
**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**


# GRAFICKÉ INTRO 64KB S POUŽITÍM OPENGL

**GRAPHICS INTRO 64KB WITH THE USE OF OPENGL**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** **JAN OLEXA**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Ing. TOMÁŠ MILET**
**SUPERVISOR**



**BRNO 2019**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

# Zadání diplomové práce


Student: **Olexa Jan, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Grafické intro 64kB s použitím OpenGL**
**Graphics Intro 64kB Using OpenGL**
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s fenoménem grafického intra s omezenou velikostí.
2. Prostudujte knihovnu OpenGL a její nadstavby.
3. Popište vybrané techniky použitelné v grafickém intru s omezenou velikostí.
4. Implementujte grafické intro s použitím OpenGL, aby velikost spustitelné verze nepřesáhla 64kB.
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte video pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3, experimenty směřující k vyřešení bodu 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 6. listopadu 2018

# Abstrakt

Cílem práce je vytvořit grafické intro využívající knihovnu OpenGL jehož spustitelný soubor má maximální velikost 64 kB. Program je psán v jazyce C++ a využívá OpenGL verze 4.6.



# Abstract

The goal of this thesis is to create a graphics intro using the OpenGL library, where the maximal size of the executable is 64 kB. The program is written in C++ language and uses OpenGL version 4.6.



# Klíčová slova

grafika, OpenGL, intro 64kB, C++, PBR, kosterní animace



# Keywords

graphics, OpenGL, intro 64kB, C++, PBR, skeletal animation



# Citace

OLEXA, Jan. *Grafické intro 64kB s použitím OpenGL*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

# Grafické intro 64kB s použitím OpenGL

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Tomáše Mileta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Olexa
21. května 2019

## Poděkování

Rád bych poděkoval panu inženýru Tomáši Miletovi za jeho přístup k vedení diplomové práce a snahu mi pomoci překonat různé překážky, na které jsem při jejím tvoření narazil.

# Obsah

# Kapitola 1

# Úvod

Intrem 64 kB je myšlen program generující spustitelný soubor o maximální velikosti 64 kB který generuje grafický výstup. Jedná se v podstatě o programátorskou výzvu, případně i organizovanou soutěž, jejíž cílem je předvést co nejvíce funkcionality a co nejlepší vizuální výstup v takto omezeném prostoru, který nutí programátora pracovat bez různých dodatečných knihoven a místo toho jejich funkcionalitu implementovat sám. Jelikož jsou zakázány i různé externí soubory – povolen je jen a pouze onen spustitelný soubor – nelze využívat ani obrázkových souborů a programátor si musí vše sám i vygenerovat, tedy musí projevit i trochu estetického cítění pokud chce, aby jím vytvořené textury vypadaly dobře.

Ve zkratce se dá říci, že vytvoření intra o velikosti 64 kB je především ukázkou schopností programátora a jeho chápání daného prostředí bez potřeby různých dodatečných nadstaveb.

Tématem intra této práce je pak vytvoření modelu robota, jeho zasazení do místnosti ústící do chodby a následné rozpohybování pomocí kosterní animace. Dále je cílem implementování osvětlujícího modelu na bázi technik pro physically based rendering a generování náležitých textur.

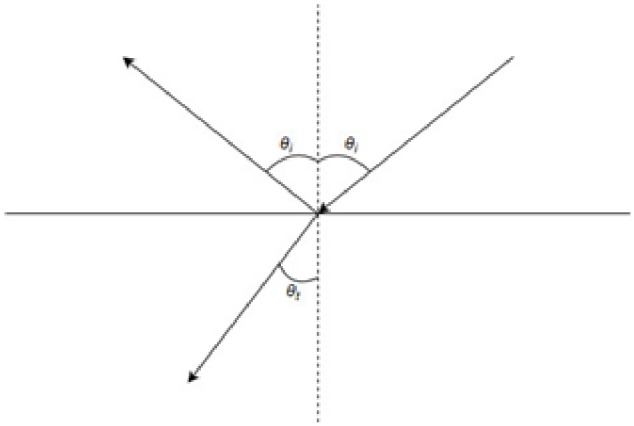
# Kapitola 2

# Teorie

## 2.1 Physically based rendering

Physically based rendering (česky *rendering založený na fyzice*, dále jen *PBR*) je technika v oblasti počítačové grafiky zabývající se interakcí světla s hmotou. Jak již název napovídá, hlavní tezí tohoto přístupu je pak snaha použití fyzikálních zákonů a vlastností reálného světa k vytvoření realisticky vypadajícího osvětlovacího modelu.

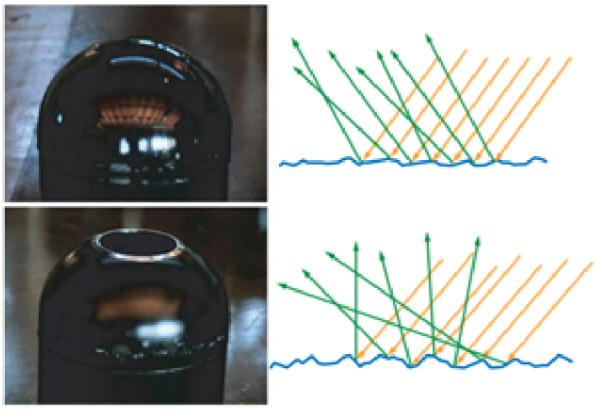
Při interakci světla s hmotou dochází k odrazu a k lomu světla, ilustrovaných na obrázku 2.1, kde lze vidět, že úhel odrazu světelného paprsku je roven úhlu dopadu. Důležité přitom je, že je plocha dopadu perfektně rovná – dalo by se namítnout, že v reálném světě není perfektně rovného nic (přinejmenším na úrovni atomů, které nikdy nebudou perfektně zarovnané), avšak je třeba mít na paměti, že viditelné světlo má vlnovou délku v řádu stovek nanometrů, což je velikost, v jaké se perfektně rovné materiály skutečně vyrobit dají (nerovnosti v řádech jednotek nanometrů nebudou mít na dopadající paprsek vliv). Hovoří se pak o *opticky rovných* materiálech, které lze obvykle nalézt v teleskopech a jiných vysoce kvalitních optických nástrojích.




Obrázek 2.1: Úhel dopadu se rovná úhlu odrazu, nikoli však úhlu lomu.

V reálném světě se však ani s takovými materiály člověk příliš často nesetká a to ani u zdánlivě velmi hladkých povrchů – pokud je takový povrch prozkoumán pod mikroskopem, obvykle lze pozorovat, že je různě zvrásněn a pokryt malými defekty které jsou lidským okem nerozlišitelné, avšak mají vliv na chování dopadajícího světla. Hrubší povrchy jsou

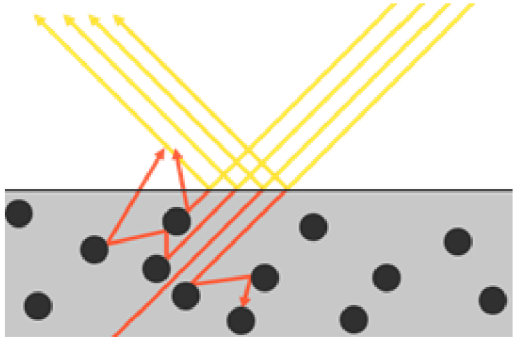
pak samozřejmě zvrásněny více než ty hladší, jak je možno vidět na obrázku 2.2. Tyto nerovnosti však, při dostatečném přiblížení, lze považovat za malé opticky rovné plošky, které se při interakci se světlem chovají stejně jako perfektně rovný povrch. Jediný rozdíl mezi hladšími a hrubšími povrchy je tedy ten, že jejich opticky rovné plošky jsou orientovány různými směry a odražené světlo je tak různou mírou roztříštěno. Čím více pak má povrch různých nepravidelností (a tedy čím více je odražené světlo roztříštěno), tím matněji se lidskému oku jeví.



Obrázek 2.2: Ačkoli jsou oba povrchy zdánlivě hladké, interagují se světlem velmi rozdílně. Za povšimnutí stojí jak nepravidelné jsou odrazy paprsků u hrubšího povrchu oproti hladšímu – právě ona roztříštěnost odchozích paprsků způsobuje vyšší matnost odrazů. Obrázek získán z: https://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf

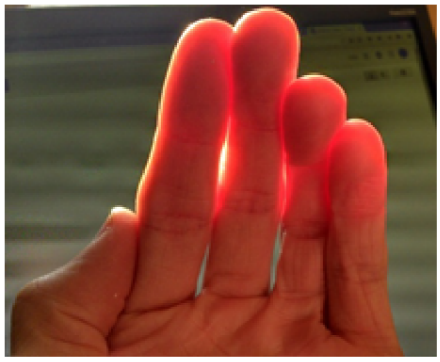
Vzhledem k tomu, že se tyto nerovnosti nacházejí v řádu mikrometrů, nedává samozřejmě smysl je detailně modelovat – plně postačí jejich statistika, která bude udávat kolik světla daný povrch odrazí v očekávaném směru a kolik ho bude roztříštěno do směrů okolních.

Je tedy zřejmé co se při interakci světla s hmotou stane se světlem odraženým, avšak co světlo lomené? V případě kovů je takovéto světlo okamžitě absorbováno prostřednictvím volných elektronů a není s ním tedy třeba dále operovat, avšak u mnoha izolantů se světlo dále odráží od částic uvnitř materiálu, až je nakonec vyzářeno tím stejným povrchem, ale na jiném místě a pod jiným úhlem. Situace je ilustrována na obrázku 2.3. Vyvstává tak další důležitý parametr PBR – mimo uvažování o rovnosti povrchu je také třeba brát v potaz zdali je povrch kovový, tedy jestli bude lomené světlo absorbováno, nebo jestli je třeba s ním v modelu nadále počítat.



Obrázek 2.3: Část světla pronikne dovnitř materiálu, kde se odrazí od jednotlivých částic. Obrázek získán z: https://learnopengl.com/img/pbr/surface_reaction.png

Význam i takto vyzářeného světla je nezanedbatelný a pozorovatelný i v reálném světě, jak je ilustrováno na obrázku 2.4.



Obrázek 2.4: Část světla proniknuvšího do pokožky je opět vyzářena, avšak na jiném místě než dopadl původní paprsek. Obrázek získán z: https://en.wikipedia.org/wiki/Subsurface_scattering#/media/File:Skin_Subsurface_Scattering.jpg

V případě zpět vyzářeného lomeného světla pak mohou nastat dvě situace. Při první z nich je světlo vyzářeno v takové blízkosti bodu dopadu, že se stále nachází v rámci stejného pixelu při výsledném renderingu počítačem. Vzdálenost od bodu dopadu tak může být ignorována a celá problematika být řešena lokálně – světlo vyzářené či odražené v daném bodě bude závislé pouze na světle dopadajícím do onoho bodu. Druhá situace nastává, pokud je světlo vyzářeno zpět ve vzdálenosti větší než jeden pixel – v takovém případě bude světlo v daném bodě závislé i na světle dopadajícím do bodů okolních. K modelování tohoto jevu je zapotřebí specializovaných renderingových technik (technik *subsurface scattering*), avšak k získání podobných výsledků lze využít i „obyčejné“ techniky pro rendering rozptýleného světla, případně tyto nelokální případy zcela ignorovat za cenu poněkud horšího výstupu. Více o základních principech PBR a o interakci světla s hmotou je možné si přečíst v práci Natty Hoffmanové [6].

### 2.1.1 Bidirectional reflectance distribution function

Bidirectional reflectance distribution function (volně přeloženo *obousměrná distribuční funkce odrazu*, dále jen *BRDF*) je funkce udávající množství světla, které je ze všech příchozích

paprsků odraženo směrem ke kameře (paprsky odražené někam jinam by nebyly pozorovatelné a tedy mohou být ignorovány) pro daný bod materiálu. BRDF je pak jednou z komponent odrazové rovnice (2.1):

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

Tedy odchozí světlo ($L_o$) je rovno integrálu (přes všechny směry nad povrchem) BRDF ($f(l, v)$) vynásobené příchozím světlem ($L_i$) a faktorem kosinu úhlu mezi povrchovou normálou a vektorem dopadajícího světla (paprsek dopadající kolmo na plochu bude více lokalizovaný než paprsek dopadající pod větším úhlem). Symbol $l$ značí vektor od bodu dopadu ke zdroji světla, $v$ je vektor od bodu dopadu ke kameře, a $n$ je normála povrchu v bodě dopadu. Integrál pak reprezentuje všechny paprsky přispívající světlem pro daný bod pro danou hemisféru.

Samotnou BRDF pak lze rozdělit na dvě části – spekulární, která popisuje světlo odražené od povrchu, a difuzní, která popisuje světlo zpětně vyzářené povrchem. Způsobů vypočítání BRDF je vícero, tento text se vak zaměří na běžně používanou Cook-Torrance BRDF, jejíž výpočet je definován následující rovnicí (2.2):

$$f(l, v) = k_d f_d(l, v) + k_s f_s(l, v) \tag{2.2}$$

Kde $k_d$ a $k_s$ udávají poměr světla odraženého ku světlu lomenému (jejich součet je vždy roven jedné, jinak by byl porušen zákon o zachování energie) a $f_d$ a $f_s$ udávají difuzní a spekulární část BRDF. Informace o odrazové rovnici a BRDF čerpány z práce Natty Hoffmanové [6] a webových stránek Joey de Vriese [11], kapitola PBR – Theory.

### 2.1.2 Difuzní BRDF

Jak již bylo zmíněno výše, existuje vícero modelů řešících rozptýlené světlo. Zde bude popsán model Lambertovský. Tento model difuzní části BRDF je velice jednoduchý, jelikož se v podstatě jedná o konstantní hodnotu (faktor kosinu je totiž již obsažen v odrazové rovnici, nikoli v samotné BRDF). Samotná rovnice (2.3) pro výpočet difuzní části BRDF pak zní:

$$f_d(l, v) = \frac{c}{\pi} \tag{2.3}$$

Kde $c$ je množství světla vyzářeného zpět z materiálu, což koresponduje s barvou daného povrchu. Pokud je tedy těleso například naprosto černé, $c$ bude nulový trojsložkový vektor – veškeré lomené světlo bylo absorbováno.

Existují i jiné, složitější modely pro výpočet difuzní složky BRDF, které kupříkladu modelují vliv hrubosti povrchu na tento aspekt – pokud je velikost nějaké nerovnosti na povrchu tělesa opravdu velká, může se stát, že světlo bude materiálem putovat déle a bude tedy vyzářeno mimo rámec pixelu, do kterého původně dopadlo, což už může mít na výsledek u některých materiálů vliv. Lambertův model je však, možná právě díky své jednoduchosti, nejčastěji užívanou variantou.

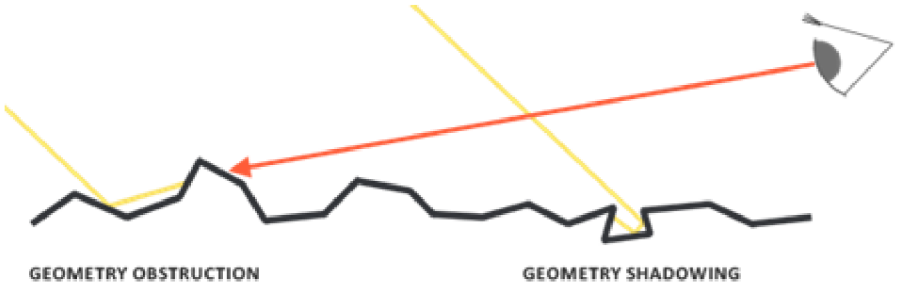
Informace o difuzní složce BRDF čerpány z článku Joey de Vriese [11], kapitola PBR – Theory.

### 2.1.3 Spekulární BRDF

Matematicky výrazně zajímavější než difuzní část BRDF je pak část spekulární. Její rovnice (2.4) je:

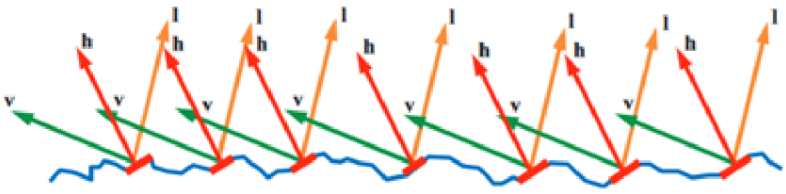
$$f_s(l,v) = \frac{F(l,h)G(l,v,h)D(h)}{4(n \cdot l)(n \cdot v)} \tag{2.4}$$

K pochopení rovnice je třeba se na chvíli vrátit zpět k mikrogeometrii. Na obrázku 2.5 lze vidět, že na mikroskopické úrovni povrchu mohou nastat jevy, které zabrání světlu být pozorováno přestože ho daná mikroploška odráží směrem k pozorovateli. Důvodem může být zablokování paprsku jinou nerovností povrchu kvůli které na mikroplošku pozorovatel nevidí, nebo je ploška nerovností zastíněna a paprsek se k ní nedostane, případně se paprsek o takovouto překážku odrazí jiným směrem – světlo které je tímto způsobem „ztraceno“ bude v modelu ignorováno, avšak je zapotřebí tyto ztráty zahrnout do statistiky. Dále jsou uvažovány pouze první odrazy takovéhoto světla – ačkoli by se možná některé paprsky po mnoha odrazech k pozorovateli dostaly, tak jsou ignorovány pokud se k němu nedostanou už po odrazu prvním.




Obrázek 2.5: Světlo může být ztraceno různými způsoby. Obrázek získán z: https://learnopengl.com/img/pbr/geometry_shadowing.png

Další vlastností kterou v oblasti mikrogeometrie lze pozorovat je, že se směrem k pozorovateli odrazí pouze paprsky dopadající na plošky které jsou orientované tím správným směrem, tedy plošky které mají určitou normálu – jelikož na všechny plošky dopadá paprsek pod stejným úhlem (v rámci mikrogeometrie jsou vzdálenosti uvnitř zpracovávaného pixelu tak malé, že úhel prakticky neovlivní), tak logicky všechny plošky orientované směrem jiným odrazí dopadající paprsky jinam a tedy budou ignorovány. Tato normála bude vždy právě na půli cesty mezi vektory dopadu a odrazu (vektor odrazu musí vést k pozorovateli) a je značena $h$, jak je možno si prohlédnout na obrázku 2.6.




Obrázek 2.6: Mikroplošky odrážející světlo k pozorovateli budou mít všechny stejnou normálu. Obrázek získán z: https://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf

$D(h)$ v BRDF rovnici značí normálovou distribuční funkci, tedy koncentraci mikroplošek v daném bodě, které mají tu správnou normálu k odrazu světla k pozorovateli ($h$ značí onu "správnou"normálu). $G(l, v, h)$ je geometrická funkce udávající kolik procent takto orientovaných mikroplošek není zastíněno a nimiž odražené světlo není nijak blokováno nerovnostmi povrchu a $F(l, h)$ je Fresnelova funkce udávající množství světla které bude odraženo oproti množství které bude lomeno. Dohromady je tedy získána informace o tom, kolik světla bude v daném bodě odraženo směrem k pozorovateli a skutečně se k němu i dostane. Jmenovatel $4(n \cdot l)(n \cdot v)$ je pak korekčním faktorem. Zbývá pouze otázka jak jednotlivé funkce implementovat.

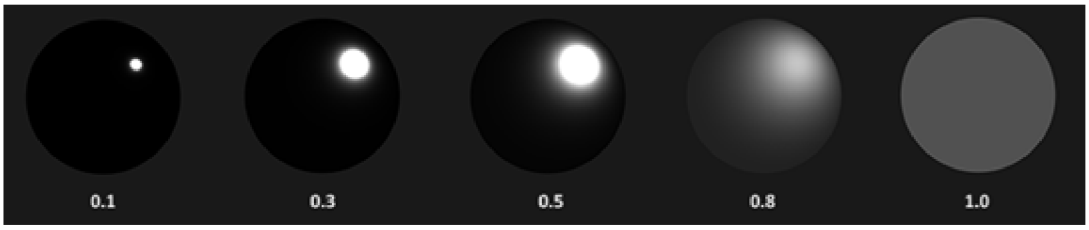
Informace o spekulární složce BRDF, normálové distribuční funkci, geometrické funkci a Fresnelově funkci jsou čerpány z práce Natty Hoffmanové [6], webových stránek Joey de Vriese [11], kapitola PBR – Theory a článku Marco Alamia [1].

### 2.1.4 Normálová distribuční funkce

Jak již bylo zmíněno v sekci výše, normálová distribuční funkce popisuje množství mikroplošek orientovaných tak, že příchozí světlo odrážejí směrem k pozorovateli. Způsobů implementace existuje více, zde bude rozebrána Trowbridge-Reitzova GGX (2.5), která je definována následovně:

$$D(n, h, \alpha) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2} \tag{2.5}$$

Symbol $\alpha$ v rovnici značí hrubost povrchu pohybující se mezi hodnotami 0 a 1. Na obrázku 2.7 pak lze pozorovat efekt tohoto výpočtu pro materiály s různou hrubostí – povrchy které mají hrubost nízkou mají spekulární odlesk koncentrován na malé ploše a s velkou intenzitou světla, zatímco povrchy hrubší budou mít odlesk o menší intenzitě a větší ploše. Důvod, proč tomu tak je, začne být jasný při vybavení si dříve zmíněných poznatků o mikrogeometrii – na pravidelném, hladkém povrchu lze najít mikroplošky orientované tak, aby odrážely světlo směrem k pozorovateli pouze na určitém, relativně malém úseku, zatímco úseky jiné budou světlo odrážet jinam. Hrubé povrchy, naproti tomu, mají mnoho různě orientovaných mikroplošek a ty které světlo odrážejí směrem k pozorovateli tak nalezneme na mnohem větší ploše, avšak v nižší koncentraci (jelikož tam opět budou plošky odrážející světlo jiným směrem).




Obrázek 2.7: Čím vyšší hrubost, tím širší odlesk, ale i nižší intenzita. Čísla pod koulemi značí míru hrubosti. Obrázek získán z: https://learnopengl.com/img/pbr/ndf.png

### 2.1.5 Geometrická funkce

Geometrická funkce vrací hodnotu v rozmezí 0 až 1 a udává pravděpodobnost, že mikroploška odrážející světlo směrem k pozorovateli toto světlo skutečně odrazí (není tedy

zastíněna) a že se tento odražený paprsek skutečně k pozorovateli dostane (není cestou zablokován nějakou jinou mikroskopickou nerovností). Opět existuje více implementací, zde bude uvedena Schlickova GGX (2.6):

$$G_{Schlick}(n, v, k) = \frac{n \cdot v}{(n \cdot v)(1 - k) + k} \tag{2.6}$$

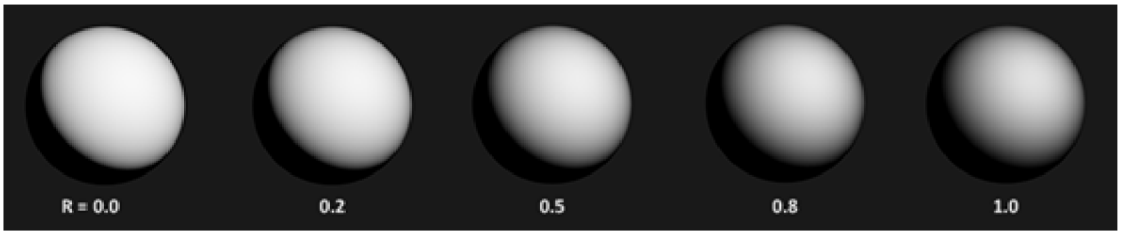
Kde $k$ je definováno následující rovnicí (2.7):

$$k = \frac{(\alpha + 1)^2}{8} \tag{2.7}$$

Dále je pak použita Smithova metoda (2.8), která využívá Schlickovy GGX a řeší jak mikroplošky zastíněné tak blokované (je tedy brán v potaz vektor dopadu a vektor odrazu):

$$G(n, v, l, k) = G_{Schlick}(n, v, k)G_{Schlick}(n, l, k) \tag{2.8}$$

Efekt této funkce v závislosti na hrubosti je možné si prohlédnout na obrázku 2.8.




Obrázek 2.8: Čím více je plošek zastíněných či jejich odrazů blokovaných, tím menší bude intenzita světla. $R$ značí hrubost. Obrázek získán z: https://learnopengl.com/img/pbr/geometry.png

### 2.1.6 Fresnelova funkce

Fresnelova funkce udává poměr světla odraženého od povrchu oproti světlu lomenému, přičemž tento poměr závisí na úhlu pod kterým je na povrch nahlíženo. Takzvaný Fresnelův efekt může být pozorován i v běžném životě – pokud se na nějaký povrch (například na vodní hladinu) člověk podívá pod malým úhlem, velmi se leskne. Když se na povrch podívá kolmo dolů, pak je lesku naopak mnohem méně (u vodní hladiny by bylo vidět dno). Na rozdíl od geometrické a normálové distributivní funkce nevrací Fresnelova funkce jednu hodnotu, nýbrž tři – konkrétně trojici reprezentující červenou, zelenou, a modrou složku světla. Toto rozdělení je nutné, jelikož kovy neodrážejí světlo stejné barvy, nýbrž trochu zbarvené. Samotná Fresnelova rovnice je pak poměrně složitá, avšak existuje vcelku jednoduchá aproximace, takzvaná Fresnel-Schlikova aproximace (2.9):

$$F(h, v, F_0) = F_0 + (1 - F_0)(1 - (h \cdot v))^5 \tag{2.9}$$

Parametr $F_0$ zde udává základní odrazivost povrchu, což je vlastnost materiálu. Tato vlastnost se však drasticky liší pro kovy a ostatní materiály, jak lze vidět v tabulce na obrázku 2.9 – zatímco ostatní materiály se pohybují v hodnotách blízkých nule, které jsou navíc stejné pro všechny barevné složky, kovy jsou naopak blízké jedničce s různými hodnotami pro své jednotlivé barevné složky. Do problematiky se tedy přidává nový parametr – kovovost. Tento parametr se pohybuje mezi hodnotami 0 a 1, což se může zdát podivné, jelikož

materiál samozřejmě buďto kovem je, nebo není a nemůže být něčím mezi, avšak v úvahu je třeba brát různé částečky prachu či písku usazené na povrchu, škrábance a podobné defekty, které lze simulovat právě snížením kovovosti u kovových povrchů a tím dosažení vizuálně lepšího výsledku.

| Material | $F_0$ (Linear) | $F_0$ (sRGB) | Color |
|---|---|---|---|
| Water | 0.02,0.02,0.02 | 0.15,0.15,0.15 | |
| Plastic / Glass (Low) | 0.03,0.03,0.03 | 0.21,0.21,0.21 | |
| Plastic High | 0.05,0.05,0.05 | 0.24,0.24,0.24 | |
| Glass (High) / Ruby | 0.08,0.08,0.08 | 0.31,0.31,0.31 | |
| Diamond | 0.17,0.17,0.17 | 0.45,0.45,0.45 | |
| Iron | 0.56,0.57,0.58 | 0.77,0.78,0.78 | |
| Copper | 0.95,0.64,0.54 | 0.98,0.82,0.76 | |
| Gold | 1.00,0.71,0.29 | 1.00,0.86,0.57 | |
| Aluminum | 0.91,0.92,0.92 | 0.96,0.96,0.97 | |
| Silver | 0.95,0.93,0.88 | 0.98,0.97,0.95 | |

Obrázek 2.9: Běžné materiály mají $F_0$ blízké nule, zatímco kovy ho mají blízké jedničce. Jednou z mála výjimek jsou drahé kameny, krystaly, polovodiče a jiné neobvyklé materiály, jejichž hodnoty jsou kdesi mezi oběma skupinami. Obrázek získán z: https://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf

Definování přesné hodnoty $F_0$ pro každý materiál by bylo zdlouhavé a pracné, avšak lze poměrně jednoduše aproximovat – pro nekovové materiály je zavedena hodnota 0,04, kterou má většina těchto materiálů a která produkuje relativně realistické výsledky. Pro kovy je pak využit parametr kovovosti k interpolaci mezi touto hodnotou a barvou povrchu (obvykle textury) – jelikož kovy veškeré lomené světlo absorbují, tak nemají difuzní složku světla a lze tedy použít jejich barvu přímo jako hodnotu základní odrazivosti. Zároveň je tím vyřešen problém různých barevných složek majících různé $F_0$ – v tabulce na obrázku 2.9 lze vidět, že kovové materiály zbarvují světlo do stejné barvy, jakou mají ony samy, tedy to samé co je činěno touto aproximací.

Je dobré připomenout, že z Fresnelovy funkce je vlastně získána ještě jedna hodnota, a to sice $k_s$, která udává množství světla odraženého oproti světlu lomenému reprezentovanému proměnnou $k_d$, tedy naprosto to samé, co udává Fresnelova funkce. Lze tedy napsat rovnici (2.10):

$$k_s = F(v, h, F_0) \tag{2.10}$$

Díky tomu je známa i $k_d$, díky vztahu (2.11):

$$k_d = 1 - k_s \tag{2.11}$$

Experimentálním dosazením maximální kovovosti do Fresnelovy rovnice (tedy $F_0 = 1$) pak lze demonstrovat, že $k_s$ vyjde 1 a $k_d$ tedy bude 0, tedy povrch nebude mít difuzní složku, což je přesně to, co je od kovového materiálu očekáváno. Jelikož je $k_s$ již obsaženo ve výpočtu spekulární BRDF, nebude tato hodnota již znova použita v obecné BRDF. Veškeré funkce a proměnné obsažené v Cook-Torranceově BRDF jsou nyní známy.

### 2.1.7 Příchozí světlo

Jak vypočítat BRDF, kde povrch interaguje s příchozím světlem, již bylo rozebráno v sekcích výše, avšak dosud nebylo zmíněno příchozí světlo samotné.

Častým způsobem modelování scény je rozmístění bodových zdrojů světla – tedy zdrojů majících nulovou velikost a šířících světlo do všech směrů. Ve skutečnosti samozřejmě něco takového nemůže existovat – nic nemá nulovou velikost – avšak výsledky tohoto přístupu jsou poměrně dobré a celá implementace je relativně jednoduchá. S takovýmto přístupem je tak možné se často setkat v oblasti počítačových her a i jiných grafických vizualizací.

Jelikož světelné paprsky vycházejí z jednoho jediného bodu, bude do každého bodu povrchu dopadat právě jeden paprsek z každého takového zdroje – velmi snadno se tak lze v odrazové rovnici vypořádat s integrálem. Pro každý z bodových světelných zdrojů je vypočítáno tělo integrálu a výsledky jsou zkrátka sečteny.

Dalším krokem je pak modifikace světla samotného pomocí atenuace, tedy jeho tlumení na základě vzdálenosti. Bez tohoto tlumení místo kilometr vzdálené od zdroje by bylo osvětlené stejnou měrou jako místo které je od něj centimetr, což je samozřejmě špatně. S narůstající vzdáleností od zdroje světla totiž klesá jeho koncentrace na jednotku plochy – pro ilustraci, na metr čtvereční Merkuru musí dopadnout mnohem více paprsků ze Slunce než na metr čtvereční Marsu. U bodového světla, které má nulovou plochu a do všech bodů tedy vyšle právě jeden paprsek, však toto přirozeně nenastává a koncentrace světla (jeden paprsek na jeden bod) je všude stejná. Řešení skýtá podělení barvy světla druhou mocninou jeho vzdálenosti od zdroje, které tuto situaci modeluje.

Dalo by se namítnout, že světlo které se nepohybuje ve vakuu ztrácí intenzitu i kvůli dalším faktorům – například absorpcí a rozptylem světla při interakci s částicemi prostředí, kterým prochází. V případě vzduchu jsou však důsledky těchto jevů pozorovatelné až při velmi velkých vzdálenostech, jak je ilustrováno například obrázkem 2.10, tedy vzdálenostech se kterými se u většiny grafických aplikací neoperuje. Pokud tedy není cílem aplikace takovouto situaci modelovat, může být tato problematika při průchodu vzduchem ignorována a implementováno být pouhé zohlednění vzdálenosti.

Informace o tlumení světla čerpány z článku Natty Hoffman [6] a Joey de Vriese [11], kapitola PBR – Lighting.

Obrázek 2.10: Rozptýlení světla ve vzduchu způsobuje bělavý horizont, avšak není patrné na menších vzdálenostech. Obrázek získán z: https://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf

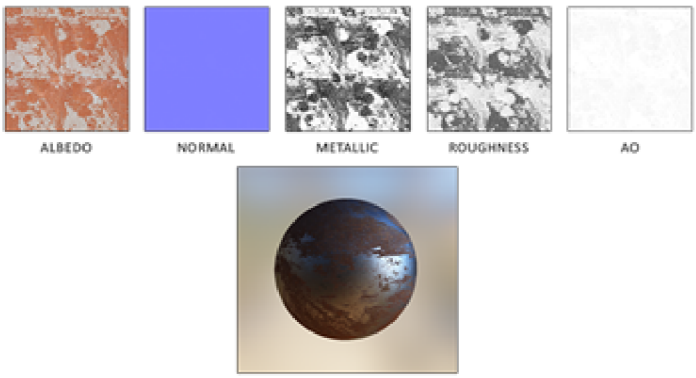
### 2.1.8 PBR materiály

V předchozích sekcích bylo zmíněno mnoho o vlastnostech materiálů, dosud však nebyla definována jejich přesná podoba. Jelikož implementací PBR existuje vícero, existuje i mnoho různých potřebných parametrů. V této práci jsou zmíněny ty nejčastější.

Parametry materiálu se do PBR modelu obvykle dodávají formou textur, kde barevná složka jejich pixelů definuje jejich číselné hodnoty. Příklady takovýchto textur lze vidět na obrázku 2.11. Albedo je barva materiálu, texturu definující normály je možno využít například pro bump mapping, kovovost a hrubost jsou extrahovány podle jasu a AO symbolizuje ambient occlusion, tedy techniku která do výsledku přidá na některá místa dodatečné stínování.

Jak si tedy lze povšimnout, po vytvoření PBR modelu začne být přidávání nových materiálu velmi modulární a jednoduchou záležitostí – stačí přidat několik textur, jejichž jednotlivé aspekty mohou být kdykoli změněny bez potřeby měnit ostatní.

Informace o materiálech v rámci PBR čerpány z webových stránek Joey de Vriese [11], kapitola PBR – Theory.




Obrázek 2.11: Jednotlivé parametry spolu často do určité míry korespondují. Obrázek získán z: https://learnopengl.com/img/pbr/textures.png

### 2.1.9 Image based lighting

Image based lighting, tedy "osvětlení založené na obraze", je kolekce technik implementujících osvětlení ne pomocí přímých zdrojů světla umístěných ve scéně, nýbrž uvažujících jako zdroj světla celou okolní scénu. Toho je obvykle dosaženo pomocí cubemapy, kdy do ní je buď přímo vykreslován snímek scény, nebo je použit nějaký obrázek (např. skybox). Každý bod takovéto cubemapy se tak stane bodem emitujícím světlo, díky čemuž lze modelovat i odrazy okolních objektů na povrchu objektu jiného.

Zatímco u bodových zdrojů světla byl integrál vyřešen poměrně triviálně, u IBL by byl příliš výpočetně náročný – jelikož zdrojem světla bude každý texel cubemapy, bylo by nutné každý fragment počítat pro ohromný počet takových světel. Řešením pak bude předpočítání většiny těchto výpočtů, tedy jejich vypočítání před spuštěním hlavní smyčky programu. Aby toho bylo dosaženo, je nutné se opět podívat na odrazovou rovnici a rozdělit ji na difuzní a spekulární část (2.12):

$$L_o(v) = \int_\Omega (k_d \frac{c}{\pi}) L_i(l)(n \cdot l) d\omega_i + \int_\Omega (k_s \frac{DFG}{4(n \cdot l)(n \cdot v)}) L_i(n \cdot l) d\omega_i \tag{2.12}$$

Informace o IBL včetně jeho difuzní a spekulární složky čerpány z článků Joey de Vriese [11], kapitoly PBR – IBL – Diffuse irradiance a PBR – IBL – Specular IBL.

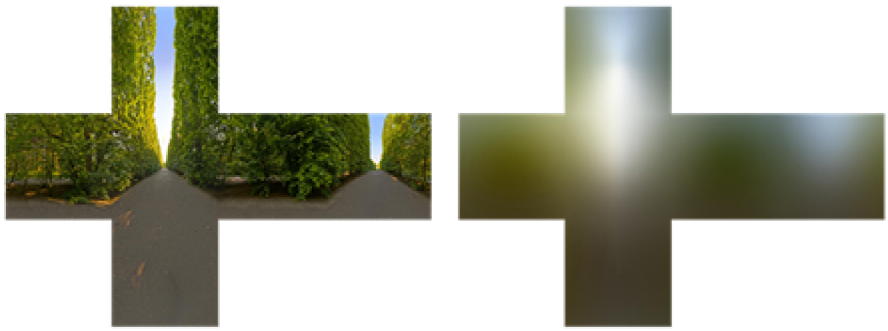
### 2.1.10 Difuzní IBL

Při bližším prozkoumání difuzní části rozepsané odrazové rovnice lze pozorovat, že jediná její nekonstantní část je vektor příchozího světla $l$. Lze tedy napsat rovnici (2.13):

$$L_{diffuse}(v) = k_d \frac{c}{\pi} \int_{\Omega} L_i(l)(n \cdot l) d\omega_i \tag{2.13}$$

Výsledný integrál je tedy závislý pouze na $l$ ($n$ bude vektorem, který bude výslednou cubemapu samplovat). Každý bod cubemapy je pak výsledkem konvoluce samplingu celé hemisféry, jejíž vrchol je orientován dle normály, která do tohoto bodu vede. Jinak řečeno, vytvářen je soubor výsledků integrálu pro všechny možné hodnoty normály při samplingu.

Samotná konvoluce je pak prosté zprůměrování výsledku integrálu – tvořena je tak takzvaná cubemapa intenzity záření, kterou stačí samplovat normálou zpracovávaného bodu a dosadit do rovnice. Jelikož je tato cubemapa vytvořena před spuštěním hlavní smyčky programu, není zde problém z hlediska výpočetní náročnosti. Příklad cubemapy intenzity záření je možné si prohlédnout na obrázku 2.12. Je důležité si uvědomit, že tento proces je prováděn pro jediný bod ve scéně – dělat ho pro body všechny by bylo opět výpočetně neúnosné. Výsledek tedy bude přesný pouze pro onen jeden bod, zatímco pro body ostatní bude tím nepřesnější čím dále od něj budou. Je tedy třeba volit tento bod s rozmyslem, aby ony nepřesnosti nebyly patrné, nebo takovýchto cubemap vytvořit pro scénu více, každou pro jiný bod, a vždy pak volit tu která je zpracovávanému bodu nejbližší.



Obrázek 2.12: Cubemapa intenzity záření vypadá jako rozmazaná cubemapa původní. Obrázek získán z: https://learnopengl.com/img/pbr/ibl_irradiance.png

Teoreticky je tedy postup jasný, avšak v praxi nelze spočíst integrál pro každý samplovací směr, jelikož takových směrů je nekonečno. Co však spočíst lze je určitý počet těchto směrů, rovnoměrně rozložených po dané hemisféře, čímž je možné získat poměrně přesnou aproximaci.

Co se proměnné $k_d$ udávající poměr difuzního světla oproti spekulárnímu týče, bude postupováno stejně jako v sekcích výše a bude využito Fresnelovy rovnice.

### 2.1.11 Spekulární IBL

Podobně jako difuzní složka IBL, i spekulární má tělo integrálu závislé na vektoru $l$, avšak na rozdíl od něj je závislé i na vektoru $v$. Nezáleží tedy pouze na směru, ze kterého přichází světlo, ale i na pozici ze které je objekt pozorován. Pokrýt všechny možné kombinace by tedy bylo ještě výpočetně náročnější než u difuzní složky a stejně jako u ní je tedy třeba

hlavní část výpočtu provést mimo hlavní smyčku programu. Rovnici (2.14) si lze rozepsat takto:

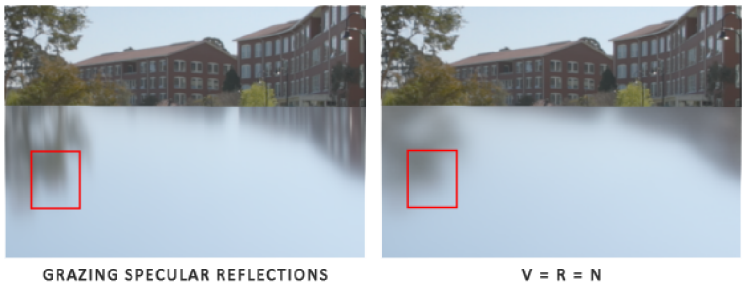
$$L_{specular}(v) = \int_{\Omega} L_i(l) d\omega_i \cdot \int_{\Omega} f(l, v) n \cdot l d\omega_i \tag{2.14}$$

První část rovnice je konvoluována do takzvané předfiltrované mapy okolí podobným způsobem jako byla konvoluována mapa intenzity záření, avšak tentokrát bude brána v potaz hrubost. Čím vyšší tato hrubost bude, tím rozptýlenější budou samplovací vektory a tedy tím rozmazanější bude výsledek. Takto vzniklé cubemapy pro různé úrovně hrubosti jsou pak ukládány do mipmap, v pořadí kde nejrozmazanější varianta s největší hrubostí bude nejmenší úrovní mipmapy. Příklad si lze prohlédnout na obrázku 2.13.

Obrázek 2.13: Podle výšky hrubosti bude později volino, která z mipmap bude použita pro vzorkování. Obrázek získán z: https://learnopengl.com/img/pbr/ibl_prefilter_map.png

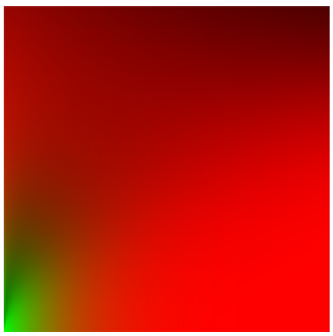
Spekulární odlesky jsou závislé i na směru, pod kterým je na ně nahlíženo (vektoru $v$), který však v době předpočítávání není znám. Použita tedy bude aproximace, kdy je na povrch nahlíženo pod nulovým úhlem, tedy $v = n$. Tato aproximace má na výsledek bohužel zaznamenatelný, negativní vliv, jak lze pozorovat na obrázku 2.14.

Druhá aproximace se pak bude týkat samotné hrubosti – cubemap nelze předpočítat neomezené množství, bude jich vytvořeno jen pár, přičemž bude pro daný bod využita ta, která se pro něj hodí nejvíce – obvykle tedy není získán přesný výsledek. Pokud pak je pak vypočítáno cubemap více, přechody mezi jednotlivými úrovněmi budou plynulejší a výsledky přesnější.




Obrázek 2.14: Vlevo odrazy získané bez aproximování směru pohledu, vpravo odrazy využívající této aproximace. Obrázek získán z: https://learnopengl.com/img/pbr/ibl_grazing_angles.png

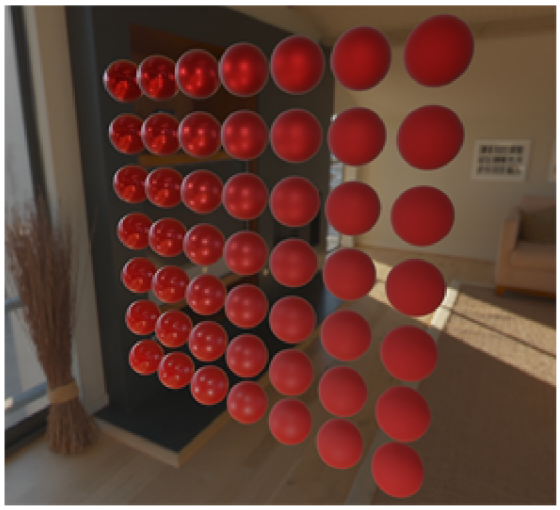
Druhá část rovnice se pak zabývá přímo BRDF. Za předpokladu, že příchozí světlo z libovolného směru bude bílé barvy, lze BRDF vypočítat pomocí hrubosti a úhlu mezi normálou a odraženým světlem (tedy $n \cdot v$), což jsou oba skaláry pohybující se svou hodnotou mezi 0 a 1. Tento fakt lze využít k vyřešení celého problému – všechny možné kombinace mezi dvěma skaláry (s rozumně zvolenou přesností) totiž lze vypočítat a výsledky zakódovat do textury. Takovouto texturu si je pak možno prohlédnout na obrázku 2.15, kde horizontální osa reprezentuje skalární součin a vertikální hrubost.



Obrázek 2.15: Oba skaláry jsou později využity jako koordináty pro sampling z textury. Obrázek získán z: https://learnopengl.com/img/pbr/ibl_brdf_lut.png

Tím jsou tedy spočítány oba integrály spekulární části IBL, které již stačí jen zkombinovat – z předfiltrované mapy okolí je vybrána mipmapa podle hodnoty hrubosti, ta je osamplována a výsledek je vynásoben s výsledky samplingu textury pro BRDF. Zapomenout samozřejmě nelze ani na $k_s$, která je získána v rámci difuzního IBL.

Difuzní a spekulární část IBL pak již stačí jen sečíst a celý model je hotov – dostatečně kovové nebo lesklé předměty budou odrážet scénu, zatímco ty hrubší budou nasvíceny tak, jak by v reálném životě bylo očekáváno. Image based lighting je možné kombinovat s bodovými zdroji světla pro ještě dynamičtější řešení, příklad kterého lze vidět na obrázku 2.16.



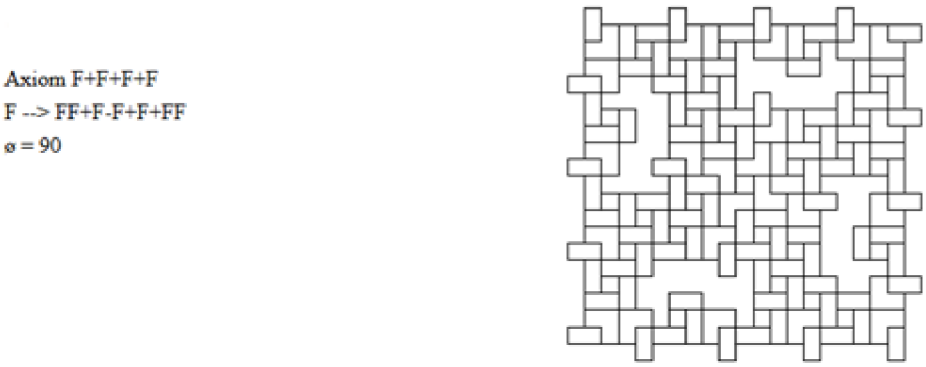
Obrázek 2.16: Koule jsou seřazeny podle hrubosti a kovovosti. Za povšimnutí stojí, že kovová, lesklá koule (vlevo nahoře) odráží celou scénu, zatímco s klesající kovovostí začnou být patrné spíše bodové zdroje světla. Obrázek získán z: https://learnopengl.com/img/pbr/ibl_specular_result.png

## 2.2 L-systémy

L-systémy, někdy také nazývány Lindenmayerovy systémy, jsou formální gramatikou – souborem pravidel pro přepisování řetězců, obvykle tvaru $A \rightarrow BC$. Aplikování takové gramatiky na řetězec pak spočívá v průchodu řetězcem, nalezením symbolu či skupiny symbolů které jsou v souboru pravidel na levé straně a jejich nahrazení symboly na straně pravé.

Gramatiky se dělí na deterministické a nedeterministické. Deterministické mají vždy právě jedno pravidlo aplikovatelné pro daný symbol či skupinu symbolů a výsledek jejich aplikace je tak vždy stejný a předvídatelný. Nedeterministické gramatiky naproti tomu mohou mít pro jednu skupinu symbolů pravidel více a výsledek je tedy závislý na pravděpodobnosti a může být při každé aplikaci jiný. Pokud se gramatika zacyklí a pravidla může opakovaně aplikovat donekonečna (např. $A \rightarrow AA$), pak je nutné definovat množství cyklů, po kterém bude proces její aplikace ukončen.

V počítačové grafice se pak L-systémy využívají zejména pro tvorbu fraktálů – každý symbol výsledného řetězce totiž je možné uvažovat jako akci, kterou vykoná program. Velmi častá je pak takzvaná analogie s želvou – program ovládá želvu příkazy typu „otoč se o 90 ° vpravo“, „posuň se o 10 jednotek vpřed“, „kresli“, ale i „vytvoř objekt“ či jiné složitější úkoly, každý reprezentován některým ze symbolů řetězce. Příkladem může být fraktál na obrázku 2.17, kde lze vidět počáteční axiom (původní řetězec před aplikací pravidel gramatiky), jedno přepisovací pravidlo a počet cyklů.




Obrázek 2.17: *F* značí posun vpřed za kreslení čáry, plus a mínus udávají změnu směru. Obrázek získán z: http://paulbourke.net/fractals/lsys/

Takovéto fraktály by pak byly příkladem deterministické gramatiky, avšak často je cílem se právě determinismu a pravidelnosti vyhnout, konkrétně například při tvorbě vegetace. Mnoho rostlin se totiž při svém růstu chová jako by byly právě nějakým fraktálem, ovšem jak lze pozorovat v reálném světě, jen velmi obtížně by se našly dvě rostliny, které by vypadaly zcela identicky. Právě zde tedy nachází využití gramatiky nedeterministické, díky kterým lze vytvářet například stromy, které si budou navzájem velmi podobné, avšak nebudou zcela identické. V současné době lze tento přístup pozorovat v mnoha počítačových hrách.

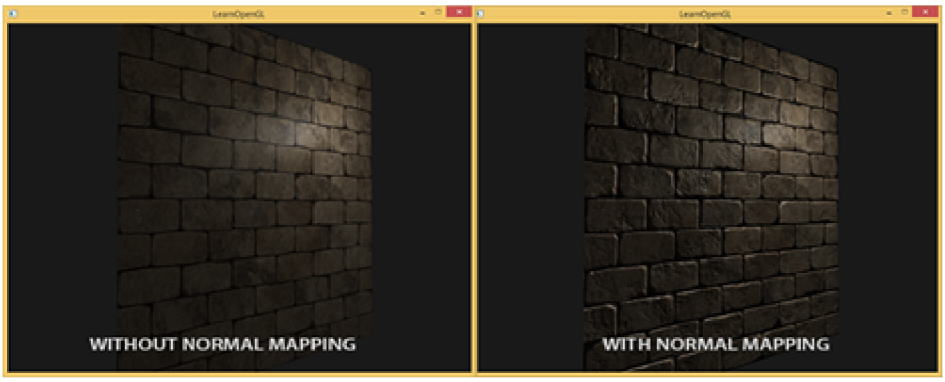
Informace o L-systémech čerpány z webových stránek Roberta M. Dickeau [4] a z knihy Noora Shakera [10].

## 2.3 Normal mapping

Normal mapping, někdy také nazývaný bump mapping, je grafická technika zabývající se zvýšením množství grafických detailů a zlepšením vizuální kvality objektu bez úpravy jeho geometrie. Technika pak obvykle předpokládá model implementující nějaký druh osvětlení, který bere v potaz normály fragmentů povrchu. Laicky řečeno, normal mapping se snaží u placatého povrchu vyvolat dojem, že placatý není.

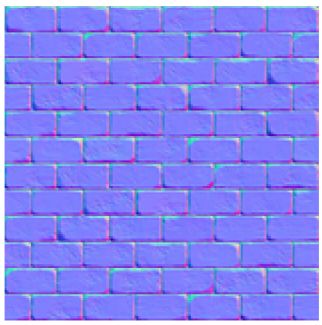
Je-li volena kupříkladu textura cihlové zdi, která je namapována na obdélník, lze pozorovat, že nepůsobí příliš přesvědčivě – že je geometrie objektu zcela rovná je zřejmé na první pohled, zejména pokud je na ní dokonce vidět spekulární odlesk. Změnit samotnou geometrii obdélníku tak, aby byl náležitě hrubý, s různými šrámy, spárami, vystouplými cihlami a dalšími kazy, by však výrazně zvýšilo výpočetní náročnost – místo čtyř vertexů pro obdélník by jich byly na grafickou kartu posílány stovky nebo i tisíce a více, podle toho jak detailní by takový objekt byl.

K takto drastickým zákrokům však, naštěstí, nemusí dojít při uvědomění si, že to, co způsobuje onu vizuální plochost je způsob, jakým plocha interaguje se světlem. Pokud bude světlo donuceno, aby s objektem interagovalo tak, jako by plochý nebyl, bude získán realisticky a detailně vypadající obraz s jednoduchou geometrií. Účinek této techniky si lze prohlédnout na obrázku 2.18.



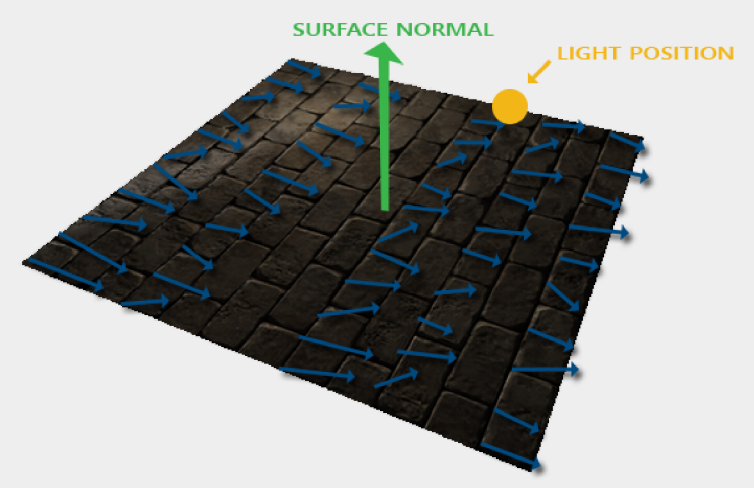

Obrázek 2.18: Vlevo textura zdi bez použití normal mappingu, vpravo s. Obrázek získán z: https://learnopengl.com/img/advanced-lighting/normal_mapping_compare.png

V běžných osvětlujících modelech je obvykle interakce světla s plochou řešena pomocí povrchových normál, které v podstatě říkají, jakým směrem je daný fragment orientován. Právě na tom staví i celá technika normal mappingu, jejíž princip je zřejmý z obrázku 2.19, kde barevné složky každého pixelu definují normálu. Místo aby normály reprezentovaly skutečný tvar povrchu, budou nastaveny tak, jako by model skutečně měl složitou geometrii plnou nerovností.



Obrázek 2.19: V tzv. normal mapách obvykle převládá modrá složka (tedy osa $z$). Obrázek získán z: https://learnopengl.com/img/advanced-lighting/normal_mapping_normal_map.png

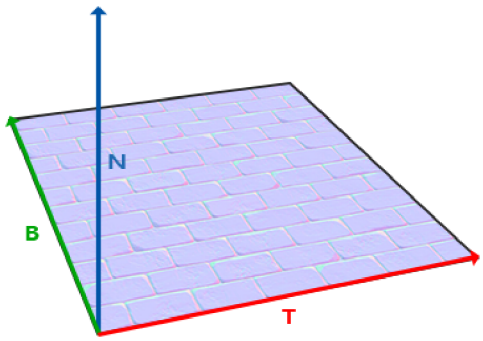
Tento jednoduchý koncept však naráží na problém – obvykle je cílem textury využít na více objektech, které zdaleka nemusejí být všechny orientovány stejným směrem. Textura zdi má být pravděpodobně využita pro více než jen jednu stěnu místnosti, avšak normal mapa bude stále ukazovat stejné normály, tedy normály orientované stejným směrem ať už je textura aplikovaná na cokoli. To způsobuje chybnou interakci světla světla s plochou, jak lze vidět na obrázku 2.20. Možným řešením by bylo vytvořit jinou normal mapu pro každou ze stěn, což je však zřejmě řešení nerealizovatelné pokud je možných směrů orientace texturovaného tělesa více než pár. Skutečné řešení problému tak spočívá v takzvaném tečném prostoru, neboli *tangent space*.



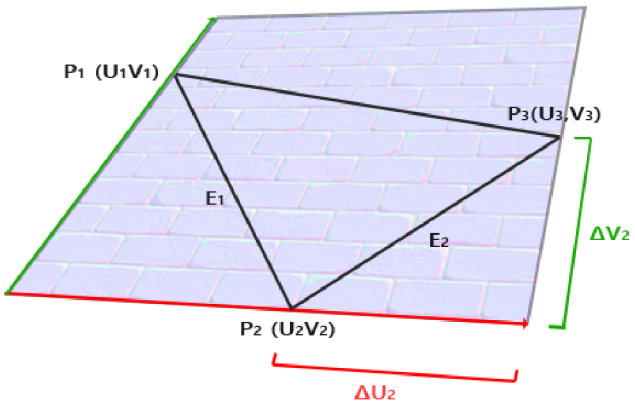

Obrázek 2.20: Textura zdi byla aplikována na zem, avšak normály ukazují stejným směrem jako dříve. Pozorovat lze výsledné (nesprávné) chování světla. Obrázek získán z: https://learnopengl.com/img/advanced-lighting/normal_mapping_ground_normals.png

Tangent space je prostor lokální pro daný trojúhelník – normály jsou uvažovány relativně vůči danému trojúhelníku, tedy se chovají tak, jak je žádoucí. Jako příklad lze uvést, že jako referenční plocha při tvorbě normal mapy bude používán obdélník umístěný do roviny tvořené osami $x$ a $y$. Většina normál v normal mapě s touto referenční plochou tedy bude orientovaná více či méně ve směru osy $z$ v pozitivním směru. V tangent space pak bude normála povrchu trojúhelníka (nikoly normal mapy) tvořit onu osu $z$, a normála získaná z normal mapy tedy bude mít správnou orientaci. Pak je již jen třeba získat matici která takovouto normálu při pronásobení převede z tangent space do world space. Tato matice se nazývá matice TBN, tedy matice Tanget-Bitangent-Normal podle svých jednotlivých složek. Tangenta, bitangenta a normála jsou pak osami tangent space, jak lze ukázat na obrázku 2.21, kde lze také pozorovat, že tangenta i bitangenta jsou orientovány stejným směrem jako texturovací koordináty povrchu (teoreticky by tangenta mohla vést libovolným směrem v dané rovině, avšak navázání na texturovací koordináty zajistí konzistenci se sousedy a předejde se tak nepěkným přechodům). To pak umožňuje tangentu i bitangentu spočítat – případ je ilustrován obrázkem 2.22 a rovnicí 2.15.

$$\begin{bmatrix} T_x T_y T_z \\ B_x B_y B_z \end{bmatrix} = \frac{1}{\delta U_1 \delta V_2 - \delta U_2 \delta V_1} \begin{bmatrix} \delta V_2 & -\delta V_1 \\ -\delta U_2 & \delta U_1 \end{bmatrix} \begin{bmatrix} E_{1x} E_{1y} E_{1z} \\ E_{2x} E_{2y} E_{2z} \end{bmatrix} \tag{2.15}$$




Obrázek 2.21: $T$ je tangenta, $B$ bitangenta a $N$ normála. Dohromady tvoří osy tangent space. Obrázek získán z: https://learnopengl.com/img/advanced-lighting/normal_mapping_tbn_vectors.png




Obrázek 2.22: Tangent space lze spočítat z pozic a texturovacích koordinátů zpracovávaného trojúhelníku. Obrázek získán z: https://learnopengl.com/img/advanced-lighting/normal_mapping_surface_edges.png

Matici TBN je pak možné použít dvěma způsoby – dříve zmíněné převedení normály z tangent space do world space, nebo naopak invertování TBN matice a s její pomocí převedení veškerých relevantních proměnných světla do tangent space. Obě metody jsou ve výsledku ekvivalentní – důležité je pouze zajistit, že veškeré výpočty probíhají ve stejném prostoru.

Více informací o normálovém mapování je dostupných na webových stránkách Joey de Vriese [11], kapitola Advanced Lighting – Normal Mapping.

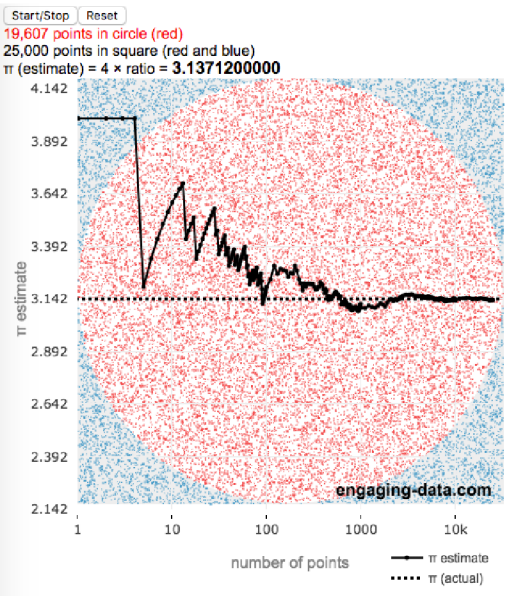
## 2.4 Metody Monte Carlo

Metody Monte Carlo jsou třídou výpočetních algoritmů, které procesem opakovaných náhodných vzorkování (samplingem) aproximují výslednou hodnotu daného problému. Existuje mnoho různých metod Monte Carlo které nalezly uplatnění v mnoha různých oborech

(finančnictví, fyzika, matematika atp.), avšak všechny obvykle operují s náhodnými (nebo alespoň pseudo-náhodnými) čísly.

Principem celé problematiky je zákon velkých čísel – pokud je zpracovávána podmnožina skutečně náhodných vzorků z velmi velké množiny čísel, pak se výsledek tohoto zpracování (např. aritmetický průměr) bude blížit výsledku stejné operace nad celou množinou, přičemž čím větší byla zpracovávaná podmnožina, tím přesnější výsledek bude. Jako příklad lze uvést změření průměrné výšky obyvatel České Republiky. Pokud by byl měřen každý z obyvatel a z výsledných čísel byl udělán průměr, byl by získán přesný výsledek, avšak bylo by to neúnosně pracné – měřit každého z obyvatel zkrátka není realizovatelné. Změření tisíce lidí však už proveditelné je, a pokud je těchto tisíc lidí vybráno doopravdy náhodně, pak výsledný průměr bude podobný tomu skutečnému.

Matematicky zajímavější je pak aplikace metody Monte Carlo pro integraci – pokud je dána křivka a úkolem je zjistit, jaká je plocha pod ní v nějakém intervalu, pak stačí ji umístit do obdélníkové (či jiné) plochy a z ní pak brát náhodné vzorky. Po sesbírání dostatečného počtu vzorků stačí podělit počet vzorků pod křivkou celkovým počtem vzorků, čímž je vlastně získán poměr plochy pod křivkou vůči ploše celé. Poté jen stačí tento poměr pronásobit plochou celého útvaru (jelikož se obvykle jedná o obdélník, je jeho plocha velmi snadno spočitatelná) a je získán výsledek. Takovýto výsledek bude opět pouze aproximací, avšak poměrně přesnou, obzvláště pokud je použito velké množství vzorků. Klasickým příkladem tohoto postupu je pak výpočet hodnoty Ludolfova čísla, který si lze prohlédnout na obrázku 2.23.



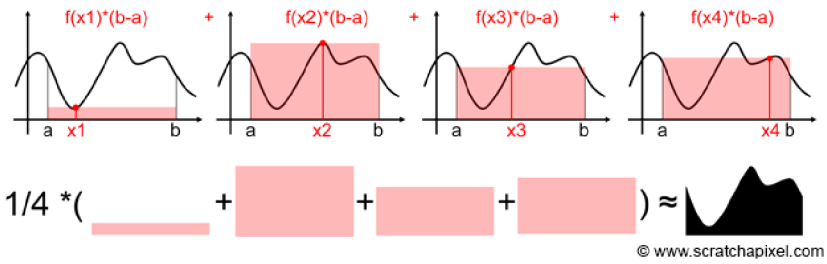

Obrázek 2.23: Křivkou je v tomto případě kružnice. Jak lze vidět, ani při 25 000 vzorcích nebylo dosaženo přesného výsledku, nýbrž výsledku přibližného, který se zpřesňuje spolu s přidáním dalších a dalších vzorků. Obrázek získán z: https://engaging-data.com/pages/scripts/pi/pi.png

Zajímavějším přístupem u kterého není zapotřebí žádný "obdélník"je výpočet určitého integrálu. Jako příklad budiž uveden následující integrál (2.16):

$$F = \int_a^b f(x)dx \tag{2.16}$$

Místo generování náhodných bodů v ploše budou generovány pouze hodnoty na ose $x$ v intervalu mezi $a$ a $b$, načež pro každou bude vypočítáno $f(x)$ – to jsou vzorky. Každý vzorek pak bude pronásoben délkou intervalu – získána tak bude aproximace plochy pro každý vzorek. Principem pak je, že s dostatečným počtem vzorků se takto získané plochy zprůměrují až ke kvalitní aproximaci správného výsledku – příklad je ilustrován na obrázku 2.24. Tuto metodu lze vyjádřit rovnicí (2.17) nazvanou *Monte Carlo Estimator* ($Xi$ značí náhodně vybranou hodnotu $x$ v intervalu mezi $a$ a $b$):

$$\langle F^N \rangle = (b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(Xi) \tag{2.17}$$



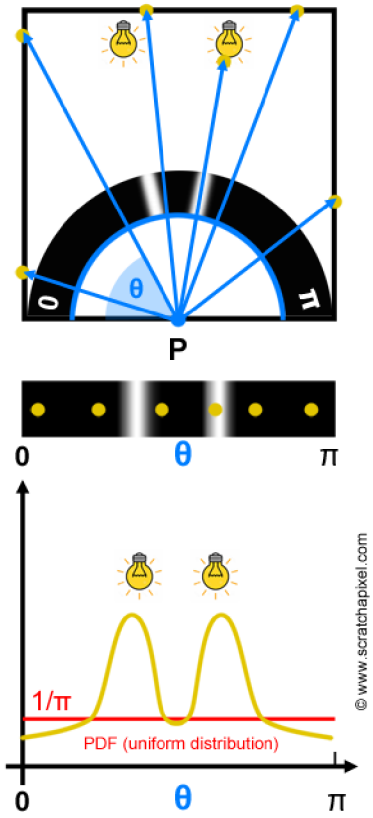

Obrázek 2.24: Výpočet určitého integrálu pomocí metody Monte Carlo. Obrázek získán z: https://www.scratchapixel.com/images/upload/monte-carlo-methods-practice/MCIntegration03.png?

U všech těchto operací je třeba, aby vzorky byly brány náhodně, nebo alespoň pseudonáhodně, s rovnoměrným rozložením pravděpodobnosti – pokud by vzorky měly vyšší pravděpodobnost, že například padnou to levé poloviny intervalu než do pravé, pak by samozřejmě metoda Monte Carlo nemusela fungovat – výsledky by byly touto nerovností pokřivené. Pokud k takovému případu dojde, je třeba vzorky podělit jejich pravděpodobností – hodnoty které mají vyšší pravděpodobnost, že budou generovány, budou mít nižší váhu, zatímco méně pravděpodobné hodnoty budou mít váhu vyšší. Obecnější Monte Carlo estimator (2.18) pak bude vypadat:

$$\langle F^N \rangle = \frac{1}{N}\sum_{i=0}^{N-1} \frac{f(Xi)}{pdf(Xi)} \tag{2.18}$$

Kde $pdf(Xi)$ je hodnota funkce hustoty rozdělení pravděpodobnosti pro proměnnou $Xi$.

Metody Monte Carlo trpí vážnou vadou, a to sice množstvím vzorků potřebných k získání přesnějších výsledků – pro zmenšení chyby ve výsledku na polovinu je třeba přidat čtyřnásobek současného množství vzorků. Bylo by dobré tento aspekt nějakým způsobem vylepšit. Budiž dána jako příklad situace, kdy je hledána míra osvětlení bodu v černé krabici, do níž dvěma otvory jejichž polohu neznáme proudí světlo zvenčí. Běžný přístup metod Monte Carlo by tkvěl v generování náhodných hodnot s rovnoměrným rozložením pravděpodobnosti do plochy celé hemisféry okolo bodu. Situace je ilustrována na obrázku 2.25. Jak lze pozorovat, u funkcí které neprobíhají plynule a mají náhlé, prudké výkyvy se může snadno stát, že vzorky onen výkyv celý minou nebo ho ovzorkují jen vzácně, a to přesto že právě onen výkyv má na výsledek největší vliv. Aby by získán rozumně přesný výsledek, bylo by zapotřebí ohromného množství vzorků.



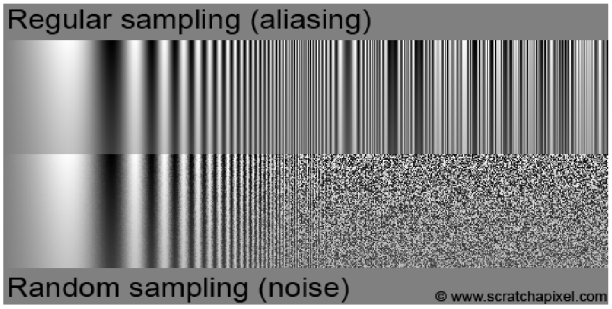

Obrázek 2.25: Ovzorkování náhlých výkyvů může být při rovnoměrném rozložení poměrně vzácné. Obrázek získán z: https://www.scratchapixel.com/images/upload/monte-carlo-methods-practice/importancesampling01.png?

Jak tedy tento problém řešit? Nechť je uvedena konstantní funkce $f(x) = 3$. Vypočítání integrálu metodou Monte Carlo je pak nejen triviální, ale i naprosto přesné – chyba bude nulová. Principem řešení složitějších funkcí pak může být jejich převedení na funkce konstantní, a to jejich podělením funkcí takovou, že je násobkem funkce původní. Tedy $\frac{f(x)}{f'(x)} = \frac{1}{c}$ kde $c$ je konstanta. Funkcí, která se pro tento přístup přímo nabízí je funkce rozložení hustoty pravděpodobnosti, jak je možné vidět na rovnici 2.19:

$$\frac{f(x)}{pdf(x)} = \frac{f(x)}{c \cdot f(x)} = \frac{1}{c} \tag{2.19}$$

Jinak řečeno, vzorky samplující důležité části funkce mají větší šanci být vygenerovány, než vzorky jiné, a je tedy snížena šance, že při vzorkování bude přeskočen nějaký důležitý aspekt, jako tomu bylo například v případě krabice s dírami. Klíčem je tak vytvoření takového rozložení hustoty pravděpodobnosti, jaké bude podobné dané funkci – získáné výsledky tak budou výrazně přesnější i při mnohem menším počtu vzorků. Samozřejmě, aby bylo možné se o vytvoření takového rozložení pokusit, je nutno o zkoumané funkci nejdříve něco vědět. Proto tento přístup nelze využít pro řešení problémů, kde o průběhu funkce není nic známo – v takových případech bude použito rozložení pravděpodobnosti rovnoměrné. Přístup, kdy důležitější místa funkce mají větší pravděpodobnost být vzorkována nazýváme *importance sampling*.

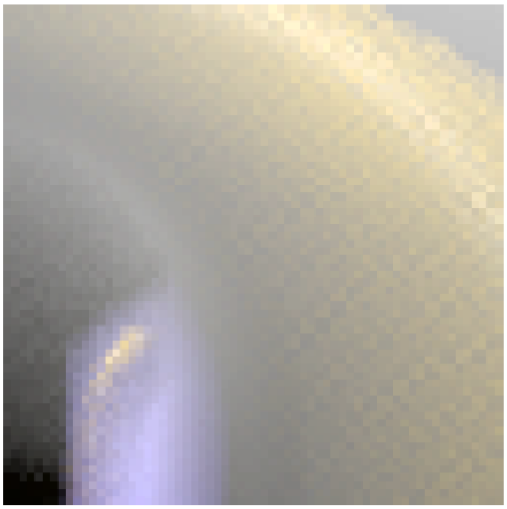
Existuje však ještě další negativní vlastnost klasických metod Monte Carlo (včetně importance samplingu), která vyplývá z využití náhodnosti – takzvaný *sample clumping*, tedy hromadění vzorků. Může se stát, že budou náhodně vygenerovány dva podobné nebo skoro stejné vzorky – informace z prvního vzorku jistě užitečná je, avšak ten druhý vlastně o zkoumaném problému neříká nic nového. Pokud jsou generované hodnoty skutečně zcela náhodné, nedá se tomuto problému zabránit, avšak pokud by byly zvoleny hodnoty bez náhodnosti a byly například pravidelně rozloženy, mohl by vzniknout aliasing, který je možné si prohlédnout na obrázku 2.26. Ideálním řešením by tedy bylo něco mezi těmito přístupy – něco co by generovalo vzorky dostatečně pravidelně na to, aby nedocházelo k sample clumpingu, ale zároveň dostatečně náhodně na to, aby nedocházelo k aliasingu. Tímto řešením jsou tvz. *low-discrepancy sequences*, volně přeloženo *sekvence nízkého nesouladu*. Discrepancy si lze představit jako odchylku od pravidelného rozložení – zcela náhodné sekvence ji budou mít vysokou, pravidelné rozložení ji bude mít nulovou. Low-discrepancy sequence je pak taková sekvence, která má tuto hodnotu nízkou, avšak nikoli nulovou – tedy sekvence jejíž hodnoty jsou více méně pravidelně rozložené a tedy se neshlukují, avšak stále jsou alespoň trochu náhodné. Metody využívající tyto sekvence pak nazýváme quasi Monte Carlo metody.




Obrázek 2.26: Nahoře lze pozorovat výstup metody Monte Carlo s plně deterministickým, pravidelným vzorkováním. Dole je vzorkování náhodné. Vzorkována je funkce sinus které postupně narůstá frekvence - v momentě kdy Nyquistova frekvence přesáhne frekvenci vzorkovací je možné vidět aliasing u pravidelného vzorkování a šum u náhodného. Šum je obvykle považován za přijatelnější, než aliasing. Obrázek získán z: https://www.scratchapixel.com/images/upload/monte-carlo-methods-practice/aliasing.png?

Vyvstává otázka, proč tedy nejsou quasi Monte Carlo metody používány úplně vždy místo těch obyčejných – vygenerovat low-discrepancy sekvenci přeci nemůže být příliš obtížné a o zkoumané funkci (narozdíl od importance samplingu) není nutné nic vědět, proč tedy nepoužít rychlejší variantu? Důvodem je, že low-discrepancy sekvence mívají periodickou strukturu, což se ve výsledku může projevit. Takový projev je ilustrován obrázkem 2.27, kde je výsledný defekt jasně viditelný. Zbavení se těchto artefaktů vyžaduje další práci, jako je například otáčení vzorků. Je tedy třeba vždy nejdříve zvážit, která z metod je nejvhodnější pro daný problém.

Informace o metodách Monte Carlo čerpány z portálu Scratchapixel [9].



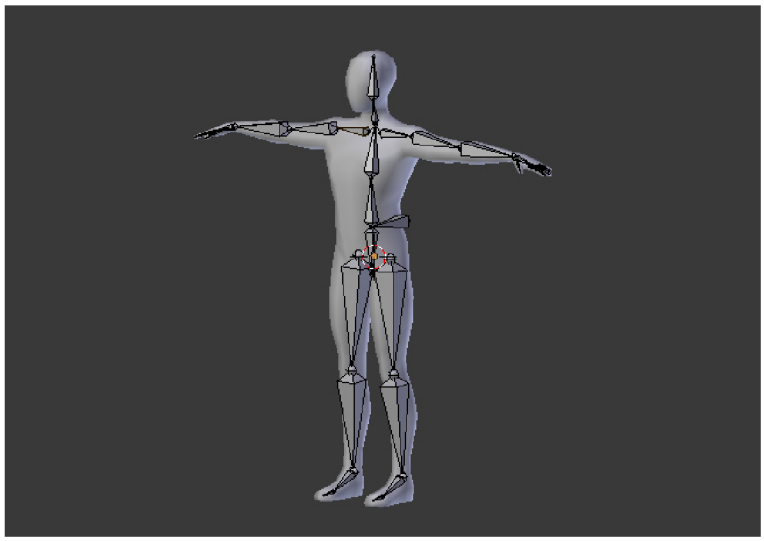
Obrázek 2.27: Na obrázku je viditelný vzor, vzniklý kvůli využívání low-discrepancy sekvence. Obrázek získán z: https://www.scratchapixel.com/images/upload/monte-carlo-methods-practice/structuredaliasing.png?

## 2.5 Kosterní animace

Kosterní animace je technika v oblasti počítačové grafiky, která umožňuje pohybovat různými částmi objektů ve scéně v čase prostřednictvím takzvané kostry. Technika je široce používána zejména v oblastech počítačových her a filmového průmyslu. V následujících podkapitolách bude nejdříve uvedena kostra a elementy, které ji tvoří, načež bude rozebráno její animování. Informace o problematice čerpány z návodu na kosterní animaci za použití knihovny Assimp [8] a z článku Curtise Beesona o demonstrační aplikaci *Dawn* společnosti NVIDIA [2].

### 2.5.1 Kostra

S pojmem kostra se, v rámci kosterní animace, pojí dva další pojmy – kostra jako taková, a povrch (někdy také nazývaný síť, anglicky *mesh*). Povrch sestává z polygonů tvořených vertexy a vytváří to, co v aplikaci lze skutečně vidět. Kostra naproti tomu renderována není. Situaci je možné si prohlédnout na obrázku 2.28.

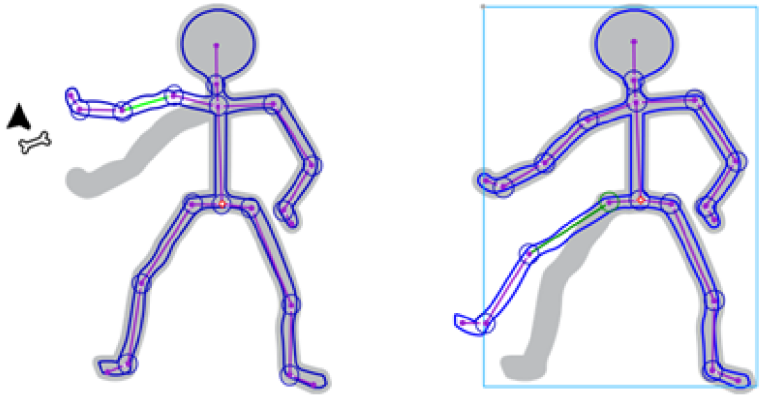


Obrázek 2.28: Na obrázku lze vidět kosti (*bones*) reprezentované polygony a klouby (*joints*) reprezentované body, které dohromady tvoří kostru. Model lidského těla je pak povrch. Obrázek získán z: http://morpheo.inrialpes.fr/~franco/3dgraphics/_images/anim1.jpg

Princip fungování kostry je poměrně jednoduchý – kosti mění svoji polohu a rotaci a ovlivňují tak části povrchu na ně napojené, jako skutečné kosti ovlivňují kůži. V rámci kostry se pak lze setkat se dvěma pojmy – kosti a klouby. Jedná se o abstrakci užívanou pro snazší pochopení problematiky, zejména v rámci modelovacích softwarů, avšak některé zdroje mezi těmito pojmy nerozlišují, jelikož kosti jako takové de facto neexistují – veškerou práci vykonávají ve skutečnosti klouby, tedy struktury uchovávající svou pozici, rotaci a hierarchii.

Jedním ze základních procesů kosterní animace je tzv. *rigging*, tedy napojení povrchu na jednotlivé kosti. Toho je dosaženo přidáním jména kosti každému vertexu, který tato kost ovlivňuje. Pokud vertexy může ovlivňovat kostí více, je třeba přidat k jejich jménům i váhy, které určují v jakém poměru jejich vlivy jsou. Tato část je pak onou výše zmiňovanou "kostí", ačkoli, jak lze snadno ukázat, nemusí být v kódu vůbec reprezentována. Nechť je kloub na nějaké pozici a v nějaké rotaci. Dále nechť je několik vertexů závislých na kosti závislé na tomto kloubu – samotná informace o pozici a rotaci kloubu plně postačuje k přesunutí vertexů tak, jako kdyby s nimi skutečně hýbala skutečná kost.

Jednotlivé kosti, či klouby, existují ve stromové hierarchii, kdy změna rodiče ovlivní i všechny potomky. Například u kostry člověka pak pohnutí paže způsobí i pohyb lokte, ruky, prstů atp. Situace je ilustrována na obrázku 2.29. Jedinou výjimkou z tohoto pravidla je kořen stromu, tedy kloub který žádného rodiče nemá a který ovlivňuje všechny uzly struktury. Tento kořen může být umístěn kdekoli (jako ostatně všechny kosti a klouby), avšak v případě modelu lidského těla se obvykle umisťuje do oblasti třísel, případně na zem. Často s ním není manipulováno a funguje jen jako referenční bod.



Obrázek 2.29: Rotace kloubem orotuje i všechny jeho potomky. Za povšimnutí stojí, že úhly a vzdálenosti mezi potomky samotnými zůstávají zachovány. Obrázek získán z: https://www.technokids.com/blog/wp-content/uploads/2016/10/technokids-10.png

### 2.5.2 Animace

Animace (v rámci tématu kosterní animace) se skládá z animačního času a klíčových snímků. Klíčové snímky představují pózu kostry v nějakém časovém momentu, tedy soubor pozic a rotací pro všechny části kostry pro nějaký čas. Interpolací mezi těmito snímky je získána póza pro daný moment, kterou po pronásobení potomků rodiči lze aplikovat na příslušné vertexy. Díky tomuto přístupu je tak znám tvar kostry v libovolném časovém okamžiku, je docíleno plynulého pohybu animace a její podobu stačí definovat relativně jednoduchým aranžováním kostry v jednotlivých klíčových snímcích, což je obvykle dále usnadněno různými modelovacími softwary.

## 2.6 Šablonování

Šablonování je geometrická konstrukce, kdy vytvářené těleso vzniká pohybem křivky či plochy – lze si představit, že pohybující se křivka za sebou zanechává stopu, která je výsledným tělesem. Přístupů existuje vícero – prvním je šablonování translační, kdy je daná křivka či plocha posouvána v nějakém směru, přičemž může měnit svůj tvar. Druhým přístupem je pak šablonování po šroubovici, kdy křivka či plocha rotují kolem nějaké osy. Přístupy je možno kombinovat.

Šablonování výrazně usnadňuje tvorbu meshe oproti jeho manuálnímu definování – stačí ručně nadefinovat jedinou křivku, která je pak algoritmicky aplikována pro generování celého tělesa. Příklady obou výše zmíněných přístupů jsou ilustrovány na obrázku 2.30. Informace o šablonování čerpány ze stránek ústavu matematiky fakulty strojního inženýrství VUT v Brně [3].

Obrázek 2.30: Vlevo útvar získaný pomocí translačního šablonování, uprostřed a vlevo příklady šablonování po šroubovici. Obrázek získán z: http://mathonline.fme.vutbr.cz/pg/Dcad/08_Sablonovani.htm

# Kapitola 3

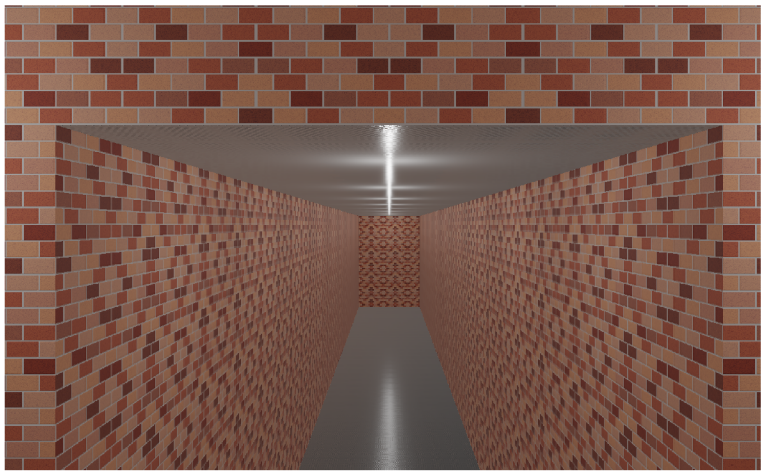
# Návrh

V následujících sekcích bude blíže popsána konkrétní podoba aplikace a postupy použité k její realizaci.

## 3.1 Osvětlovací model

Osvětlovací model programu vychází z technik PBR, konkrétně pak kombinuje techniky pro bodové zdroje světla a image based lighting a využívá PBR materiály pro albedo, normály, kovovost a hrubost. Velmi podobnou implementaci PBR pak lze nalézt v herním enginu Unreal Engine 4 od Epic Games [7].

Nejdříve bylo implementováno osvětlení pomocí bodových zdrojů světla. Pozice bodových světel ve scéně jsou posílány do fragment shaderu, kde je pak implementována Cook-Torranceova BRDF popsaná v sekcích 2.1.1, 2.1.2 a 2.1.3 využívající Trowbridge-Reitzovu GGX pro výpočet normálové distribuční funkce, Schlickovu GGX pro geometrickou funkci a Fresnel-Schlickovu aproximaci pro Fresnelovu funkci. Vyzařované světlo je definováno bílé a implementuje tlumení světla popsané v sekci 2.1.7. Samotná implementace pak spočívá v pouhém dosazení do zmíněných rovnic, přičemž výsledek tohoto přístupu je patrný na obrázku 3.1.



Obrázek 3.1: Kromě zřejmých odlesků na kovové ploše stojí za povšimnutí i vliv světel na zdi, kde sice nejsou tolik vidět spekulární odlesky díky vysoké hrubosti a nízké kovovosti, avšak jejich povrch je stále znatelně světlejší než povrchy nenasvícené, viz střídavé pruhy na konci chodby.

Podstatně složitější pak byla implementace IBL – bylo třeba tří dalších shaderů pro předpočítání cubemap intenzity záření a okolí a pro BRDF texturu. Zdrojem pro IBL pak byla zvolena samotná scéna, respektive v ní zasazená místnost. Před spuštěním hlavní smyčky programu tak je nejdříve vyrenderovaná ona místnost, do níž je umístěna tzv. sonda, tedy kamera se zorným polem o velikosti devadesát stupňů. Obraz z ní je pak renderován do textury, načež se kamera otočí o devadesát stupňů a proces opakuje. Tyto kroky jsou pak opakovány dokud není vyrenderovaných textur šest, každá pro jednu stěnu pomyslné krychle, a je tedy právě jednou nasnímán každý bod scény viditelný z bodu, do kterého je sonda umístěna. Z těchto snímků je pak vytvořena cubemapa.

Jelikož není možné během předpočítávání vyfotit místnost s již implementovaným IBL, je místo toho focená místnost renderována pouze s aplikovaným světlem z bodových zdrojů světla. To může způsobit, že odrazy opět nebudou úplně přesné, avšak do výsledné kvality obrazu se to příliš nepromítne.

Pro tvorbu cubemap intenzity záření a okolí a BRDF textury byl využit stejný vertex shader, který pouze vytváří fullscreen quad (obdélník zabírající celé zorné pole kamery) a k němu generuje texturovací koordináty, které jsou pak pošlány do patřičných fragment shaderů. Do fragment shaderů je pak při tvorbě cubemap poslána uniformní proměná říkající, která ze stěn je právě zpracovávána (každá stěna se renderuje zvlášť), takže stačí patřičně orotovat texturovací souřadnice a použít je pro sampling. Na výstupu tak opět bude jedna ze stěn cubemapy, která bude vyrenderována do textury a posléze s ostatními takto vytvořenými texturami opět sestavena do cubemapy. U BRDF textury samozřejmě nejsou tvorba cubemapy ani operace s ní spojené třeba.

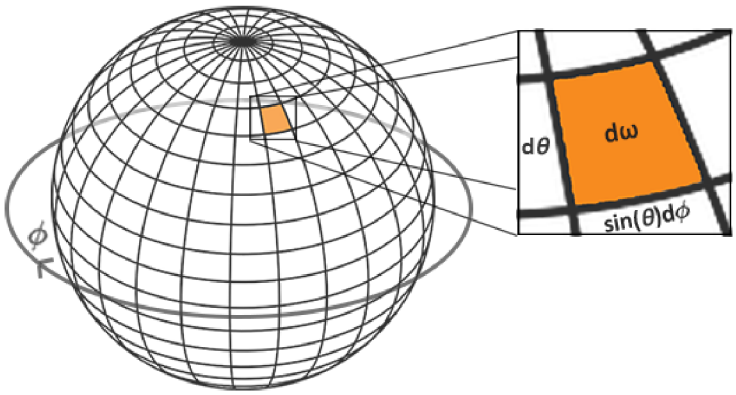
Jelikož je, jak je zmíněno v sekci 2.1.10, sečtení světla ze všech směrů, které do daného bodu vedou, prakticky neproveditelné, je pro tvorbu cubemapy intenzity záření třeba samplovat hemisféru nad daným bodem. Konkrétně tedy budou prováděny kroky o pevné velikosti po obvodu celé hemisféry (od 0 do $2 \cdot \pi$) přičemž po každé takto provedené kružnici proběhne posunutí od vrcholu hemisféry směrem níže, k jejímu okraji (tedy od 0 do $\frac{1}{2}\pi$). Situace je ilustrována na obrázku 3.2. Takto získané samplovací vektory jsou pak použity k samplování z cubemapy (získané z vyfocení scény sondou), avšak získaný vzorek

je třeba dále vynásobit kosinem druhého zmíněného úhlu, jelikož světlo přicházející pod větším úhlem je slabší, a jeho sinem, protože čím více se vzorkování blíží vrcholu hemisféry, tím menší jsou vzorkované kružnice a tedy tím hustěji koncentrované jsou tam vzorky relevantní pro tím menší plochu. Všechny takto získané vzorky jsou sečteny a zprůměrovány a nakonec vynásobeny hodnotou $\pi$. Důvod násobení hodnotou $\pi$ vyplývá z řešení samplingu hemisféry pomocí metody Monte Carlo – vznikají dva Monte Carlo estimatory, každý pro jeden z úhlů. Situaci si lze prohlédnout v rovnici 3.1:

$$\begin{aligned} L_{diffuse}(v) &= k_d \frac{c}{\pi} \frac{2\pi}{N_1} \frac{\pi}{2N_2} \sum^{N_1} \sum^{N_2} L_i(\theta_i, \phi_i) cos(\theta_i) sin(\theta_i) \\ L_{diffuse}(v) &= k_d \frac{\pi c}{N_1 N_2} \sum^{N_1} \sum^{N_2} L_i(\theta_i, \phi_i) cos(\theta_i) sin(\theta_i) \end{aligned} \tag{3.1}$$

Přičemž $c$ a $k_d$ bude zpracováváno až během hlavní smyčky programu.

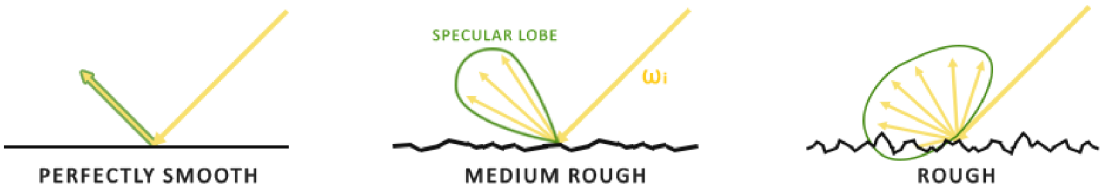
Při pozdějším samplingu takto vzniklé cubemapy pak bude pro získání difuzní složky IBL stačit pronásobení vzorku s barvou povrchu a poměrem difuzní složky ku složce spekulární. Na konec je třeba vzít v potaz, že kvůli diskretizaci integrálu je vlastně uskutečněna aproximace, a výsledek tak nebude zcela přesný. Zvolením menšího kroku pak přesnost je možné zvýšit, avšak na úkor výkonu.




Obrázek 3.2: $\phi$ značí úhel kolem dokola hemisféry, $\theta$ úhel mezi jejím vrcholem a krajem (obě postupně inkrementovány o krok pevné velikosti). Lze si povšimnout, že plocha připadající na vzorek se zmenšuje čím blíže je vrcholu hemisféry. Obrázek získán z: https://learnopengl.com/img/pbr/ibl_spherical_integrate.png

Pro předfiltrovanou cubemapu okolí byla také samplována hemisféra, avšak zatímco u předfiltrované cubemapy intenzity záření byly voleny vzorky dle pravidelného rozložení, zde by tento přístup nebyl příliš efektivní – spekulární odlesky odráží světlo ve směru odrazového vektoru povrchu, přesný směr se však může odlišovat na základě hrubosti – čím hrubší, tím více se vektor může odchýlit, viz obrázek 3.3, jelikož jeho mikroplošky odráží světlo tím různějšími směry. Lze si však ale teké povšimnout, že světlo, které se odráží směrem do kamery, je dohromady tvořeno všemi paprsky světla dopadajícími do daného bodu a že ony paprsky spadají do oblasti o analogickém rozsahu. Cílem tedy je, aby při běhu programu byla zapotřebí pouze míra hrubosti (určující která mipmapa bude samplována) a vektor získaný odrazem paprsku z kamery od daného povrchu, který ovzorkuje předfiltrovanou cubemapu a získá z ní tak už předpočítanou konvoluci světla z celé zmíněné oblasti. Pokud tedy bude

povrch mít nulovou hrubost, nebude možná žádná odchylka od odrazového vektoru a bude tedy získán perfektní odraz. Při vyšších hrubostech bude odraz čím dál rozmazanější. Je tedy zřejmé, že nemá smysl provádět sampling hemisféry mimo onu zmíněnou oblast. K samplování oblasti, kterou potřebujeme, tak využijeme techniku importance sampling.




Obrázek 3.3: Vyšší hrubost způsobuje, že mikroplošky mohou odrážet světlo z více směrů. Z toho plyne že vyšší hrubost znamená větší spekulární odlesk avšak o menší intenzitě. Obrázek získán z: https://learnopengl.com/img/pbr/ibl_specular_lobe.png

Implementace probíhá následovně: Nejdříve je vytvořena low-discrepancy sekvence, konkrétně pak tzv. Hammersleyho sekvence, která vrací dvousložkový vektor, kdy první hodnotou je číslo vzorku dělené jejich celkovým počtem (tedy pravidelné rozložení na ose $x$) a druhou hodnotou je zrcadlové otočení binární reprezentace čísla vzorku podle desetinné čárky, tedy např. jedenáctý vzorek s binární reprezentací 1011,0 bude mít hodnotu 0,1101, tedy 0,8125 v desítkové soustavě.

Druhým krokem je pak využití takto získaného vektoru ke generování vektoru $h$, tedy normály mikrogeometrie pro daný vzorek, která bude orientovaná zhruba podle očekávání vzhledem k dané míře hrubosti (čím vyšší hrubost, tím vyšší možná odchylka od normály povrchu). Při představě, v jaké oblasti je možné tyto vektory získávat, lze zpozorovat, že v podstatě vytvářejí kruh kolem normály povrchu a přes ně odražené paprsky tedy budou také tvořit kruh na hemisféře – kruh tvoří onu kýženou oblast, paprsky z níž se od povrchu odrazí ke kameře.

Ve třetím kroku pak byl vytvořen odraz view vektoru (tedy paprsku z kamery, v této části stejný jako normála povrchu) od vektoru $h$ a získán tak byl vektor $l$, tedy vektor od bodu dopadu ke zdroji světla. Ten pak byl použit pro sampling z cubemapy (získané vyfocením scény sondou) a získání vzorku. Samozřejmě, čím více jsou vektory $h$ odchýleny od normály povrchu, z tím větší oblasti bude samplováno a tedy tím různější budou vzorky, z čehož plyne vyšší rozmazání výsledného obrazu.

Vzorek pak byl pronásoben váhou, která je získána vektorovým součinem mezi normálou povrchu a vektorem $l$, kdy paprsky které mají menší vliv na výsledek budou mít menší váhu. Součet všech takovýchto vzorků pak byl podělen váhou celkovou. Tím byl získán výsledek konvoluce.

Předpočítaná BRDF textura, tedy textura vzorkována k získání výsledku BRDF, na ničem nezávisí – bude vždy vypadat stejně (tedy jako na obrázku 2.15) ať už je scéna jakákoli. Její vytvoření je pak velmi podobné vytvoření předfiltované cubemapy okolí – opět je vytvořena Hammersleyho sekvence a opět je využita k vytvoření vektoru $h$, který je opět ovlivněn hrubostí (hrubost zde ale bude hodnota texturové souřadnice na ose $y$, zatímco texturová souřadnice na ose $x$ vyjadřuje skalární součin normály a vektoru odraženého světla) a ze kterého je pak znovu, za přispění vektoru $v$ (spočítaný z texturové souřadnice $x$), vytvořen vektor $l$.

Poté se však přístup liší a postupováno je podle rovnice Cook-Torranceovy BRDF, pro kterou jsou nyní známy všechny potřebné proměnné až na kovovost. Kovovost je využita

pro výpočet $F_0$, které je třeba k výpočtu Fresnelovy funkce, tedy součásti BRDF. Řešením je se Fresnelovy funkce zbavit a přidat ji až za běhu hlavní smyčky programu, po samplingu zbytku BRDF z předpočítané textury.

Druhou část spekulární části IBL je třeba upravit následujícím způsobem (pro jednoduchost nejsou uváděny vstupy funkcí; podrobnější popis dostupný v článku Joey de Vriese [11], PBR – IBL – Specular IBL) (3.2):

$$\int_\Omega \frac{F \cdot G \cdot D}{4(n \cdot l)(n \cdot v)} \frac{F}{F} n \cdot l d\omega_i \tag{3.2}$$

Fresnelova funkce v čitateli zlomku BRDF se vykrátí. Zbývající Fresnelova funkce se rozepíše (3.3). $(1 - (h \cdot v))^5$ je pro jednoduchost substituováno symbolem $\alpha$.

$$\int_\Omega \frac{G \cdot D}{4(n \cdot l)(n \cdot v)} (F_0 + (1 - F_0)\alpha) n \cdot l d\omega_i \tag{3.3}$$

Tento zápis lze dále upravit do následujícího tvaru (3.4):

$$\begin{gathered}
\int_\Omega \frac{G \cdot D}{4(n \cdot l)(n \cdot v)} (F_0 + \alpha - F_0 \cdot \alpha) n \cdot l d\omega_i \\
\int_\Omega \frac{G \cdot D}{4(n \cdot l)(n \cdot v)} (F_0 \cdot (1 - \alpha) + \alpha) n \cdot l d\omega_i \\
\int_\Omega \frac{G \cdot D}{4(n \cdot l)} (n \cdot v)(F_0 \cdot (1 - \alpha)) n \cdot l d\omega_i + \int_\Omega \frac{G \cdot D}{4(n \cdot l)(n \cdot v)} (\alpha) n \cdot l d\omega_i \\
F_0 \int_\Omega \frac{G \cdot D}{4(n \cdot l)(n \cdot v)} (1 - (1 - h \cdot v)^5) n \cdot l d\omega_i + \int_\Omega \frac{G \cdot D}{4(n \cdot l)(n \cdot v)} (1 - h \cdot v)^5 n \cdot l d\omega_i
\end{gathered} \tag{3.4}$$

Výsledné dva integrály reprezentují scale a bias $F_0$ a jsou přesně tím, co je uloženo do červené a zelené barevné složky pro jednotlivé body předpočítané BRDF textury. Geometrická funkce je téměř stejná jako pro bodové zdroje světel, pouze se použije jiné $k$ ($\alpha$ zde reprezentuje hrubost) (3.5):

$$k = \frac{\alpha^2}{2} \tag{3.5}$$

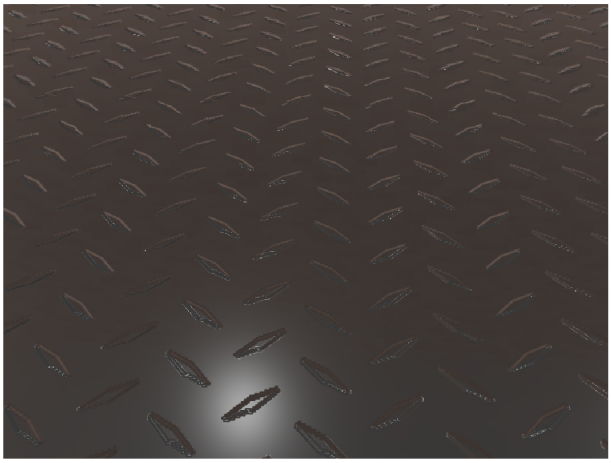
Normálová distribuční funkce se ztrácí po podělení rovnice funkcí hustoty pravděpodobnosti (vyplývající z importance samplingu a mající hodnotu $\frac{D \cdot (n \cdot h)}{4(v \cdot h)}$). Na závěr jsou hodnoty poděleny počtem samplovaných vzorků a uloženy do textury.

Výsledná spekulární složka IBL je získána samplingem vzorku z předfiltrované cubemapy okolí a jeho pronásobením $(F \cdot BRDF.r + BRDF.g)$ kde F je výstup Fresnelovy funkce a BRDF.r a BRDF.g jsou červená a zelená složka předpočítané BRDF textury pro dané $n$, $v$ a hrubost. Výsledek celého IBL pak je $k_d \cdot L_{diffuse} + L_{specular}$ kde $k_d$ je $1 - F$. Konečně, výsledkem celého osvětlovacího modelu pro daný bod je součet výsledného světla z IBL s výsledný světlem pro bodové zdroje světla a následná HDR a gamma korekce.

## 3.2 Normal mapping

Implementace normal mappingu této aplikace se od klasické trochu liší – jelikož není možné využívat externí obrázkové soubory, nelze normal mapy generovat v grafických softwarech pro to určených – je nutné je generovat v rámci programu, což však může být algoritmicky poměrně náročné. Zvolena proto byla trochu jiná varianta a místo klasické normal mapy

byla použita mapa výšková, tedy černobílý obrázek ve kterém čím světlejší je bod, tím vyšší polohu reprezentuje. Algoritmicky pak lze získat normály přechodů mezi tmavšími a světlejšími body a dosáhnout tak kýženého efektu. Takto vytvořená normal mapa sice nedokáže zaznamenat vše, co normal mapa normální, avšak mnohem snáze se definuje v rámci programu. Příkladem může být textura plechové podlahy na obrázku 3.4, jejíž vzorek je vytvořený právě tímto způsobem. Stejně je pak v programu tvořena i textura cihlové zdi.



Obrázek 3.4: Textura využívající techniku normal mappingu za použití výškové mapy. Důležitá není výška bodů, ale míra její změny mezi sousedními body, která definuje "strmost"normály.

Normála povrchu je spočítána pomocí vektorového součinu hran trojúhelníku a výpočet tangenty probíhá stejně jako v rovnici 2.15. Obě jsou následně pro body každého trojúhelníku poslány spolu s vertexy do vertex shaderu. Bitangentu není nutné počítat či ukládat, jelikož musí být vždy kolmá na rovinu tvořenou tangentou a normálou povrchu, není tedy problém ji v shaderu dopočítat pomocí vektorového součinu. Stavba TBN matice je tedy zřejmá, zajímavější však je získání normály z výškové mapy.

Jelikož jsou zajímavé především přechody mezi výškami (pokud bude výška všech okolních bodů stejná, bude normála zkrátka ukazovat ve směru osy $z$ ať už je velikost výšky jakákoli), je nutné při výpočtu normály vzít v úvahu okolní body, tedy celkem čtyři hodnoty pro body vpravo, vlevo, nad a pod zpracovávaným bodem. Z takto získaných hodnot jsou pak vytvořeny dva vektory: jeden kladný ve směru osy $x$ a s hodnotou pro osu $z$ získanou odečtením hodnoty levého bodu od pravého, druhý kladný ve směru osy $y$ a s hodnotou pro osu $z$ získanou odečtením hodnoty bodu dole od hodnoty bodu nahoře. Vektorový součin těchto dvou vektorů je pak hledanou normálou v tangent space pro zpracovávaný bod. Pronásobením této normály maticí TBN je pak získána normála ve world space, která je pak dále využívána v rámci PBR.

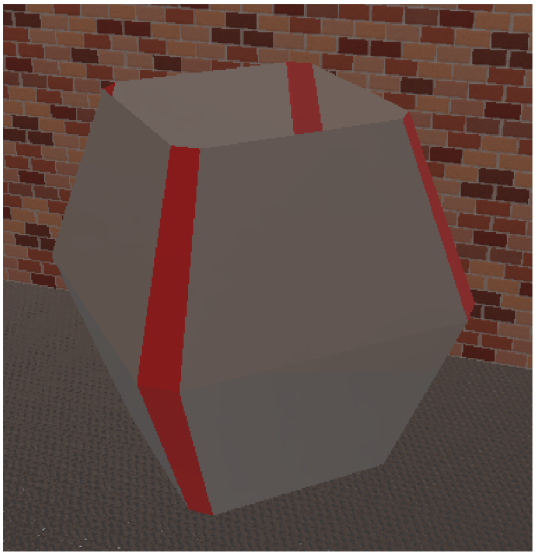
## 3.3 Šablonování

Aplikace využívá jednoduchou formu šablonování po šroubovici pro vytváření válcovitých těles tvořených pouze pláštěm, nikoli však podstavami. Situaci lze ilustrovat na příkladu: nechť je rovina tvořena osami $x$ a $z$. Do této roviny nechť je zasazen bod s nadefinovanou vzdáleností od počátku na ose $x$. Tímto bodem je pak proložena kružnice, na které leží

nadefinovaný počet bodů (výsledný "válec"tedy bude mít půdorys mnohoúhelníku). Poté se provede přesun o nadefinovanou vzdálenost výše po ose $y$ a proces se zopakuje s nově definovaným poloměrem pro kružnici. Tento jednoduchý algoritmus je pak dále upraven tak, aby body definovány na kružnici nebyly rozloženy rovnoměrně po jejím obvodu, ale aby sudé body měly negativní offset a liché pozitivní – budou se tedy vytvářet dvojice blízkých bodů, které vytvoří dojem zkosených hran. Příklad takto vytvořeného tělesa si je možné prohlédnout na obrázku 3.5.

Algoritmus pracuje ve dvou fázích – v první jsou vytvořeny body meshe, v druhé jsou tyto body využívány k tvorbě trojúhelníků pro rendering. Součástí algoritmu je i výpočet texturovacích koordinátů.

Tento algoritmus byl použit pro tvorbu nohou, kolen, paží, loktů, krku a torza robota, čímž bylo dosaženo značného usnadnění práce a úspory místa. Algoritmus nelze použít na objekty nepravidelné a nevytváří podstavy, takže tělesa jako např. chodidla či ruce jsou definována manuálně.



Obrázek 3.5: Útvar vytvořený šablonováním. K vytvoření stačilo jen nadefinovat počet bodů podélně a po obvodu a určit velikosti poloměrů pro jednotlivá "patra".

## 3.4 Kosterní animace

Kosterní animace byla využita pro stavbu a rozpohybování robota – jeho chůzi kupředu chodbou – a pro pohyb kamery. Povrch byl vytvořen kombinací ručně nadefinovaných těles a techniky šablonování zmíněné výše.

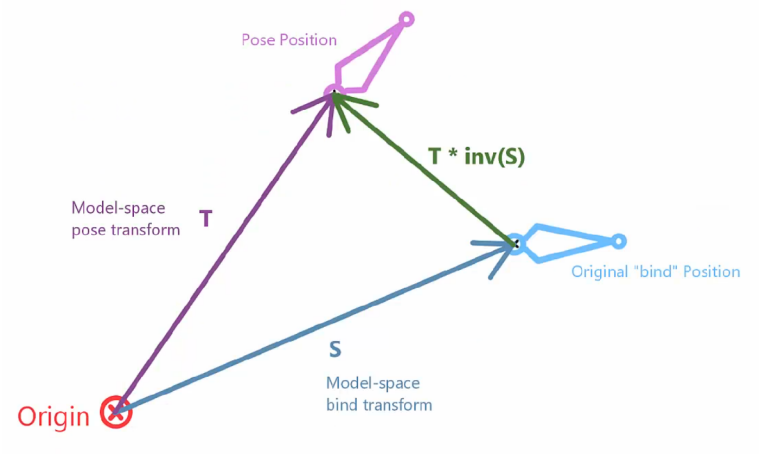
Jakožto hlavní stavební prvek kostry byly využity klouby, přičemž každý obsahoval svou pozici, rotaci (implementovanou pomocí kvaternionů) a pole ukazatelů na své potomky (ukazatel na rodiče není potřeba). K vertexům povrchu pak byla přidána možnost být ovlivněné až třemi kostmi (tedy tři integery obsahující ID kloubů) a tedy i třemi vahami (tři floaty). Na paměti je třeba mít, že součet všech vah pro jeden vertex se musí rovnat jedné, jinak se výsledná animace nebude chovat dle očekávání. ID i váhy byly spolu s dalšími informacemi (pozice, texturovací souřadnice atd.) poslány do vertex shaderu.

Ze všeho nejdříve byla z kloubů vytvořena hierarchie – chodidlo je potomkem holeně, která je potomkem kolena, které je potomkem stehna a tak dále. V podstatě každé těleso

tvořící robota má svůj kloub, přičemž ta, která budou ohýbána (lokty a kolena), mají ještě kloub pomocný, pomocí něhož jsou deformována místa ohybu. Další dva klouby pak náleží kameře.

Každý klíčový snímek je tvořen časovou značkou a polem transformačních matic. Tyto transformační matice nejsou nic jiného než matice získané z pozice a rotace každého kloubu, tedy klíčový snímek je v podstatě snímkem celé kostry v daném čase. Množství těchto snímků pak závisí na délce a složitosti animace. Tento program jich má přes 40 – celý jeho průběh je jedna velká animace, od otáčení kamery kolem robota po jeho chůzi.

Pro získání stavu kostry v daném čase byly vytvořeny třídy *animation* a *animator*. Třída *animation* uchovává celkovou délku animace a pole všech jejích klíčových snímků. Třída *animator* na základě momentálního času animace klíčové snímky interpoluje. Samotná interpolace je pak triviální. Takto získané matice však každá určuje pouze svou pozici a rotaci vůči svému rodiči, nikoli vůči celému modelu – k získání transformačních matic potřebných k aplikaci na vertexy povrchu je třeba nejdříve rekurzivně pronásobit matice potomků jejich rodiči (postupuje se od kořenového kloubu níže). Takto získaná matice však uvažuje translaci a rotaci vůči počátku modelu, což ale nemusí být místem, ve kterém se kloub na začátku nachází – je tedy třeba ho nejdříve posunout zpět do počátku a a poté ho přesunout pomocí výsledné matice. Jinak řečeno, je třeba vynásobit výslednou matici inverzní maticí specifikující počáteční stav kloubu před aplikováním animace. Celou problematiku ilustruje obrázek 3.6. Takto získané matice pak už stačí jen v každém snímku poslat do vertex shaderu jakožto *uniform*.




Obrázek 3.6: *Origin* je počátek modelu, *pose position* ilustruje stav, do kterého se kloub má dostat, *original "bind"position* je stav kloubu před aplikováním animace, $T$ je výsledná matice a $S$ matice inverzní. Obrázek získán z: https://youtu.be/cieheqt7eqc?t=718

V samotném vertex shaderu je pak aplikace těchto matic na vertexy triviální – pozice každého je zkrátka vynásobena maticí příslušné kosti a její váhou, přičemž pokud je kostí více, je tato operace provedena pro každou zvlášť a jejich výsledky jsou sečteny. Stejným způsobem jsou poté pronásobeny i normála a tangenta vertexu.

Animaci je třeba v každém snímku aktualizovat zvýšením animačního času, přičemž pokud tento čas přesáhne maximální čas animace, animace se resetuje. V tomto programu je toto resetování znemožněno, jelikož by nevytvářelo pozitivní dojem.

V průběhu animace může v některých místech docházet k deformaci tělesa - stává se to pokud jsou různé vertexy tělesa ovlivněny jinými kostmi či jinými vahami. V tomto programu k deformaci dochází v oblastech loktů a kolen, jak lze vidět například na obrázku 3.7. Jedná se o deformaci úmyslnou, díky které celý pohyb vypadá reálněji. Deformaci lze samozřejmě vylepšit – pokročilé animace například konzervují objem deformovaných těles a ohýbaná část tak při deformaci tloustne (modelují tak např. akumulaci masa v loktu při ohýbání ruky) apod.

Obrázek 3.7: Původní tvar tělesa je deformován působením různých kostí na jeho jednotlivé vertexy.

Zvlášť je pak v animaci řešena kamera – její pohyb je činěn kosterní animací, a to sice pomocí dvou kloubů, které oba rotují protichůdnými směry. Druhý kloub je potomkem prvního a jeho pozice a rotace jsou shodné s pozicí a rotací kamery. Jelikož jsou však různé parametry kamery potřebné pro několik různých shaderů, bylo by její transformování stejným způsobem jako vertexů problematické. Místo toho jsou z transformační matice pro kameru posílané do shaderu extrahovány její translace a rotace, které jsou pak aplikovány na kameru mimo shader. V shaderu již s kamerou není dále manipulováno.

Samotná animace začíná obkroužením robota kamerou, přičemž se pohybuje od chodidel vzhůru až k hlavě. Poté se kamera přesune do míst, kde by robot měl oči – na scénu je tedy nahlíženo z pomyslného úhlu pohledu robota. Následně se postupně rozsvěcují světla chodby, kamera se sklání dolů a je možné pozorovat robotovu chůzi. Po chvíli kamera sjede do oblasti robotových nohou a opět je obkrouží, aby ukázala animaci chůze. Pak se opět vrací do pohledu robota. Cílem je demonstrovat dva hlavní aspekty programu, tedy implementaci kosterní animace a Physically Based Rendering.

## 3.5 Textury

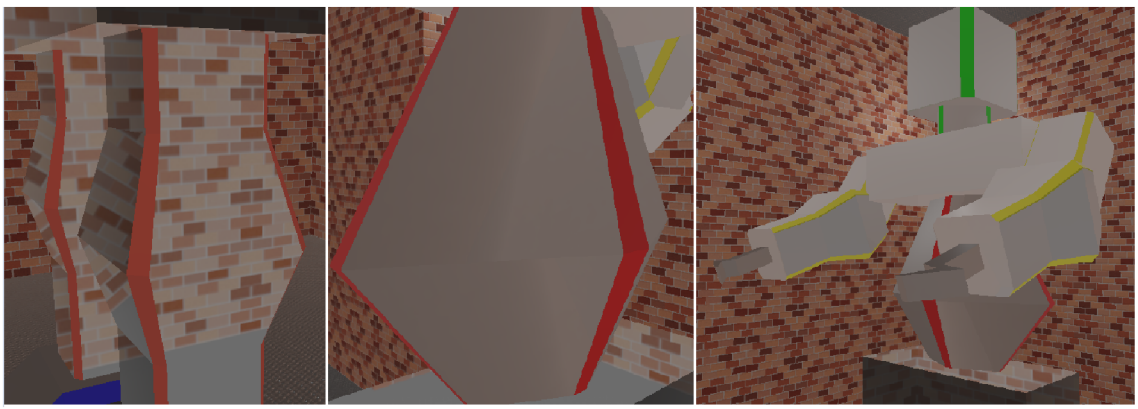
Veškeré textury v programu jsou implementovány s použitím PBR materiálů a generovány za běhu programu – využití obrázkových souborů nepřipadá vzhledem k velikostnímu omezení programu v úvahu. Každá výsledná textura se tedy ve skutečnosti skládá ze čtyř materiálů – albeda určujícího barvu povrchu, kovovosti, hrubosti a normal mapy (implementované pomocí výškové mapy).

Textura plechové podlahy (a stropu) se vyznačuje vysokou mírou kovovosti a především pak užitím normal mappingu pro získání svého vzorku, prezentovaného na obrázku 3.4. Tvar vzorku bylo třeba kreslit doslova pixel po pixelu (při dostatečném přiblížení je dokonce možné vidět jeho aliasing. Ten je možné zmírnit vyšším rozlišením textury, avšak vzhledem k tomu, že kamera na vzorek z takové blízkosti v průběhu programu nikdy nezacílí, nebyl tomu důvod), načež byl periodizován a zrcadlově obrácen v závislosti na tom, v jaké se nacházel řadě.

Textura cihlové zdi, kterou si lze prohlédnout na obrázku 3.8, pak také využívá efektů normal mappingu, ačkoli to kvůli vysoké hrubosti a barevným změnám není tak patrné jako u vzorku plechové podlahy – jednotlivé cihly jsou vystouplé oproti spárám. Navíc na ně byl aplikován šum pomocí generátoru pseudonáhodných čísel, který na jejich povrchu vytvořil různé malé důlky – cihly tak vypadají lépe a opravdověji.

Obrázek 3.8: Malé kazy dodávají zdi dojem hrubosti.

Textury robota normal mappingu nevyužívají; místo toho mají různé kombinace kovovosti a hrubosti s cílem ukázat jejich vliv na výsledné chování a podobu textur. Pruhy zkosených hran získaly jinou barvu pro lepší estetickou kvalitu výsledku. Na obrázku 3.9 si lze textury robota prohlédnout.



Obrázek 3.9: Různé části robota mají různé hodnoty kovovosti a hrubosti.

## 3.6 Model

Robot je složen z mnoha různých těles, která jsou vytvořena buď manuálně (ta složitější) nebo pomocí šablonování. Pomocí L-systému je pak definováno množství těles, která mají být renderována. Prostřednictvím kloubů kosterní animace jsou tato tělesa poskládána do patřičných pozic a póz pro jednotlivé klíčové snímky (pozicování objektů bylo možné také udělat pomocí L-systémů – jsou na to plně připraveny – avšak použití nástrojů kosterní animace se ukázalo být implementačně jednodušší). Použitý L-systém je plně deterministický a robot tak bude vypadat vždy stejně. Výsledný model robota si lze prohlédnout na obrázku 3.10.

Místnost byla vytvořena manuálně.

Obrázek 3.10: Model robota v prvním klíčovém snímku.

# Kapitola 4

# Implementace

Program byl vytvořen v jazyce C++ a GLSL (pro shadery) v prostředí Microsoft Visual Studio Express 2013 a využívá knihovnu OpenGL. Program si neklade za cíl být multiplatformní – cílová platforma je počítač s operačním systémem Microsoft Windows s grafickou kartou podporující OpenGL verze 4.6.

## 4.1 Velikost programu

Dle zadání je maximální povolená velikost výsledného spustitelného souboru 64 kB. Tato norma byla splněna. Velikost spustitelného souboru je před užitím exe packeru sice přes 118 kB, avšak po kompresi zabírá pouhých 32 256 bajtů, tedy téměř polovinu maximální povolené velikosti. Ke kompresi spustitelného souboru byl využit exe packer kkrunchy of Fabiana Giesena [5], který je navržen právě pro intra do 64 kB.

# Kapitola 5

# Závěr

Cílů projektu bylo úspěšně dosaženo – aplikace implementuje některé pokročilé grafické techniky v krátkém grafickém intru a její spustitelný soubor je menší, než 64 kB.

Výslednou aplikaci považuji za zdařilou – implementaci kosterní animace a především pak physically based rendering považuji za poměrně náročná témata, obzvláště s přihlédnutím k nemožnosti využít většinu knihoven a externích souborů, avšak obojí bylo nejen implementováno úspěšně, ale i s velkou rezervou co se týče maximální povolené velikosti spustitelného souboru.

Na druhou stranu uznávám, že aplikace není dokonalá a v mnohém by se dala dále vylepšit – ona velká rezerva totiž také znamená, že by se do aplikace daly vměstnat ještě další techniky. Intro samotné by pak mohlo být delší a zajímavější – v této podobě slouží spíše k ukázání implementovaných technik, avšak pokud by skutečně bylo míněno pro nějakou soutěž, potřebovalo by mít také lepší děj a vyšší důraz na estetickou kvalitu (např. modelování robota se složitějšími tvary, propracovanější textury s více přidanými kazy, osazení místnosti různými dalšími objekty atp.). Rozhodně tedy existuje prostor pro možný další vývoj této aplikace.

Celý projekt považuji především za významnou praktickou zkušenost v rámci počítačové grafiky, kde byly aplikovány mnohé poznatky a koncepty se kterými jsem se setkal během svého studia a kde jsem si rozšířil své obzory.

# Literatura


[1] Alamia, M.: *Article - Physically Based Rendering - Cook–Torrance.* [Online; navštíveno 13.05.2019].
URL http://www.codinglabs.net/article_physically_based_rendering_cook_torrance.aspx

[2] Beeson, C.: *NVIDIA – GPU Gems – Chapter 4. Animation in the "Dawn"Demo.* [Online; navštíveno 13.05.2019].
URL https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch04.html

[3] Ústav matematiky fakulty strojního inženýrství VUT v Brně: *VUT Brno – Fakulta strojního inženýrství – ústav matematiky – Matematika online – Šablonování a pole.* [Online; navštíveno 13.05.2019].
URL http://mathonline.fme.vutbr.cz/pg/Dcad/08_Sablonovani.htm

[4] Dickeau, R. M.: *Two-dimensional L-systems.* [Online; navštíveno 13.05.2019].
URL http://mathforum.org/advanced/robertd/lsys2d.html

[5] Giesen, F.: *kkrunchy: pretty good executable compression.* [Online; navštíveno 13.05.2019].
URL http://www.farbrausch.de/~fg/kkrunchy/

[6] Hoffman, N.: *Background: Physics and Math of Shading.* 2012, [Online; navštíveno 13.05.2019].
URL https://blog.selfshadow.com/publications/s2013-shading-course/hoffman/s2013_pbs_physics_math_notes.pdf

[7] Karis, B.: *Unreal Engine – Real Shading in Unreal Engine 4.* [Online; navštíveno 13.05.2019].
URL https://cdn2.unrealengine.com/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf

[8] Meiri, E.: *OGLdev – Tutorial 38: Skeletal Animation With Assimp .* [Online; navštíveno 13.05.2019].
URL http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html

[9] Prunier, J.-C.: *Scratchapixel 2.0 – Monte Carlo Methods in Practice – Variance Reduction Methods: a Quick Introduction to Quasi Monte Carlo.* [Online; navštíveno 13.05.2019].
URL https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice/

[10] Shaker, N.; Togelius, J.; Nelson, M. J.: *Procedural Content Generation for Games (Computational Synthesis and Creative Systems)*. Springer, 2016, ISBN 978-3319427140.

[11] de Vries, J.: *PBR – Theory*. [Online; navštíveno 13.05.2019].
URL https://learnopengl.com/