá *Model*. Struktura celého modelu je ukázána na následujícím obrázku 4.3.



Obrázek 4.3: Ukázka struktury modelu určeného pro vykreslení.

V první podčásti jsem oddělil do třídy *Vertex* jednotlivé množiny bodů. Vertex je jeden z vrcholů, který obsahuje souřadnice vrcholu, jeho UV souřadnice, normálový vektor a tangent vektor. Slouží k definování jednoho bodu pro vykreslení objektu. Tyto vertexy jsou v praxi posílány přímo na paměť grafické karty, kde je následně zpracovává grafická pipeline a je vhodné je oddělit.

Jednotlivé vertexy z množiny se slučují do skupin nazývaných mesh se stejnojmennou třídou *Mesh*. V rámci meshe lze indexovat vertexy pro optimalizaci vykreslování. Tyto skupiny jsou části povrchu se stejnými materiálovými vlastnostmi, a proto je vhodné vertexy dělit do meshů.

[10]MSAA neboli *Multisample anti-aliasing* je technika pro redukování artefaktů vzniklých při rasterizaci, založená na převedení obrazu do vyššího rozlišení, kde se hrany objektů rozdělí na menší pixely, které se následně při zpětném převodu zprůměrují a tím dojde k vyhlazení hran.

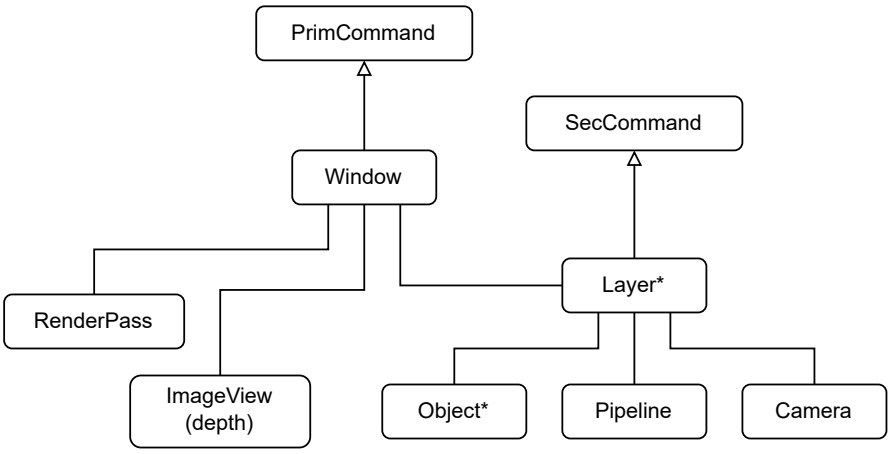
Ke každému meshi je přiřazeno několik vlastností, které pospolu definují, jakým způsobem je světlo rozptýleno od povrchu, který je zadán dle jednotlivých vertexů z meshe. Lze také hovořit, že tyto vlastnosti definují *Materiál*. Musí platit, že model obsahuje ekvivalentní počet meshů a materiálů. Vlastnosti lze dělit podle rozsahu vlivu. Na vlastnosti, které platí pro celý povrch rovnoměrně a na vlastnosti, které jsou definovány pro každý bod povrchu zvlášť.

Právě tyto vlastnosti lze uložit do struktur nazývaných *Textura*. Jedná se o bitmapu o určitých rozměrech, která definuje vlastnosti pro každý bod dle nanesení na povrch. Například se může jednat o barvu, normálu nebo třeba i drsnost, metalicitu a třeba i tzv. *Ambient Occlusion*[11].

4.5.3 Vykreslení modelů

Vykreslení modelů je rozsáhlý proces. Práci jsem si rozdělil do dvou tříd. První třídou je *Window*. Účelem této třídy, jež je derivovaná od třídy *PrimCommand*, je vykreslení množiny *SecCommand* na surface, který vzniká voláním funkce, předané parametrem konstruktoru. Součástí této třídy jsou swapchain, třída *RenderPass*, framebuffery a další Vulkan struktury. Jejich úkolem je vykreslit a prezentovat snímek na vytvořený surface. Navíc třída umožňuje použít MSAA pro vyostření hran. Třída samotná slouží hlavně pro zpracování sekundárních příkazů, které se provedou v rámci primárního příkazu a neslouží pro vykreslení modelů.

K tomu souží druhá třída *Layer*, která je vlastně kontejner pro modely se stejnou grafickou pipeline. Třída je derivovaná od *SecCommand* a je předávána třídě *Window* pro vykreslení. Parametry konstruktoru třídy *Layer* jsou cesty k fyzickým souborům s kompilovanými shadery a strukturou *PipelineInit*. Kromě grafické pipeline třída obsahuje deskriptory, popisující vstupy shaderů, projekční matici, kameru, lokální osvětlení a také množinu obsažených *Modelů*.



Obrázek 4.4: Ukázka struktury, která má za úkol vykreslení modelů.

[11] *Ambient Occlusion* neboli „Okolní okluze“ je technika, která napomáhá zvýšení kvality osvětlení modelu tím, že si předpočítáme pro tento model jeho zastínění a následně tento výsledek využijeme při výpočtu v reálném čase. [10]

4.6 Skybox

Implementace skyboxu spočívala hlavně v načtení cubemap textury a v jejím přenosu na grafickou kartu. Skybox je vykreslován podobným způsobem jako modely, ale využívá se odlišné grafické pipeline a jiných shaderů. Z toho vyplývá, že odlišně od modelů, skybox musí být vykreslován jinou instancí třídy *Layer*. Tedy po načtení textury a vytvoření modelu z předdefinovaných vrcholů, se vytváří model, který je předán k vykreslení instanci třídy *Layer*, která je určena pro vykreslení skyboxu. Tato cubemap textura se taktéž předává modelům jako textura prostředí a slouží k IBL. Nedostatkem tohoto přímého předání textury prostředí je, že při více modelech ve scéně, nebudou součástí této textury a v rámci IBL nebudou reflektovány.

4.7 Vizualizační aplikace

Vizualizační aplikaci, sloužící k vykreslení modelů s patřičnými fyzikálními vlastnostmi, jsem si z hlediska problematiky rozdělil do tří částí.

První částí je okno aplikace, do kterého se vykresluje scéna. K vytváření a správě okna využívám již zmíněnou externí knihovnu GLWF. Část využívá třídu *Window* z knihovny pro vykreslení scény do okna. Její hlavní účel je řízení vykreslení. Stará se také o návratová volání ze vstupu okna. Jedná se o následující čtyři události:

- Framebuffer resize - změna rozlišení okna
- Keyboard - vstup z klávesnice
- Mouse Button - vstup tlačítek na myši
- Cursor position - vstup aktuální pozice kurzoru

Za pomoci návratových volání jsou měněny hodnoty přepínačů této části, podle kterých jsou prováděny adekvátní reakce dle typu události. Například: jedná-li se o událost změny rozlišení, aplikace zajistí předání této informace třídě *Window* z knihovny, aby se mohla přizpůsobit novému rozlišení. V případě ostatních zmíněných událostí, je prováděna změna kamery podle definovaného ovládání dostupné v příloze A. Za speciální případ události lze považovat uzavření okna. V tomto případě má aplikace za úkol přerušit činnost vykreslování, uvolnit prostředky a ukončit se.

Pro snadnější manipulaci s daty, která definují aktuální vykreslovací scénu, bylo vhodné vytvořit strukturu pro uchování a spravování dat o vykreslovací scéně. Součástí této struktury jsou veškeré modely, zdroje osvětlení, skybox, kamera, atd. Hlavním účelem vytvoření struktury bylo umožnění grafickému rozhraní snadnější přístup ke scéně. Za výhodu považuji konstruktor, který inicializuje jednotlivé části scény a připraví je pro vykreslení. K dispozici je taktéž funkce, která na základě úprav scény aktualizuje data, uložená na paměti grafické karty.

Další část se týká grafického rozhraní. O tuto činnost se stará třída, která využívá externí knihovny ImGui pro vykreslení a správu nad grafickým rozhraním. Tato třída je derivovaná od třídy *SecCommand*. Je tedy možné za pomoci třídy *Window*, která je derivovaná od třídy *PrimCommand*, vykreslit grafické rozhraní prostřednictvím knihovny.

Za poslední část považuji funkci main, ve které se nejprve inicializují potřebné třídy z knihovny a připraví se zmíněná struktura s daty o scéně. Následně v hlavním while cyklu probíhá vykreslování. Podmínkou cyklu je boolean hodnota získaná od části, která se stará

o okno. Změně této proměnné předchází událost uzavření okna. V cyklu se nejprve oznámí grafickému rozhraní, že započalo vykreslení nového snímku. Poté je možné zkontrolovat, zda-li se uživatelský vstup týkal grafického rozhraní, nebo ovládání kamery aplikace. Na základě vstupu je možné změnit vlastnosti nebo objekty ze scény. Dále ve smyčce dochází k aktualizaci scény a poté se aktualizuje grafické rozhraní. A jako poslední funkce se oznámí třídě *Window*, že může začít vykreslovat snímek na zadaný surface. Pseudokód funkce main s inicializací a vykreslovacím cyklem z vizualizační aplikace vypadá následně:

Algoritmus 4: Pseudokód funkce main

```
1  inicializace renderovacího jádra
2  inicializace okna
3  inicializace dat scény
4  inicializace grafického rozhraní
5  while okno nemá být uzavřeno do
6      započetí nového snímku
7      aktualizace dat scény na základě vstupu
8      aktualizace GUI
9      vykreslení snímku
10 end while
11 dealokace potřebných dat
```

Kapitola 5

Možná rozšíření

Během procesu tvorby mé práce jsem se setkával s nedostatky a s možnostmi rozšíření, které sice z hlediska zadání nebyli relevantní, avšak lze nad nimi uvažovat, jako nad možnostmi pokračování této práce.

5.1 Vylepšení knihovny

Co se týče knihovny, za hlavní nedostatek považuji její čiré zaměření na vykreslení physically-based modelů. To se týká hlavně struktury modelů. V budoucnu by se knihovna mohla vylepšit a udělat třídu abstraktní, aby si uživatel mohl definovat své vlastní struktury modelů, včetně informací posílaným do shaderů. Jako další rozšíření knihovny mě napadá, vylepšení načítání modelu a textur pro širší podporu i jiných formátů. Například formátů FBX a OBJ.

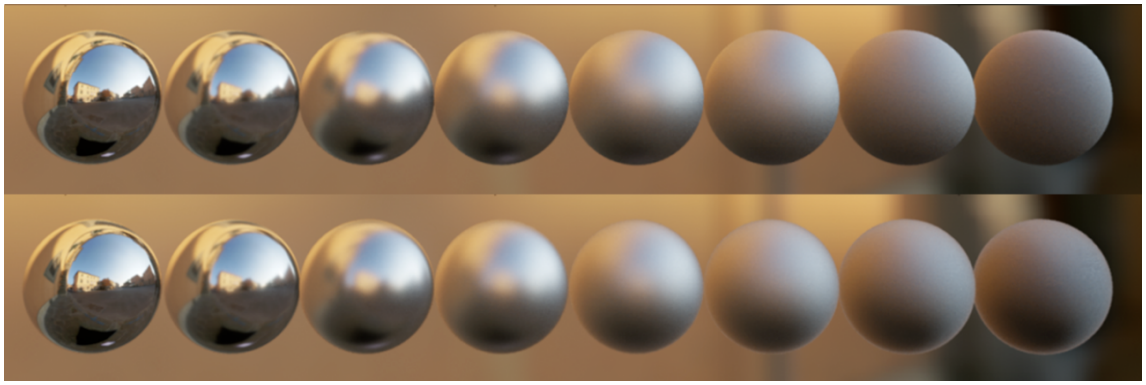
5.2 Optimalizace IBL

IBL je hodně náročnou technikou pro výpočetní zdroje a v aktuálním implementovaném stavu lze v reálném čase pozorovat jednotky modelů s uspokojivou efektivitou. Při vykreslení složitějších modelů, může tento výkon být nedostačující. Z toho důvodu by bylo vhodné osvětlovací model urychlit za mírnou cenu kvality.

Jak jsem se zmínil v kapitole 2.5.3, vzorec 2.19, lze dále aproximovat do dvou sum, které se mohou předvypočítat a dojde tak ušetření hardwaru. Rovnice pak vypadá následně [7]:

$$\frac{1}{N}\sum_{k=1}^{N}\frac{L_i(l_k)f(l_k,v)cos\theta_{l_k}}{p(l_k,v)} \approx \left(\frac{1}{N}\sum_{k=1}^{N}L_i(l_k)\right)\left(\frac{1}{N}\sum_{k=1}^{N}\frac{f(l_k,v)cos\theta_{l_k}}{p(l_k,v)}\right) \tag{5.1}$$

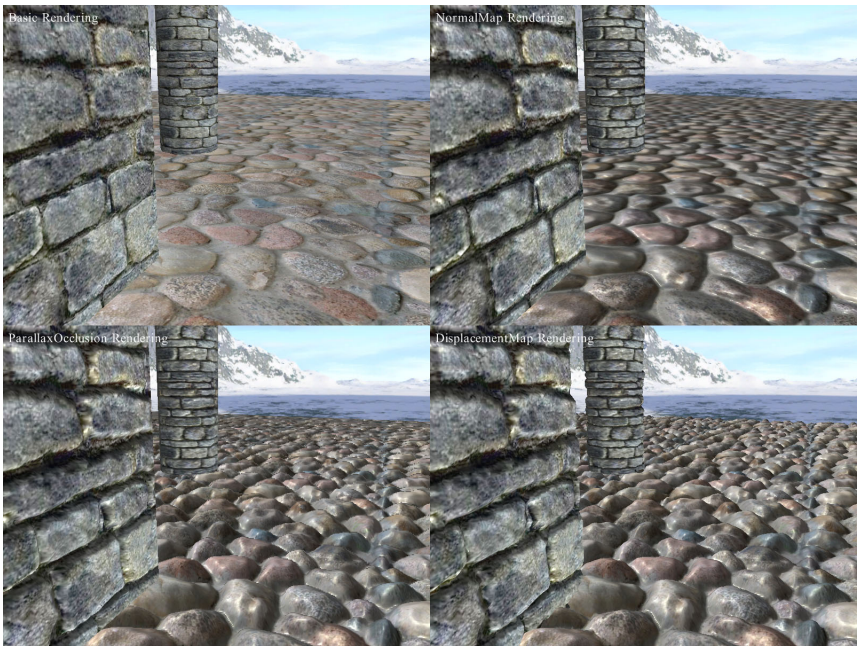
Jak je možné vidět na následujícím obrázku 5.1, rozdíl je téměř nepoznatelný a dojde k výraznému snížení nároků na výpočetní výkon. Zda-li implementovat tuto optimalizaci, záleží čistě na zvolených prioritách. Ve většině real-time aplikacích s IBL se upřednostňuje plynulost vykreslení, a proto se více využívá tato optimalizace. Na druhou stranu například filmová studio preferují kvalitu aproximace. Čas vykreslení snímku pro tyto případy není prioritou.



Obrázek 5.1: Porovnání vlastností IBL bez a s předvypočítanými hodnotami pro různé drsnosti povrchu [7].

5.3 Pokročilejší vykreslení povrchových nerovností

Mezitím, co se normal mapping ze vzdálených pohledů kolmých k ploše může zdát dostačující, při pohledu zblízka, nebo pod ostrým úhlem, můžeme spatřit nedokonalosti této techniky. Proto by bylo vhodné implementovat navíc parallax mapping nebo dokonce displacement mapping. Je možné spatřit na obrázku 5.2, rozdíl mezi těmito technikami a využitým normal mapping je dost razantní. Zvláště v případě PBR, které je v kombinaci s technikami pokročilejšího vykreslení nerovností schopno přesvědčivě aproximovat odraz světelných paprsků od hrbolatých povrchů.



Obrázek 5.2: Porovnání technik vykreslení nerovností: Vlevo nahoře se nachází vykreslení bez techniky nerovností, vpravo nahoře normal mapping, vlevo dole parallax occlusion a vpravo dole displacement mapping[2].

[2]Obrázek převzat z webové stránky [navštíveno 04.05.2021]: https://warosu.org/3/thread/397522

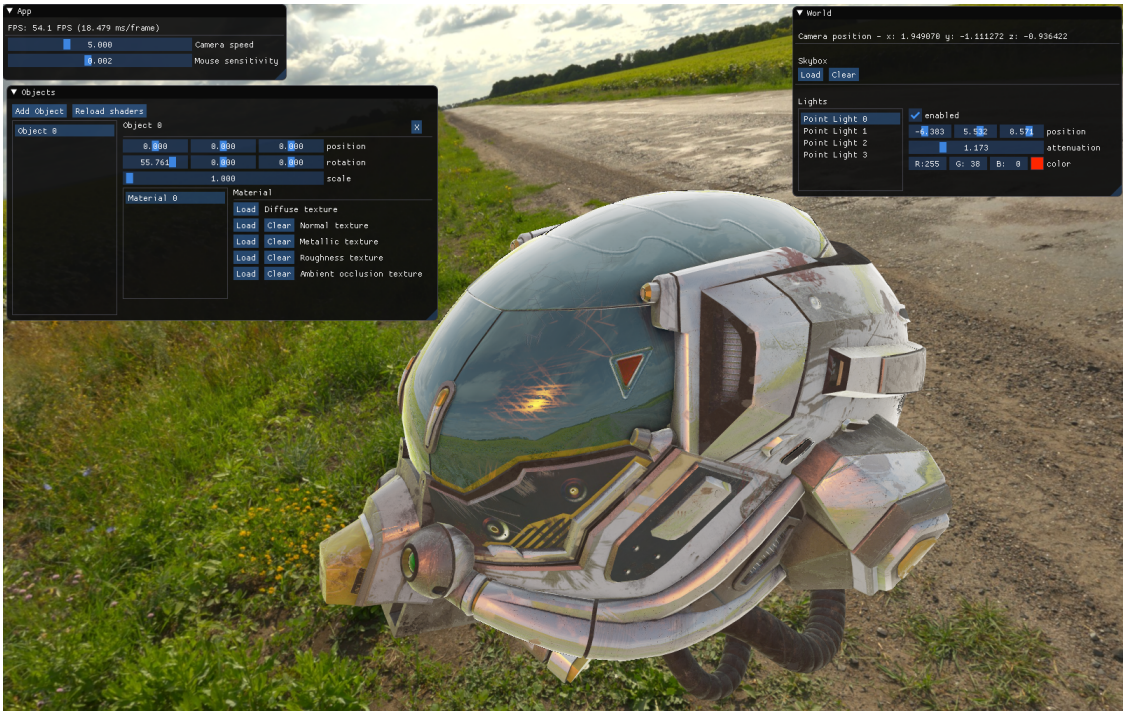
Kapitola 6

Závěr

Cílem bakalářské práce bylo vytvoření knihovny realizující PBR a následně demonstrační aplikace, která tuto knihovnu využívá. Dalším úkolem bylo zhotovení krátkého videa, zobrazujícího vypracovanou PBR aplikaci. Uvedené lze najít na přiloženém médiu.

Pro dosažení těchto cílů jsem v první řadě musel nastudovat techniky PBR. Během studia literatury PBS jsem se dozvěděl mnoho informací o vzorci odrazu, funkci BRDF včetně jejích částí, Lambertovy funkce odrazu a také o různých modelech osvětlení. Seznámil jsem se také s teorií skyboxu a povrchových nerovností.

Na základě získaných vědomostí jsem následně zpracoval návrh, v němž jsem si nejdříve stanovil požadované funkcionality a požadavky programu. Pokračováním byla funkce BRDF včetně jejich součástí. Osvětlovací model byl další částí návrhu a věnoval jsem se mu po zvolení BRDF funkce. Vybral jsem osvětlení vhodná ke zvoleným potřebám. Po vybrání



Obrázek 6.1: Ukázka aplikace vykreslující model helmy v texturovaném prostředí s jedním zdrojem bodového světla.

technik z kapitoly 2 jsem se zaměřil na návrh knihovny a samotné demonstrační aplikace. V knihovně jsem si především definoval cíle a podle nich jsem knihovnu rozdělil na dvě části. Následně jsem se soustředil na vizualizační aplikaci a na její grafické rozhraní.

Poté jsem své úsilí přesunul na implementaci. Zde jsem nejprve popsal využité technologie. Dále, dle návrhové kapitoly 3, jsem implementoval funkci BRDF, osvětlovací model a techniku normal mapping. Uvedl jsem zde i některé pseudokódy a kódy funkcí, které jsem implementoval. Závěrem došlo i na knihovnu, skybox a vizualizační aplikaci, kde se pro lepší představu nacházejí různé diagramy.

Během implementace a testování jsem se setkal s některými nedostatky nebo možnostmi vylepšení, které sice neovlivňují požadavky z hlediska zadání, ale mohou být zajímavým námětem pro vylepšení nebo i pokračovaní v této práci. O těchto námětech pojednávala kapitola 5. Výsledek tohoto procesu je možné spatřit na obrázku 6.1, kde se vyskytuje vizualizační aplikace, která za pomoci knihovny vykresluje model s fyzikálními vlastnostmi.

Literatura

[1] Blinn, J. F. Models of Light Reflection for Computer Synthesized Pictures. *SIGGRAPH Comput. Graph.* červenec 1977, sv. 11, č. 2, s. 192–198. ISSN 0097-8930.

[2] Blinn, J. F. Simulation of wrinkled surfaces. In: Chasen, S. H. a Phillips, R. L., ed. *SIGGRAPH*. ACM, 1978, s. 286–292. Dostupné z: http://dblp.uni-trier.de/db/conf/siggraph/siggraph1978.html#Blinn78a.

[3] Cook, R. L. Shade trees. In: Christiansen, H., ed. *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1984, Minneapolis, Minnesota, USA, July 23-27, 1984*. ACM, 1984, s. 223–231. Dostupné z: https://doi.org/10.1145/800031.808602.

[4] Cook, R. L. a Torrance, K. E. A Reflectance Model for Computer Graphics. *SIGGRAPH Comput. Graph.* srpen 1981, sv. 15, č. 3, s. 307–316. ISSN 0097-8930.

[5] Greene, N. Environment Mapping and Other Applications of World Projections. *Computer Graphics and Applications, IEEE*. listopad 1986, sv. 6, č. 11, s. 21–29. ISSN 0272-1716.

[6] Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y. et al. Detailed shape representation with parallax mapping. In: *Proceedings of ICAT*. 2001, sv. 2001, s. 205–208.

[7] Karis, B. *Real Shading in Unreal Engine 4*. Epic Games, 2013. Dostupné z: http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf.

[8] Lazarov, D. *Physically Based Lighting in Call of Duty: Black Ops* [online]. 2011. část z „Advancesin Real-Time Rendering in 3D Graphics and Games“, SIGGRAPH 2011 Course Notes, Naposledy navštíveno: 2021–04-07. Dostupné z: http://advances.realtimerendering.com/s2011/.

[9] McAuley, S., Hill, S., Hoffman, N., Gotanda, Y., Smits, B. et al. Practical Physically-based Shading in Film and Game Production. In: *ACM SIGGRAPH 2012 Courses*. New York, NY, USA: ACM, 2012, s. 10:1–10:7. SIGGRAPH '12. ISBN 978-1-4503-1678-1.

[10] Möller, T. *Real-time rendering*. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2018. ISBN 978-1-1386-2700-0.

[11] Phong, B. T. Illumination for Computer Generated Pictures. *Commun. ACM*. 1975, sv. 18, č. 6, s. 311–317. Dostupné z: http://dblp.uni-trier.de/db/journals/cacm/cacm18.html#Phong75.

[12] Schlick, C. An Inexpensive BRDF Model for Physically-based Rendering. *Computer Graphics Forum*. Blackwell Science Ltd. 1994, sv. 13, č. 3, s. 233–246. ISSN 1467-8659.

[13] Schretter, C., Kobbelt, L. a Dehaye, P.-O. Golden Ratio Sequences for Low-Discrepancy Sampling. *J. Graphics Tools*. IEEE Computer Society. 2012, sv. 16, č. 2, s. 95–104. Dostupné z: http://dblp.uni-trier.de/db/journals/jgtools/jgtools16.html#SchretterKD12.

[14] Walter, B., Marschner, S. R., Li, H. a Torrance, K. E. Microfacet Models for Refraction Through Rough Surfaces. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, s. 195–206. ISBN 978-3-905673-52-4.

Příloha A

Ovládání aplikace

Klávesnice je využita společně s myší pro ovládání kamery pro pohyb a změnu orientace v prostoru. V následující tabulce A.1 se nachází všechny možnosti ovládání aplikace.

Klávesa	**Reakce**
W	Pohyb kamery dopředu
S	Pohyb kamery dozadu
A	Pohyb kamery doleva
D	Pohyb kamery doprava
Levé tlačítko myši + posun myší	Změna orientace kamery

Tabulka A.1: Tabulka ovládání aplikace.