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

POROVNÁNÍ STÍNOVÝCH METOD

COMPARISON OF SHADOW METHODS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARCEL KISS**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL TÓTH**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Kiss Marcel**
Obor: Informační technologie
Téma: **Porovnání stínových metod**
Comparison of Shadow Methods
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte knihovnu OpenGL a její nadstavby.
2. Nastudujte metody tvorby stínů v 3D grafice
3. Navrhněte aplikaci pro předvedení a změření nastudovaných metod
4. Implementujte navrženou aplikaci
5. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu
6. Vytvořte video pro prezentování projektu.

Literatura:

- Podle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3
- Funkční prototyp aplikace

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Tóth Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá porovnávaním techník vrhania tieňov objektov v rámci scény. V teoretickej časti popisuje a porovnáva možné riešenia vrhania tieňa a podrobnejšie postupy metód Shadow Mapping a Shadow Volumes, ktoré patria medzi najpoužívanejšie techniky tieňovania v reálnom čase. Hlavnou časťou je návrh a implementácia týchto dvoch tieňových metód s využitím knižnice OpenGL. V časti merania porovnáva podľa grafov s nameranými hodnotami a na záver zhodnotenie výsledkov.

Abstract

This thesis talks about comparison of shadow casting techniques within a scene. In the theoretical part, it describes and compares possible solutions of shadow casting and more detailed about Shadow Mapping and Shadow Volumes, which are among the most commonly used real-time shadowing techniques. The main part is about design and implementation of these two shadow methods using the OpenGL library. In the measurement part it compares methods based on measured values. The outcome of my measurements can be found in the final part of my thesis.

Kľúčové slová

tieň, tieňová pamäť hĺbky, hĺbková textúra, tieňové telesá, C++, OpenGL, 3D, grafika, meranie, porovnávanie, textúra, antialiasing, projekcia, scéna, vykresľovanie

Keywords

shadow, shadow mapping, depth texture, shadow volumes, C++, OpenGL, 3D, graphics, measurement, comparison, texture, antialiasing, projection, scene, rendering

Citácia

KISS, Marcel. *Porovnání stínových metod*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Tóth Michal.

Porovnání stínových metod

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Michala Tótha. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Marcel Kiss
15. mája 2017

Poďakovanie

Ďakujem p. Ing. Michalovi Tóthovi za ochotu a odbornú pomoc na konzultáciách. Ďakujem za praktické rady pri riešení problémov počas tvorenia aplikácie.

Obsah

Kapitola 1

Úvod

Realistické vykresľovanie scény počítačom je bežným spôsobom ako tvoriť filmy, reklamy alebo obrázky bez toho aby scéna mala reálny podklad. Najčastejšie je to simulovanie reálneho sveta z pohľadu obrazu a napodobňovania fyzikálnych vlastností svetla v umelo vytvorenej scéne. Aby bol výsledok čo najdôveryhodnejší, je nevyhnutným efektom svetla jeho nedostatok v podobe tieňa. Tento jav je veľmi dôležitý pre vnímanie trojrozmerného priestoru a dodáva pocit reality. Okrem lokálneho osvetlenia modelu je nutné vyriešiť vrhanie tieňa pevných objektov do scény, kde je výsledný tieň ovplyvnený transformáciou objektov a samotného zdroja svetla.

Existujú rôzne techniky ako riešiť vrhanie tieňa v počítačovej grafike. Táto práca opisuje výhody a nevýhody metód zo širšieho obzoru riešení. Každá technika je vhodná na iný druh účelu v závislosti na prioritách aplikácie. V praxi sú najčastejšie využívané techniky tieňovania Shadow Mapping a Shadow Volumes, na ktoré sú dostatočne efektívne na vrhanie tieňa v reálnom čase.

Návrh aplikácie využíva ako základ štruktúry knižnicu OpenGL, podporovanú veľkou časťou platforiem na trhu. S využitím jej možností a rozšírení, práca popisuje návrh a implementáciu demonštračnej aplikácie. Primárny cieľ je vizuálne porovnávať a numericky merať techniky rastrového a vektorového tieňovania realistickej scény s predpokladom bodového svetla. Aplikácia dokáže merať čas vykresľovania farebnej a tieňovej fázy v rôznych pozíciách scény a rôznymi parametrami tak aby bolo meranie účinné.

V implementačnej časti je popis riešenia problémov pomocou knižníc použitých v aplikácii. Taktiež sú spomenuté niektoré problémy, na ktoré som narazil, ich príčiny a riešenia. Z nameraných hodnôt sú v časti merania vyhotovené grafy a z nich vyhodnotenia. Záver práce zhodnotí jednotlivé techniky na základe meraní.

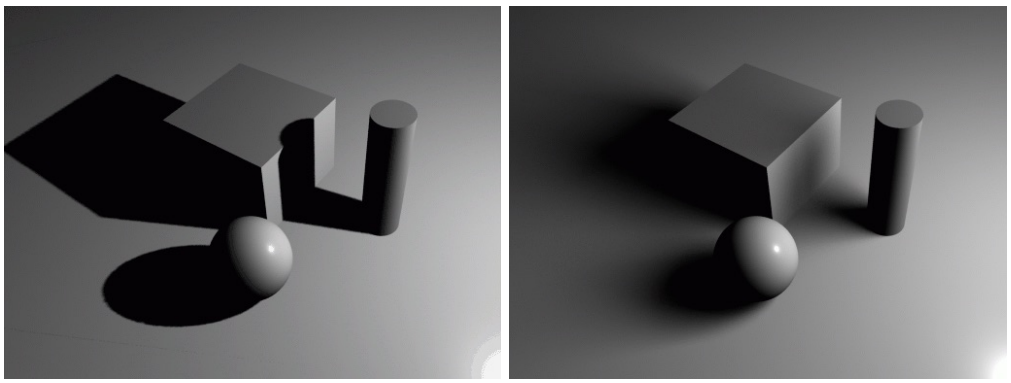
Kapitola 2

Teoretické a analytické riešenia vrhania tieňov

Táto kapitola popisuje ako je možné dosiahnuť výsledok vrhnutého tieňa do scény z pohľadu teórie. Vybrané sú niektoré najčastejšie využívané, významné nosné techniky, ktorých algoritmy sú popísané len okrajovo no najviac pozornosti majú Shadow Mapping a Shadow Volumes.

2.1 Vrhanie tieňa

Vo vykresľovaní umelých realistických scén sa často zameriava na tiene. Bez tieňov by výsledok nebol veľmi dôveryhodný, pripadal by plochý a nereálny, či už ide o statický obraz alebo video. Zatienenie predstavuje zamedzenie lúčov priameho svetla zo zdroja svetla. Rozlišujú sa dva druhy tieňa a to je vlastný *(self shadow)* a vrhnutý *(cast shadow)* [8]. Vlastný tieň objektu sú stmavnutá časť objektu odvráteného od svetelného zdroja, kde naň nedopadá žiadne priame svetlo. Vrhnutý tieň závisí od tvaru objektu, vzdialenosti a typu zdroja svetla.

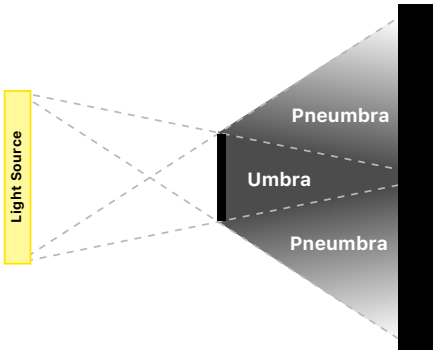


Obr. 2.1: **Vľavo** je ukážka tvrdých tieňov *(hard shadows)*.
Vpravo je ukážka mäkkých tieňov *(soft shadows)*. [1]

Zdroj svetla môže byť bodový, kde všetky lúče svetla vychádzajú z jedného bodu a priamočiaro dopadajú na povrch objektu a tým vytvárajú tvrdé vrhané tiene *(hard shadows)*

[1] Zdroj obrázkov: renderman.pixar.com

(viz 2.1 vľavo). Bodové zdroje sa však v reálnom svete nevyskytujú [8], reálnejšie typy svetla sú plošné, pri ktorých lúče vychádzajú z objemu, čiže na povrch objektu dopadajú z rôznych smerov a tým vznikajú mäkké vrhané tiene *(soft shadows)* (viz 2.1 vpravo).



Obr. 2.2: Vizualizácia mäkkých tieňov *(soft shadows)* vzniknutých z plošného zdroja svetla.

Mäkké tiene majú plynulý prechod v rozhodovacej hranici. Časť, ktorá je úplne zatienená ako je naznačené na obrázku sa nazýva hlavný tieň *(umbra)* rozotrená časť tieňa sa nazýva polotieň *(pneumbra)* (2.2, 2.1 vpravo) [8]. Veľkosť pneumbry tieňa sa mení v závislosti od vzdialenosti a uhla zdroja svetla k objektu v scéne.

2.2 Metódy vrhania tieňov

Techniky vrhania tieňa sa z história časom vyvíjali. Vyvíjal sa aj pohľad ako sa na tento problém v počítačovej grafike pozerať a vznikli tak techniky, ktoré sa od základov líšia. Postupne sa vylepšovali a optimalizovali aby bol výsledok čo najpresnejší, najrealistickejší a zároveň rýchly. Každá technika má svoje výhody ale prinášajú aj nevýhody, ako napríklad s kvalitným tieňom prichádza problém s náročným a pomalým tieňovacím procesom. Pokrok výkonu počítačov a grafickej karty sa neustále vyvíja, no časom sa zvyšujú nároky na kvalitu a kvantitu tieňovaných objektov. Preto je potrebné pri tvorbe grafického softvéru vybrať vhodnú techniku tak, aby sa naplnili priority aplikácie.

Obr. 2.3: **Naľavo** je zobrazená scéna s veľmi detailným tieňovaním. **Napravo** je ukážka veľmi jednoduchých pseudo-tieňov. Prioritou pseudo-tieňov je rýchle zorientovanie v priestore, v tomto prípade pozícia lietajúcich objektov. [2]

[2]Zdroj obrázkov: www.tobias-franke.eu, www.moddb.com

V niektorých prípadoch je preferovaná kvalita a detaily tieňovania, na náročnosti, čiže na čase vyhotovenia obrazu nezáleží, ako napríklad v softvéroch na tvorbu grafiky alebo videí (viz 2.3 naľavo). Na druhej strane, často hry s 3D grafikou, preferujú plynulosť na úkor kvality, ktorá je druhoradá. Na obrázku 2.3, napravo je názorná ukážka veľmi jednoduchých, nereálnych tieňov, ktorých podstata je len v rýchlom orientovaní sa v trojrozmernom priestore a odhad pozície lietajúcich objektov. Takéto nereálne tiene nie sú veľmi dôveryhodné, preto spomeniem viac realistickejšie techniky také, ktoré zachovávajú tvar objektu a dokážu spracovať akúkoľvek scénu.

Ray Tracing

Technika Ray Tracing *(stopovanie lúča)*, alebo Ray Casting *(vrhanie lúčov)*, je motivovaná myšlienkou simulovania svetla vo fyzickom smere ako sa správa v reálnom svete. Náš dnešný model hovorí, že svetlo sú častice tak ako aj vlnenie. Táto metóda sa zameriava na časticovú vlastnosť, čiže fotóny. Fotóny sú vyžarované svetelným zdrojom, odrazené, lomené a prenášané plochami modelu a nakoniec vnímané naším okom. Tieto fotóny nasledujeme po ich trajektóriách, ktoré voláme lúče *(rays)* [2].

Jednotlivé objekty majú definovaný materiál, z ktorého pozostávajú. Materiál nesie informácie o tom, ako sa má prichádzajúci lúč svetla zdeformovať, či už farebnou zložkou alebo smerom odrazu. Zohľadnenie lámania a odrážanie svetla tvorí v konečnom výsledku mäkké tiene *(soft shadows)*. Farba materiálu definuje pohlcované farebné zložky svetla a tým odrazené svetlo už bude mať len zlomok pôvodnej intenzity. Najväčším problémom pri vrhaní lúča je, že väčšina lúčov vyžarovaných svetlom sa nikdy nedostanú do oka (kamery).

Ak fotón narazí na plochu, musí sa rozhodnúť kam bude cez scénu trajektória letiaceho fotónu pokračovať. V technike Ray Tracing je to riešené pravdepodobnostnou distribučnou funkciou. Táto funkcia popisuje ako veľmi sa svetlo odráža, láme alebo prenáša zo špecifického smeru [2]. Výsledky tejto techniky sú veľmi realistické avšak kvalita záleží od počtu vyžarovaných lúčov, pričom veľké množstvo lúčov zaťažuje grafický výkon. Keďže tento typ techniky vrhania tieňov produkuje veľmi detailné osvetlenie scény, je vhodný napríklad na generovanie statických obrazov, kde nezáleží na rýchlosti produkcie výsledku.

Obr. 2.4: Výsledky použitia metódy Ray Tracing. [3]

Kvalita obrazu však ide na úkor výkonu a teda aj času. V programoch, kde čas vykresľovania hrá dôležitú úlohu môže nastať problém. Táto technika umožňuje zvoliť počet

[3] Zdroj obrázkov: wikimedia.org, web.cs.wpi.edu

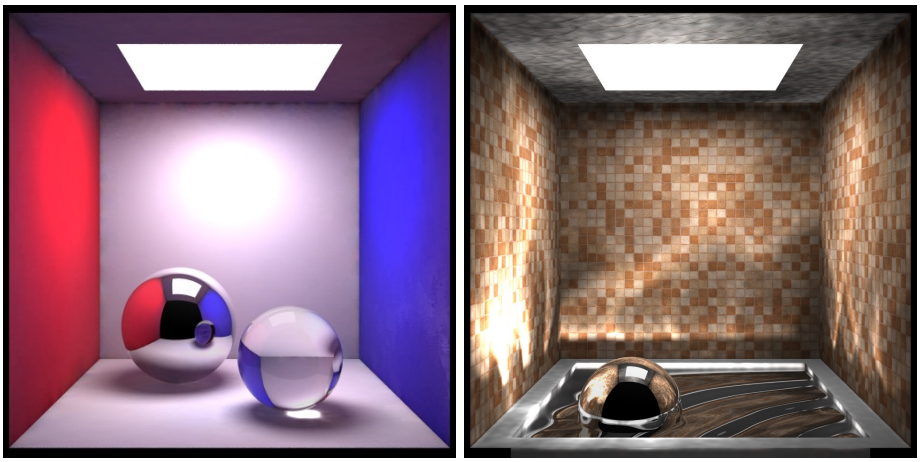
sledovaných lúčov. Vždy sa zvolí taký počet lúčov aby bol výsledok dostatočne detailný, no pri náročných scénach môže ten istý počet výrazne spomaliť *framerate*. V týchto prípadoch sa môže zvoliť menší počet lúčov za jedno prekreslenie, čo spôsobí zrnitý obraz, ktorý sa pri nehybnej kamere ďalšími prekresleniami zaostrí [8].

Photon Mapping

Globálne osvetľovacie algoritmy majú dlhú históriu v počítačovej grafike od skorých výskumov založených na rádiozity a Monte Carlo ray tracing, po novšie algoritmy ako sú Photon Mapping [11]. Photon Mapping je v súčasnosti najúspešnejší prístup spájajúci rádiozitu *(radiosity)* a sledovanie lúčov *(ray tracing)* [12].

Rádiozita *(radiosity)* je globálne osvetlenie, ktoré sa rieši rovnicou rádiozity. Jedným zo spôsobov je rozdelenie každej plochy do niekoľkých častí *(patches)* a pre každú sa počíta faktor viditeľnosti *(view factor)*, čo je faktor ako veľmi je viditeľná pre ostatné plochy. Plochy, ktoré sú od seba vzdialené, majú faktor menší. Veľmi vzdialené plochy majú faktor rovný alebo blížiaci sa nule. Vo vykresľovaní je tento faktor použitý pre vypočítanie intenzity svetla. Ďalším spôsobom je vrhanie rádiozity *(shooting radiosity)*, kde je rovnica vyriešená vrhaním svetla z každej pod-časti plôch vo forme energie. V prvom kroku sa osvetlia iba tie plochy, ktoré sú priamo vo výhľade. V ďalších krokoch sa svetlo odráža na ďalšie a ďalšie plochy, čím sa osvetlí celá scéna.

Technika Photon Mapping je zameraná na globálne osvetlenie, čo dáva pocit nerozoznania od reality. Prostredie je rozdelené do rovnomerne rozložených častí a simulovaná vyžarovaná energia sa prenáša medzi nimi až kým nie sú energeticky vyrovnané. [7]



Obr. 2.5: Príklad výsledku použitia techniky Photon Mapping. [4]

Photon Mapping je jedna z najširšie používaných algoritmov, pretože je veľmi praktická a umožňuje efektívne skonštruovať plné riešenie globálneho osvetlenia. Je to dvoj-kroková technika, v ktorej prvý prechod spočíva v stopovaní fotónov *(photon tracing)* cez scénu a zaznamenávanie ich interakciu s elementmi v scéne do dátových štruktúr, fotónových máp *(photon map)*. Táto fotónová mapa je použitá počas druhého prechodu, renderovacieho kroku, na odhad difúzneho nepriameho osvetlenia [11].

Pred tým ako môže byť fotónová mapa zhotovená, fotóny musia byť vyžarované do scény. Proces stopovania lúčov oka *(tracing eye)* a stopovania fotónov *(tracing photons)* zo svetla

[4] Zdroj obrázkov: web.cs.wpi.edu

je veľmi podobný. Najdôležitejším rozdielom je, že pri každej interakcii je fotón uložený a ďalší vyžarovaný. Podobne ako pri stopovaní odrazených lúčov, zaberie preskakovanie fotónov niekoľko vykresľovacích prechodov na rozšírenie fotónov po celej scéne [11].

Shadow Mapping

Metóda tieňovej mapy *(shadow mapping)* je rastrová metóda využívajúca textúru ako úložisko zatieneného priestoru. Generuje tvrdé tiene s využitím zapísania tých plôch, ktoré sú viditeľné svetlom. S použitím pozície a smeru svetla, ako zdroja pohľadu, sa vyrenderuje scéna do tieňovej mapy *(shadow map)*. Tieňová mapa neukladá farebnú zložku obrazu ale iba hĺbkové hodnoty jednotlivých pixelov. Táto hĺbková mapa sa následne použije vo farebnom vykresľovaní na obrazovku z pohľadu pozorovateľa s riešením viditeľnosti algoritmom *z-buffer* [8].

Pixely *(fragmenty)* hĺbkovej mapy tohoto obrazu sa transformujú do pohľadu zo svetla a táto vzdialenosť sa porovná so súradnicou z v pamäti hĺbky. Ak je pixel z pohľadu kamery po transformácii ďalej než z pohľadu svetla, je v tieni a jeho intenzita osvetlenia sa zníži [8].

Algoritmus tieňovej mapy vyzerá nasledovne [8]:

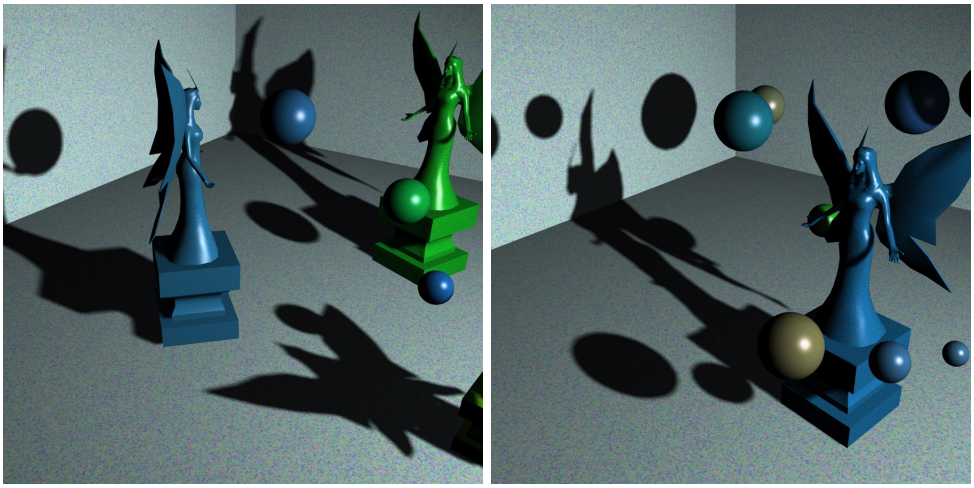
1. Zobraz scénu z pohľadu svetelného zdroja L_i a hodnoty z pamäti hĺbky (z-buffer) ulož do mapy H_i.
2. Zobraz scénu z pohľadu kamery pomocou pamäti hĺbky.
3. Pre všetky pixely $[u, v]$ (s hĺbkou w) zobrazené scény vykonaj:
   (a) Preveď bod $[u, v, w]$ do sústavy súradníc zdroja svetla L_i a získaj tak jeho nové súradnice $[x, y, z]$.
   (b) Vyber hodnotu hĺbky z vygenerovanej hĺbkovej mapy H_i; $A = H_i[x, y]$
   (c) $B = z$
   (d) Pokiaľ $(A < B)$, tak je pixel $[u, v]$ v tieni, inak je zdrojom L_i osvetlený.

Pokiaľ je zdrojov viac, vytvorí sa v kroku 1. viac hĺbkových máp pre každý zdroj. Kroky (3a) až (3d) v algoritme sa opakujú pre každú uloženú hĺbkovú mapu [8].

Výhodou tejto techniky je, že v niektorých prípadoch pokiaľ nieje nutné prekreslenie tieňov, je možné jedno vykreslenie použiť aj v nasledujúcom prekreslení, jedná sa teda o statické scény, kde sa žiadny z modelov nehýbe. V opačnom prípade je nutné v každom prekreslení túto textúru znova vygenerovať. Táto metóda má nevýhodu využitia príliž veľkého množstva grafickej pamäte a niekoľkonásobné vykresľovanie scény, pri využití viacerých zdrojov svetla alebo tieňovej kocky *(shadow cube map)*. Napríklad pri použití troch svetelných zdrojov je nutné vykresliť celú scénu štyri krát - tri krát pre tieňové textúry a jeden krát výsledný, farebný obraz. Taktiež naráža na problémy ako pri bežnom mapovaní textúry, kde je použitie diskrétnej hĺbkovej mapy spôsobená nepresnosť.

Projekčná matica z pohľadu zdroja svetla môže byť ortografická, čo by reprezentovalo svetlo, ktorého všetky lúče vychádzajú zo zdroja kolmo v istom smere. Tento efekt by malo svetlo veľmi malé a vzdialené, kde sa môže zanedbať uhol vyžarovania lúčov. Toto riešenie má menšiu chybovosť [8], no takéto svetlo sa však v prírode nevyskytuje. Zvyčajne sa

[5] Obrázky sú vyrobené z demonštračnej aplikácie.

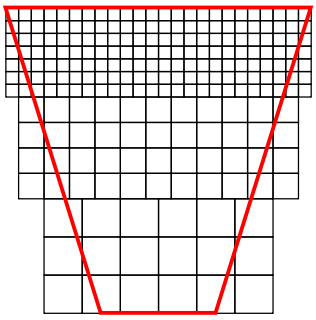


Obr. 2.6: Vizualizácia výsledku použitia Shadow Mapping. [5]

používa perspektívna projekčná matica, podobne ako pri obyčajnom priestorovom priemete s uhľom viditeľnosti, kde lúče vychádzajú zo zdroja pod rôznymi uhlami.

Perspektívna matica má však obmedzenie na uhol viditeľnosti maximálne $\alpha < \pi$. Týmto obmedzením uhla pohľadu, a teda aj uhla generovania hĺbkovej mapy, robí zo svetla reflektor *(spotlight)*. Riešením je generovať viac hĺbkových máp aby sa obsiahla celá scéna v okolí. Častým riešením je hĺbková kocka (shadow cube map) ktorá je myslená ako šesť rovnako veľkých textúr do všetkých strán. S využitím perspektívnej matice a s uhlom $\alpha = \frac{\pi}{2}$ (45 °), obsiahne celý priestor okolo seba. Pamäť hĺbkovej mapy a čas vyhotovenia sa pri takomto všestrannom priemete približne 6-krát násobí.

Kvalita a detaily rastrového tieňa závisia od rozlíšenia textúry alebo uhla záberu. Použitím perspektívnej matice, metóda sa nazýva aj *perspective shadow map*, sa zvýši rozlíšenie u bližších objektov a naopak zníži u tých, ktoré sa nachádzajú vo väčšej vzdialenosti od stredu premietania [8]. Objekty vzdialené od svetla, sú z jeho pohľadu malé a priemet do tieňovej textúry je veľmi nedetailný, čo vedie k nepresnostiam.



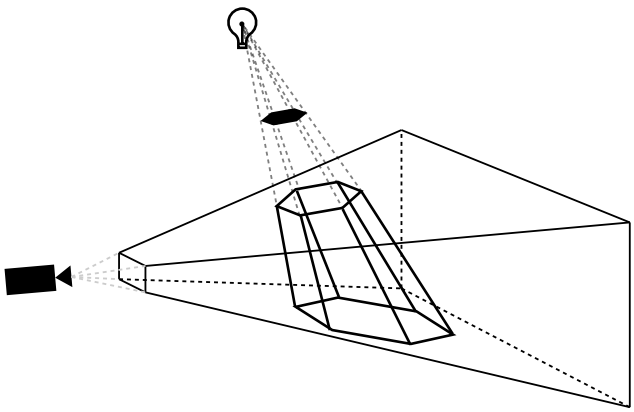
Obr. 2.7: Ilustrácia rozlíšení kaskádových hĺbkových máp ku kamerovému pohľadu (červenou).

Riešením rôznych vzdialeností sú kaskádové tieňové techniky *(cascaded shadow maps)* (viz 2.7). Algoritmus generovania tieňa využíva niekoľko hĺbkových máp, ktoré majú väčšie rozlíšenie v závislosti od vzdialenosti objektu. [4]

Shadow Mapping patrí medzi najrýchlejšie tieňovacie techniky. Keďže pracuje podobným spôsobom ako pri obyčajnom vykresľovaní scény, dokáže generovať tieň aj scény reprezentovanú nie len plochami. Na úkor rýchlosti čelí všetkými nedostatkami pri bežnom renderovaní obrazu [8].

Shadow Volumes

Ďalšou obľúbenou technikou je Shadow Volumes *(tieňové telesá)*. Taktiež patrí medzi najčastejšie používané techniky na vrhanie tieňov v reálnom čase. Základný algoritmus pracuje s polygónmi a vytvára ostré, vektorové tiene. Myšlienkou je vytvorenie tieňového telesa pre každý z objektov *(occluder)* tak, že reprezentuje objem vrhnutého tieňa do scény. Tieňové teleso je prienik vrhnutého tieňa objektom a pohľadového telesa (viz 2.8). Teleso tak vymedzuje priestor neosvetlený zdrojom svetla, z ktorého je vytvorený.



Obr. 2.8: Ilustrácia prieniku vrhnutého tieňového telesa polygónu a pyramídového viditeľného priestoru kamery.

Pokiaľ sú známe všetky tieňové telesá, je možné s nimi testovať ich povrchové polygóny. Z testu môžu nastať tri stavy a to [8]:

- Polygón leží celý vnútri tieňového telesa - je zatienený
- Polygón leží cely mimo tieňového telesa - je osvietený svetelným zdrojom
- Polygón leží čiastočne v telese

Teleso je jednoduché vytvoriť s pomocou jeho obrysov *(silhouette edges)* z pohľadu zdroja. Každá obrysová hrana definuje jednu stenu tieňového telesa. Dá sa však vytvoriť aj inými technikami, ako napríklad pre každý trojuholník zvlášť, čo je viac náročné na výpočet [8]. Pre jeden trojuholník, je táto oblasť vymedzená samým trojuholníkom, plochami definovanými jeho extrudovaných okrajov a uzavretím trojuholníkom vo vzdialenosti predĺžených okrajov. Extrúzie okrajov sú skonštruované štyrmi bodmi, a to dvoma krajovími trojuholníka a dvoma extrudovanými alebo teda projektovanými do najvzdialenejšiemu bodu v scéne [3].

Bod P leží v tieni trojuholníka, ak leží v jeho tieňovom telese. Na zistenie, či je bod P obsiahnutý v telese, je prevedený test, či je jeho obsahom. Jedným zo spôsobov je vykonať tento test vrhnutím lúča z referenčného bodu mimo tieňa do bodu P. Vždy, keď lúč vstúpi do telesa, sa počítadlo pripočíta a odpočíta keď vychádza. Ak je hodnota čítača nenulová,

tak bod P je osvetlený, v opačnom prípade leží v tieni [3]. Iným spôsobom, že plochy privrátené ku kamere posúvajú všetko čo je za nimi do tieňa a odvrátené presný opak. Aby teleso alebo jeho časť ležala v tieni, tak musí platiť, že sa nachádza za privrátenými a súčasne pred odvrátenými plochami tieňového telesa [8].

V algoritme *depth-pass (Z-pass)* sa na rozlíšenie týchto situácii používa *depth-test*. Najprv sa vyrieši viditeľnosť scény bez tieňových telies a výsledok sa uloží. Ku každému pixelu sa priradí čítač, ktorého hodnota je rovná počtu tieňových telies v ktorých sa kamera momentálne nachádza [8]. Potom stačí previesť *depth-test* tieňových telies, čím zistíme koľko privrátených a odvrátených plôch je pred a za pixelom. Zistenie na počiatočný počet telies, v ktorý sa kamera nachádza je netriviálny a preto je algoritmus *depth-pass* určený len na scény, kde je kamera umiestnená mimo scénu. Problém môže nastať s tieňovými telesami, keď pohľadové teleso kamery z prednej strany oreže niektoré časti v dôsledku šetrenia spracovávania polygónov. Môžu tak nastať diery v telesách, ktoré majú za dôsledok chybné tiene.

Riešením tohoto problému je algoritmus *depth-fail (Z-fail)*, ktorý má obrátený zmysel algoritmu *depth-pass*. Hodnota čítača, na začiatku nastaveného na nulu, sa bude meniť v prípade, že *depth-test* neprejde. Čítač teda počíta hodnoty odzadu, teda z opačného smeru ako *depth-pass*, z nekonečnej vzdialenosti k obrazovému povrchu. Tento algoritmus neprichádza do konfliktu s prednou orezávacou rovinou a je nezávislý na počiatočnom počte telies polohy kamery [8]. Tento krát je potrebné dávať pozor na orezanie zadnej časti pohľadového telesa. Tento problém je riešený premietnutím extrúdovaných častí tieňových telies do nekonečnej vzdialenosti [8]. Na premietanie hrán do nekonečna je potrebná projekčná matica, ktorá všetky body transformuje do rozmedzia bodu najbližšieho ku kamere *(the nearest point)* a abstraktného nekonečna *(the farest point)*. Výpočet koeficientov nekonečnej matice *(infinite matrix)* sa odvíja od bežnej (2.1), ibaže vzdialenosť f *(far)* je položená v limite do nekonečna [9].

$$\lim_{f\to\infty} \begin{pmatrix} e & 0 & 0 & 0 \\ 0 & \frac{e}{a} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} e & 0 & 0 & 0 \\ 0 & \frac{e}{a} & 0 & 0 \\ 0 & 0 & -1 & -2n \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{2.1}$$

Všetky vektory vynásobené touto maticou, by mali mať súradnicu $z = 1.0$. Takáto modifikácia do nekonečna ale môže mať za následok nepresnosť výpočtu a hĺbka z pri malých hodnotách, môže byť niečo nad alebo pod maximálnou hodnotou 1.0. Preto je nutné do matice zakomponovať odchýlku ϵ a výsledný výpočet vyzerá ako je na ukážke (2.2) [9].

$$P_{infinite} = \begin{pmatrix} e & 0 & 0 & 0 \\ 0 & \frac{e}{a} & 0 & 0 \\ 0 & 0 & \epsilon - 1 & (\epsilon - 2)n \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{2.2}$$

Keď sa zaistí aby nenastali diery v tieňových telesách tak sa tento algoritmus stáva univerzálnou metódou označovanou ako *robust stencil volumes* [8]. Algoritmus je možné rozšíriť aj na viacero svetelných zdrojov, ale pri veľkom počte tieňových telies je výpočet veľmi náročný na vykresľovanie v reálnom čase. Technika Shadow Volumes sa dá použiť aj na mäkké tiene tým, že vytvoríme tieňové telesá pre všetky rohy nebodového svetla, čo taktiež môže výrazne zaťažiť celý proces tieňovania.

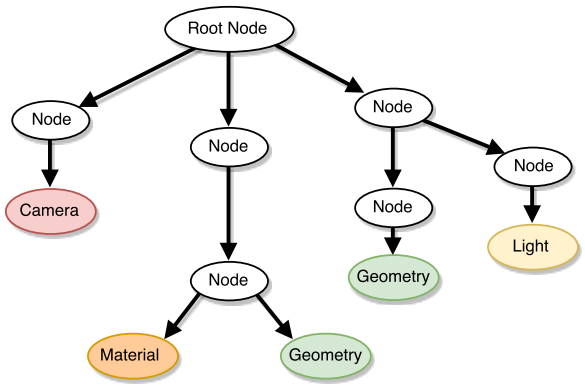
Kapitola 3

Návrh riešenia

V tejto kapitole je popísaný návrh implementácie demonštračnej aplikácie. Riešenie Shadow Mapping a Shadow Volumes techník v praxi, ako ich dosiahnuť s využitím grafickej karty a knižnice OpenGL.

3.1 Všeobecný graf scény

Keďže pracujeme so všetkými objektami spolu ako celok, je vhodné premýšľať na návrhu štruktúry scény. Takáto scéna je častým riešením vykresľovania systematicky rozdelených objektov a pod-objektov. Scéna dokáže obsiahnuť akékoľvek ľubovoľné rozpoloženie a to je veľkým prínosom pre požiadavky zvoleného tieňovania. Pozostáva z uzlov *(nodes)* ktoré reprezentujú transformáciu vo forme transformačnej matice. V mojom návrhu je scéna hierarchická a uzly môžu obsahovať atribúty ako svetlo, model a materiál.



Obr. 3.1: Vizualizácia scénového grafu použitého v aplikácii.

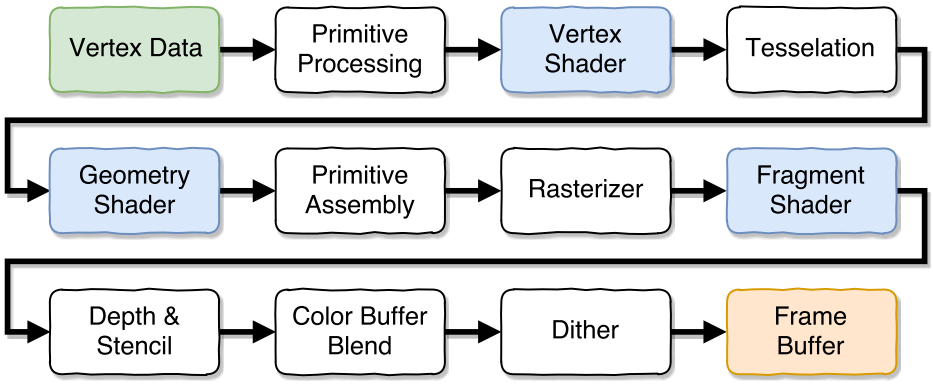
Pomocou hierarchického usporiadania je možné ľahko transformovať zreťazené uzly a ľahko získať napríklad pohyb kamery, rotáciu *(rotation)*, pozíciu *(translation)*, pomer veľkosti *(scale)*, globálnu pozíciu a rotáciu zdroja svetla, projekčnú a modelovú maticu *(model-view projection matrix)*, atď. Uľahčuje to aj rekurzívne vykresľovanie, čo pridá ešte väčšiu dynamiku v implementácii. Podobný model popisuje aj autor článku [6] a vizuálny príklad ako vyzerá graf scény je na obrázku 3.1.

Väčšinu parametrov je nutné nastavovať manuálne do GLSL programu v každom vykresľovaní, preto je príjemné si vyrobiť objekt, ktorý pracuje s GLSL programom a jednotlivé uzly automaticky spracuje.

3.2 Využitie OpenGL a GLSL

Knižnica OpenGL umožňuje vývojárovi pracovať s grafickou kartou a jej rozšíreniami. Patrí medzi najrozšírenejšie knižnice medzi grafickými a hernými aplikáciami, keďže pracuje na rôznych operačných systémoch. Sú aj ďalšie alternatívne knižnice ako napríklad DirectX (platforma Windows) alebo nízko-úrovňovo navrhnuté Metal (platforma OS X) alebo Vulkan.

Renderovací proces objektu ma niekoľko úrovní pospájaných do jednej línie *(pipeline)* (viz 3.2) a ako výsledok je obraz na monitore. Knižnica umožňuje programátorovi zasiahnuť do niektorých častí tohoto procesu pomocou malého programu *(shader program)* v jazyku GLSL *(Graphics Library Shading Language)*. V návrhu generovania tieňov je potrebné zasiahnuť vlastnými programami *vertex shader*, *geometry shader* a *fragment shader* (viz 3.2). Program *vertex shader* pracuje na úrovni pracovania vrcholov *(vertices)*. Tieto vrcholy postupne prichádzajú ako vstup a operácie sa vykonávajú pre všetky nezávisle. V tomto kroku *pipeline* sa vykonávajú všetky transformácie s projekčnými maticami. *Geometry shader* má na starosti generovanie z transformovaných vrcholov primitívne elementy ako napríklad body, úsečky ale najčastejšie sú to trojuholníky. V programe *fragment shader* sa spracovávajú jednotlivé fragmenty (pixely) výsledného obrazu. Vykonávajú sa v ňom operácie pre každý fragment samostatne a výstupom musí byť štvorprvkový farebný vektor *(RGBA)*.



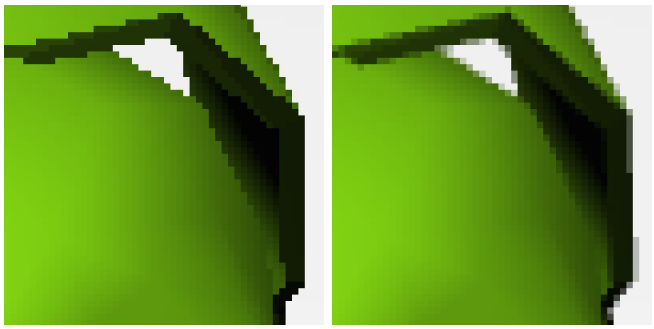
Obr. 3.2: OpenGL Pipeline. Červeno vyznačené sú fázy do, ktorých knižnica OpenGL umožnuje zasiahnuť pomocou shader programu v GLSL.

Dôležitou časťou knižnice sú textúry. V OpenGL sú textúry polia pixelov, ktoré môžu byť reprezentované v rôznych formátoch ako napríklad *R*, *RGB*, *RGBA*, *BGRA*, *depth component* alebo *depth-stencil component* [15]. Textúry môžu byť v rôznych dimenziách, no najčastejšie sa hovorí o dvojrozmerných.

Východzí cieľ vykresľovania je hlavný buffer obrazovky. Pri použití *double buffering*, sa vykresľuje do zadného *(back buffer)*, ktorý sa zamení s predným, čím sa zobrazí na obrazovke. Vykresľovať sa však dá aj do sekundárnych cieľov pomocou objektu FBO (Frambuffer Object) [14].

Na tento objekt sa pripájajú textúry na rôzne prílohy *(attachment)*, pozície. Každý FBO má niekoľko farebných príloh *(color attachment)*, v závislosti od implementácie grafickej karty a jednu hĺbkovú prílohu *(depth attachment)* alebo kombinovaný *depth-stencil attachment* [14]. V jednom vykresľovacom okne sa zakresľuje do všetkých príloh paralelne. Algoritmy generovania tieňov počítajú s medzikrokmi a FBO objekty sú nevyhnutnosťou.

V aplikácii som použil *multisampling*, ktorý rieši problém s vysokou frekvenciou vzorkovania *sampling rate* [1] (viz 3.3).

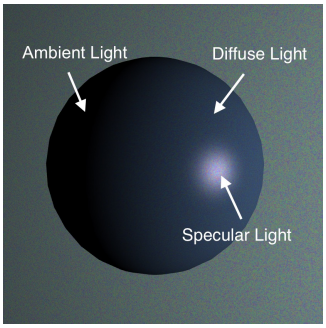


Obr. 3.3: Porovnanie častí obrazu **vľavo** bez použitia *multisampling* a **vpravo** s použitím.

Tento problém je známy aj pri textúrovaní, kde sa používajú filtre na vyhladenie obrazu. Vo vykresľovaní scény je to spôsobené tým, že hrany trojuholníka vytvárajú prerušenie signálu vzorkovania, čo vedie k ľubovolne vysokým frekvenciám [1]. S použitím *multisampling (multisample antialiasing - MSAA)*, je každý pixel na okraji polygónu vzorkovaný niekoľko krát. Spriemerovaním všetkých týchto vzoriek má výsledok plynulejšie prechody medzi hranami [13]. Knižnica OpenGL podporuje funkciu *multisampling* implicitne.

3.3 Osvetlovací model

Vizualizácia scény by mala vyzerať realisticky, preto som sa rozhodol použiť Phongov [1] svetelný model. V tomto modeli sa simulujú tri druhy svetelných efektov, ktoré vznikajú na základe fyzikálnych vlastností fotónov a materiálu. Všetky druhy sú len výsledkom odrazu svetelných lúčov od materiálu.



Obr. 3.4: Phongov svetelný model tieňovania objektov. Zobrazené tri druhy efektov svetla, odrazové *(specular)*, priame *(diffuse)* a okolité *(ambient)*.

[1] Autor modelu je Bui Tuong Phong, ktorý ho publikoval v roku 1975

Priamo odrazené svetlo do oka alebo kamery, v dôsledku lesku materiálu, sa nazýva odrazové svetlo *(specular light)*, lúče po prvom styku s povrchom rozptýlené nerovnosťou sa nazýva priame svetlo *(diffuse light)* a svetlo odrážajúce sa všetkými smermi po scéne sa nazýva okolité svetlo *(ambient light)* (viz obrázok 3.4).

Všetky tri zložky vychádzajú z jedného zdroja no napriek tomu môžu mať rôzne výsledné farby. V závislosti od druhu materiálu môže mať odrazové svetlo inú farbu ako priame. Preto sa často používa farebný faktor pre pohlcovanie svetla pre každú zložku svetla zvlášť. Výsledná farba sa vypočíta vynásobením farebných vektorov RGB alebo RGBA svetla a materiálu (3.1). Kde C je výsledná farba, L je vektor farebných zložiek svetla a M je vektor farebných zložiek materiálu.

$$I = L \cdot M = \begin{bmatrix} l_r \\ l_g \\ l_b \\ l_a \end{bmatrix} \cdot \begin{bmatrix} m_r \\ m_g \\ m_b \\ m_a \end{bmatrix} \tag{3.1}$$

Výsledné farby každej zložky sa vypočítajú ako (3.2) (3.3) (3.4), kde N je normála plochy, L je smerový vektor z bodu na ploche do zdroja svetla, R je smerový vektor odrazenia z kamery od plochy a n je vlastnosť materiálu, ktorá reprezentuje koncentráciu lesku odrazového svetla [10].

$$I_a = L_a M_a \tag{3.2}$$
$$I_d = L_d M_d (N \cdot L) \tag{3.3}$$
$$I_s = L_s M_s (R \cdot L)^n \tag{3.4}$$

Farby svetelného zdroja sú často krát toho istého odtieňa len s inou intenzitou. Okolité svetlo *(ambient)* je často krát len časť intenzity priameho svetla, keďže má rovnaký fyzikálny pôvod. Znamená to že na miesto nedopadá priame svetlo ale nieje ani úplne v tme. Pri rasterizácii obrazu sa výsledná farba jedného pixelu I_p počíta sumou všetkých druhov svetla (3.5), kde I_a, I_d a I_s sú vypočítané svetelné zložky pre daný fragment [10].

$$I_p = I_a + I_d + I_s \tag{3.5}$$

Vrhanie tieňa na iný objekt je významovo iba zatienenie difúzneho a odrazového svetla. V metóde Shadow Mapping je možné zasiahnuť do výpočtu svetla modifikáciou priamych svetiel. Úplne zatienený bod má teda iba farbu a intenzitu okolitého *ambient* svetla. V mäkkých tieňoch je teda intenzita zatienenia premenná a výpočet pixela I_p vyzerá ako v (3.6), kde s je intenzita zatienenia a platí $0 \leq s \leq 1$.

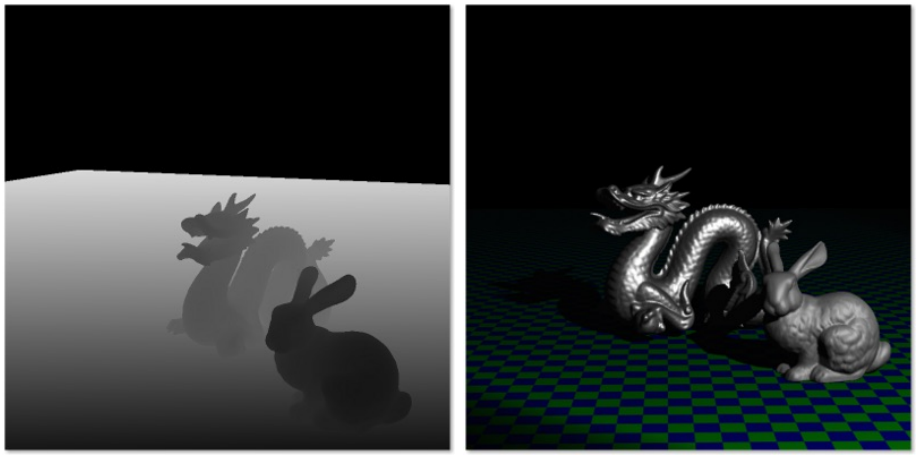
$$I_p = I_a + ((I_d + I_s)(1 - s)) \tag{3.6}$$

V prvej fáze metódy Shadow Mapping je intenzita tieňa rovná 1.0, čím zostane len okolité *(ambient)* svetlo. V tomto kroku sa dosiahne rovnaký efekt, ak sa zložky I_d a I_s budú rovnať nule.

3.4 Návrh Shadow Mapping

Algoritmus na použitie Shadow Mapping zahŕňa dva hlavné kroky. V prvom sa produkuje tieňová mapa a v druhom sa použije. V oboch fázach sa vykresľuje celá scéna, rozdiel je v tom, kam sa vykresľuje, ako sa vykresľuje a čo za výsledok sa ukladá. Veľkosť prvej fázy závisí od počtu svetiel, koľko máp sa musí vytvoriť, toľko krát sa musí celá scéna vyrenderovať. V aplikácii sa bude predpokladať iba s jedným, bodovým svetlom, čiže jedna tieňová mapa.

Princíp tieňovej mapy, alebo hĺbkovej mapy, je využitie z-buffera. V praxi je to OpenGL objekt textúry s vnútorným formátom pre uchovávanie hĺbky *(depth component)*. V prvom kroku algoritmu vykreslíme scénu z pohľadu zdroja svetla do vytvorenej hĺbkovej textúry. Keďže zachovaný výsledok je iba hĺbková mapa, všetky efekty týkajúce sa farebnej zložky, ako je tieňovanie, osvetľovanie, farbenie, sa môžu v tomto kroku vynechať aby sa zjednodušil vykresľovací proces. V shader programe sa vykonávajú len nevyhnutné priestorové transformácie a na výslednej farbe fragmentu nezáleží, pretože hĺbka geometrie sa do *z-buffer*-u ukladá nezávisle, automaticky v poslednom kroku OpenGL pipeline. Hĺbkovú textúru je nutné prekresliť vždy, keď sa scéna zmení. Výhodou sú statické scény, kde sa neudeje žiadna zmena, tým sa hĺbková mapa vykreslí iba raz a používa sa počas zvyšného renderovania. Výsledná hĺbková mapa je zobrazená na ľavej strane obrázku 3.5.



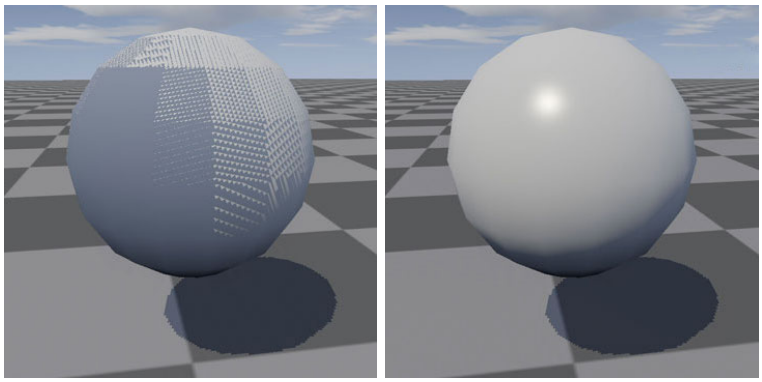
Obr. 3.5: Na lavej strane je zobrazená výsledná hĺbková mapa zo strany zdroja svetla. Na pravej strane je využitá hĺbková mapa na tieňovanie. [2]

V druhej fáze sa vykresľuje na obrazovku výsledný obraz. V tejto fáze sa môžu použiť všetky farebné korekcie, tieňovanie, aplikovanie farebných textúr na objekty, atď. S vytvorenou hĺbkovou mapou sa v tomto kroku môžu hĺbky porovnávať s hĺbkou fragmentu D_f. V každom fragmente (pixeli) sa pretransformuje pozícia do projekcie zdroja svetla. Ak P_l je projekčná matica svetla, M_l je transformácia svetelného zdroja, M_m je transformácia objektu a v_p je jeden z vrcholov vrchol objektu, tak premietnutý vrchol do súradníc svetelného obrazu v_l sa vypočíta podla vzorca (3.7).

$$v_l = P_l \cdot V_l \cdot M_m \cdot v_p \tag{3.7}$$

[2] Zdroj obrázkov: bitbucket.org

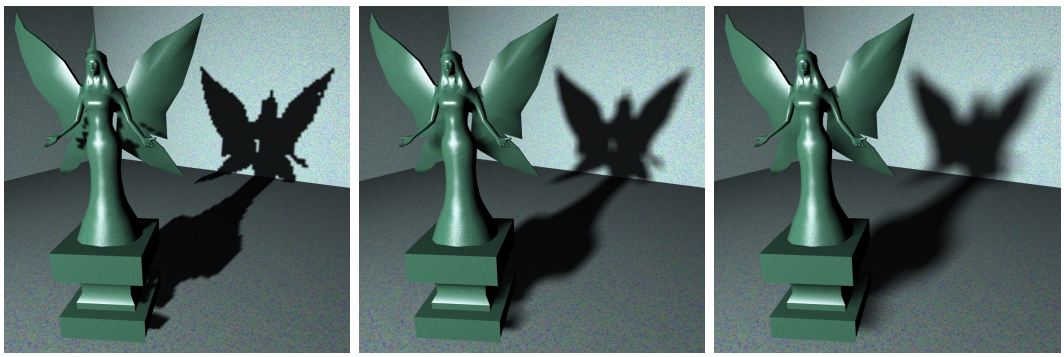
Transformáciou dostaneme súradnice hĺbkovej textúry a z nej hĺbku D_t. Pokiaľ platí $D_t < D_f$, tak z pohľadu svetla je pred pixelom bližší iný objekt a pixel je zatienený. S touto transformáciou sa ale môže stať, že viac pixelov pri sebe, spadá pod jeden pixel na hĺbkovej mape a tým vzniká hranatý tieň *(aliasing)*. Riešením kvalitnejších tieňov je použitie hĺbkovej textúry s väčším rozlíšením, no nerieši to problém s hranatosťou tieňov.



Obr. 3.6: **Vľavo** je problém predbiehania hĺbok pri porovnávaní s hĺbkovou textúrou. **Vpravo** je problém vyriešený pomocou posunutia hĺbky v smere k svetlu a v smere normály. [3]

Ďalším problémom sú tieňové akné *(shadow acne)* (viz obrázok 3.6), ktoré sú spôsobené taktiež nepresnosťou malého rozlíšenia textúry a transformáciou na lokálne súradnice. Akné vzniká predbiehaním hĺbky rovnakého bodu v scéne, z pohľadu svetla a z pohľadu kamery, kde sa stratia hodnoty pri zapisovaní do pixelov hĺbkovej textúry a vzniká chyba. Riešením je posunutie hĺbky pixelov hĺbkovej textúry o malú hodnotu *(depth bias)* a imaginárne posunutie pixelov v smere normálov objektu *(normal offset)*, čím sa predíde prelínaniu oboch hĺbok pri porovnávaní.

Keďže hĺbková mapa je tvorená bodovým svetlom, vytvára tvrdé tiene *(hard shadows)*. Tie však nie sú prirodzené ale dajú sa čiastočne simulovať technikami antialiasingu (viz 3.7).



Obr. 3.7: Rôzne úrovne antialiasingu. Prvý obrázok zľava využíva len jeden pixel, druhý plochu 3x3 pixely a tretí 7x7 pixelov. [4]

[3] Zdroj obrázkov: digitalrune.github.io
[4] Obrázky sú vyrobené z demonštračnej aplikácie.

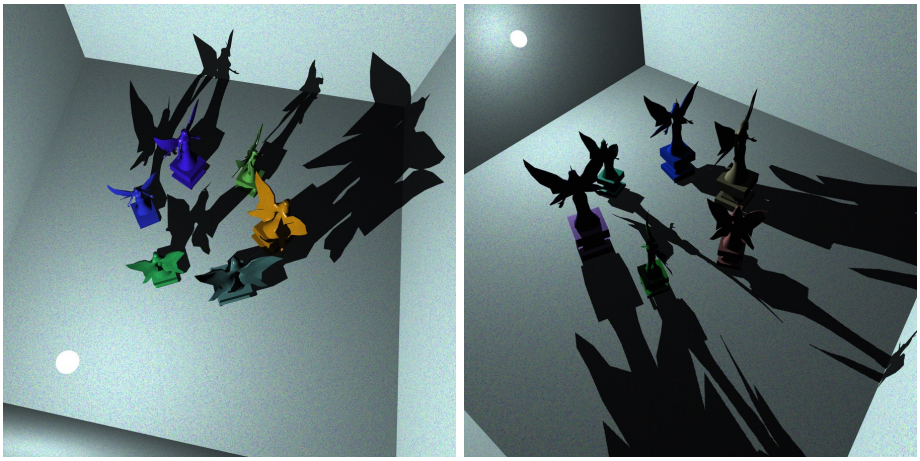
Rozostrenie pixelov na obrysoch tieňa vytvorí čiastočný polotieň *(preumbra)* a dodáva pocit plošného, realistickejšieho tieňa. Knižnica OpneGL a objekt textúry umožňuje nastaviť filtrovanie hĺbkových máp tak, aby pri porovnávaní hĺbok v teste, sa brala do úvahy interpolácia medzi pixelmi podľa toho, ako blízko sú súradnice k jednotlivým pixelom *(PCF - percentage-closer filtering)* [5].

Táto rozostrujúca korekcia je len na úrovni pixelov a čím je rozlíšenie hĺbkovej mapy väčšie, tým je preumbra menšia a menej viditeľná. Aby bolo rozostrenie na väčšej ploche, používajú sa algoritmy na anti-aliasing na zohľadnenie okolitých pixelov shadow mapy. Najjednoduchším spôsobom je priemerovanie pixelov v určitej ploche (viz obrázok 3.7). Existujú viaceré anti-aliasingové algoritmy, jedným z najznámejších je pomocou Gaussových filtrov.

Nevýhodou techniky použitia jednej textúry Shadow Mapping je, že hĺbková textúra neobsiahne celú scénu, iba jej časť. Hodnota tieňa bodov, ktorých projekcia je už mimo textúrových súradníc je nedefinovaná, najčastejšie je bod nezatienený, čím vzniká odseknutý tieň. Riešením sú trojrozmerné textúry *texture cube map*, ktoré mapujú celú scénu dookola.

Texture Cube Map

Pevné textúry môžeme popísať ako lepenie objektu namaľovaným papierom, priestorová textúra *(texture cube map)* je vyrezanie telesa z jedného bloku materiálu, ktorý má vnútornú štruktúru popísanú textúrou [8]. Textúrovacia funkcia mapuje definovaný priestor do priestoru telesa. Proces textúrovania spočíva v aplikácii súradníc x, y, z priestoru do odpovedajúcich u, v, w v textúre. Nevýhodou je veľká pamäťová náročnosť, pretože textúra *cube map* je v OpenGL reprezentovaná ako šesť dvojrozmerných textúr virtuálne posadených do tvaru kocky.

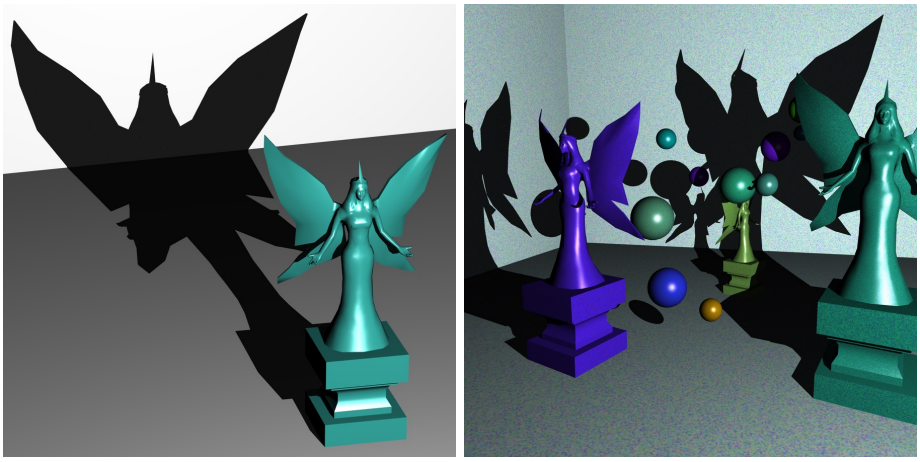


Obr. 3.8: Ukážka scény s použitím mapovania tieňa do *shadow cube map*.

Použitím tejto textúry na tieňovanie, dostaneme Shadow Mapping pre mapovanie okolitého priestoru z pohľadu svetla. Dôvodom návrhu trojrozmernej textúry je ekvivalent všestranného svetla Shadow Volumes.

3.5 Návrh Shadow Volumes

Aplikácia by mala umožniť ľubovolný pohyb po scéne, čím sa vylúči algoritmus *depth-pass*. Keďže je umožnené kamere aby sa pohybovala aj v tieni, je vhodnejšia metóda *depth-fail* (viz kapitolu 2.2). Ako počítadlo, knižnica OpenGL ponúka *stencil buffer*, ktorý slúži ako celočíselné počítadlo pre každý pixel. Na vytvorenie tieňových telies, som zvolil jednoduchšiu ale výpočtovo náročnejšiu variantu, vytvoriť teleso pre každý trojuholník objektu. Dôvodom je vymoženosť zasiahnutia do OpenGL pipeline v bode vykonštruovania primitívov s programom *geometry shader*. *Geometry shader* má na starosti poskladať z určitého počtu vrcholov do trojuholníkov alebo iných podporovaných formátov primitívov.



Obr. 3.9: Ukážka výsledku použitia Shadow Volumes *depth-fail* [5]

Tento shader sa vykonáva pre sekvenciu vrcholov, v tomto prípade pre tri vrcholy, jeden trojuholník. Z týchto bodov, okrem pôvodného trojuholníka, sa zhotovia všetky hrany telesa predĺžením vrcholov do nekonečna. Týmto spôsobom nie je nutné počítať obrysové hrany, je to univerzálny spôsob ako vyrobiť telesá pre akýkoľvek ďalší objekt. Projekcia bodov do nekonečna sa dosiahne vynásobením s nekonečnou projekčnou maticou, ktorá posadí súradnicu z každého vrcholu do nekonečna, teda do maximálnej hodnoty 1 (viz 2.2).

Algoritmus v prvej fáze vykreslí celú scénu farebne ale iba s okolitým *ambient* svetlom akoby bola celá scéna v tieni, čím sa zapíšu a uložia hĺbky pixelov. V ďalšej fáze by sa mali vykresliť geometrie tieňových telies, kedy sa zapíšu do *stencil buffer*-u počty vstupov a výstupov tieňových telies. OpenGL má podporu rozlišovania, ktoré plochy sú ku kamere privrátené a ktoré odvrátené pomocou normál plôch. V druhom kroku stačí zapnúť vykresľovanie iba privrátených plôch a potom odvrátených, no knižnica dokáže nastaviť pripočítavanie a odčítavanie hodnôt *stencil buffer*-a separátne pre obe varianty, takže všetky telesá dokáže spracovať v jednom vykreslení. V poslednom kroku je potrebné osvetliť tie pixely, ktoré majú nulovú hodnotu *stencil buffer*-a. Riešenie je jednoduché, nastavením filtra len na pixely, ktoré sú nulové. Následným vykreslením celej scény Phongovým osvetľovacím modelom, sa aplikuje difúzne a odrazové svetlo len pixely, ktoré spĺňajú nulovú požiadavku algoritmu *depth-fail*. Výsledkom sú tvrdé tiene ako vidieť na obrázku 3.9.

[5] Obrázky sú vyrobené z demonštračnej aplikácie.

3.6 Možné vylepšenia

Problém Shadow Mapping a Shadow Volumes sú, že pôvodne vytvárajú tvrdé tiene. Cieľom tieňov sú však mäkké tiene ale nie je to postačujúce. Aby sa výsledok čo najviac podobal realite, je potrebné simulovať všetky javy vzniknuté svetlom. Nevýhoda oproti ostatným technikám je globálne osvetľovanie *(global illumination)*, ktorý čiastočne so spojením Phongovho modelu chýba. Odrážanie svetla od plôch osvetľuje celú scénu a vznikajú malé tiene v rohov alebo spojoch plôch.

Taktiež algoritmus anti-aliasing v hĺbkových mapách sa môže zlepšiť na realistickejší, no na úkor kvalitnejšieho mäkkého tieňa je náročnosť, ktorá potom exponenciálne rastie pri väčších rozlíšeniach hĺbkovej mapy.

Návrh aplikácie počíta len s jedným bodovým svetlom. Implementácia viacerých zdrojov je trochu náročnejšia ale ide len o algoritmus kombinovania všetkých tieňových faktorov dokopy. Zväčšovaním počtu svetelných zdrojov sa zvyšuje aj náročnosť na pamäť a na grafický výkon [8].

Grafické karty majú však limity na, ktoré sa dá veľmi ľahko pri tieňovaní naraziť, preto je nutné hľadať alternatívy a optimalizácie.

Hierarchický návrh scény ponúka ďalšie možnosti ako optimalizovať výkon aplikácie. Jedným z účinných je algoritmus na vylúčenie objektov, ktoré nie sú momentálne viditeľné *(view frustrum culling)*, čo výrazne ušetrí výpočty počas testovania a rasterizácie.

Kapitola 4

Implementácia aplikácie

Kapitola o implementácii popisuje informácie o projekte, problémoch a riešení počas vývoja aplikácie.

4.1 Projekt a knižnice

Ako programovací jazyk aplikácie som zvolil C++ štandard z roku 2014. Výhodou tohoto jazyka je multiplaformovosť, no programovanie OpenGL vyžaduje kontext, ktorý je nutné vytvoriť pomocou platformovo závislých objektov. Tento problém rieši knižnica **GLFW**, ktorá je jednoduchá, malá a jej podstatou je vytvoriť grafické okno s OpenGL kontextom. Minimálna verzia OpenGL je 3.3, na ktorej bola aplikácia testovaná. Použitie knižnice **GLEW** je nutné len na platforme Windows aby umožnila používať rozšírené funkcie knižnice OpenGL [1].

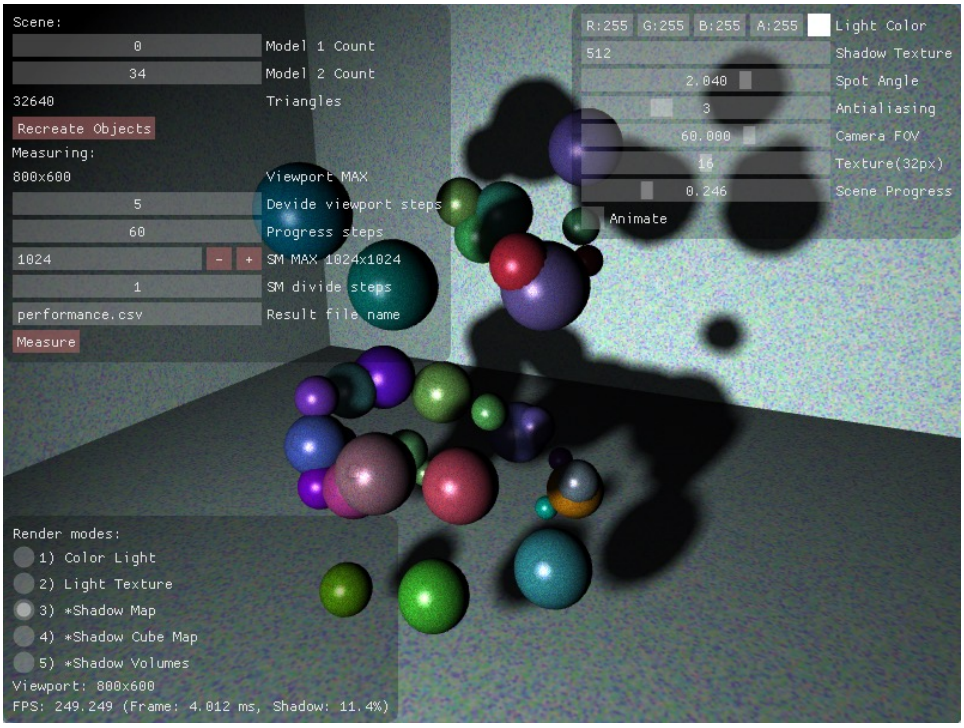
V aplikácii používam textový formát 3D objektov *.obj*. Počas vývoja som kóde tento formát načítaval vlastným algoritmom no časom mi to prišlo ako veľmi chybové riešenie. Formát je jednoduchý ale je lepšie to nechať na univerzálne riešenie - použitie knižnice **Assimp**. Táto knižnica dokáže načítať a spracovať rôzne druhy súborových formátov a vrátiť ju ako jednotnú reprezentáciu, ktorá výrazne zjednodušuje manipuláciu s dátami. Má nevýhodu, že je veľmi veľká a je nutné ju prikladať ako dynamickú knižnicu. Na kompiláciu projektu som použil multiplatformový nástroj CMake, s ktorým sa projekt ľahko prekladá na rôznych platformách. Aplikácia bola vyvíjaná na systéme macOS Sierra a testovaná aj na operačnom systéme Windows 7.

4.2 Užívateľské rozhranie

Scéna v aplikácii je dynamická, jej parametre sa dajú meniť. Na pohodlné manipulovanie je vhodné riešiť užívateľské rozhranie s ktorým je ľahko implementovaný a pohodlný. Knižnica **ImGui** umožňuje vytvoriť užívateľské rozhranie v kontexte OpenGL veľmi jednoducho. Na obrázku 4.1 je názorná ukážka aplikácie s rozhraním ImGui.

V užívateľskom rozhraní je možné meniť farbu svetla, pozíciu zdroja svetla, viditeľný uhol kamery, uhol reflektoru, veľkosti mapovaných textúr alebo parametre anti-aliasing pri hĺbkových mapách. Základom je samozrejme prepínanie medzi piatimi módami bez textúrovania, s mapovanými textúrami, technikou Shadow Mapping, Shadow Mapping s použitím *cube map* a Shadow Volumes.

[1] Na platforme Windows je potrebné extrahovať prototypy funkcii.



Obr. 4.1: Screenshot aplikácie so zapnutým užívateľským rozhraním.

Poslednou časťou je okno merania, kde si užívateľ vie nastaviť meranú scénu podľa uváženia. Užívateľ vie generovať rôzny počet objektov v scéne aby bolo možné manipulovať s počtom vykresľovaných trojuholníkov. Samotné meranie je možné ovplyvniť počtom delení rozlíšenia framebufferu a hĺbkovej mapy. Kroky rozlíšenia sa počítajú s maximálnym momentálnym rozlíšením okna. Maximálna štvorcová veľkosť hĺbkovej mapy, ktorá sa delí krokmi, je nastaviteľná užívateľom. Prioritou tohoto nastavenia merania je porovnávanie časov na celkový počet pixelov medzi jednotlivými krokmi.

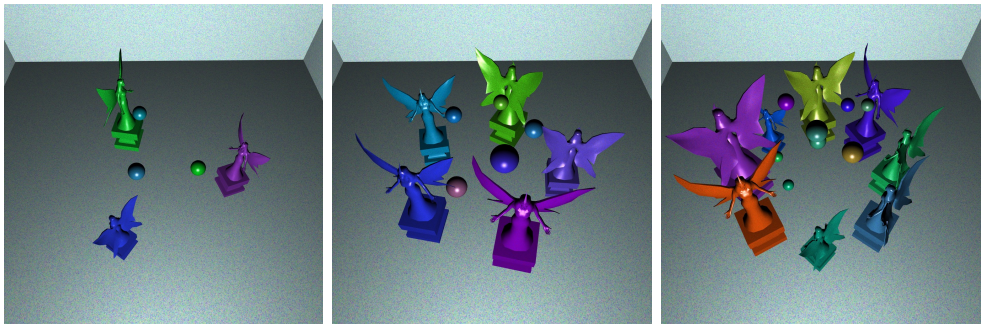
Meranie prebehne vo všetkých krokoch pozície svetla, preskúšaním všetkých rozlíšení a poprípade preskúšaním všetkých rozlíšení hĺbkovej mapy. Výsledkom je minimum, maximum a priemer časov vykreslenia pre každú variáciu rozlíšení. Separátne zaznamenáva časy čisto generovania tieňa (hĺbkovej mapy pri Shadow Mapping a generovanie tieňových telies pri Shadow Volumes) a zvyškového farebného vykresľovania obrazu. Namerané hodnoty sa postupne ukladajú do súboru, ktorého názov je tiež modifikovateľný z rozhrania vo formáte *.csv*.

4.3 Zaujímavosti a riešenia problémov

Nastavenie scény

Okno užívateľského rozhrania s nastavovaním scény na meranie umožňuje nastaviť počet dvoch typov objektov. Informácie o definícii objektov, čiže celé obsahy súborov *.obj*, sú zabudované do zdrojových súborov, takže sú súčasťou spustiteľného súboru. Predišiel som tak problémom s relatívnou cestou k externým zdrojom. Pixely textúry použitej na mapovanie modelov a samotnej obvodovej kocky sú tiež náhodne generované s maximalizovaním

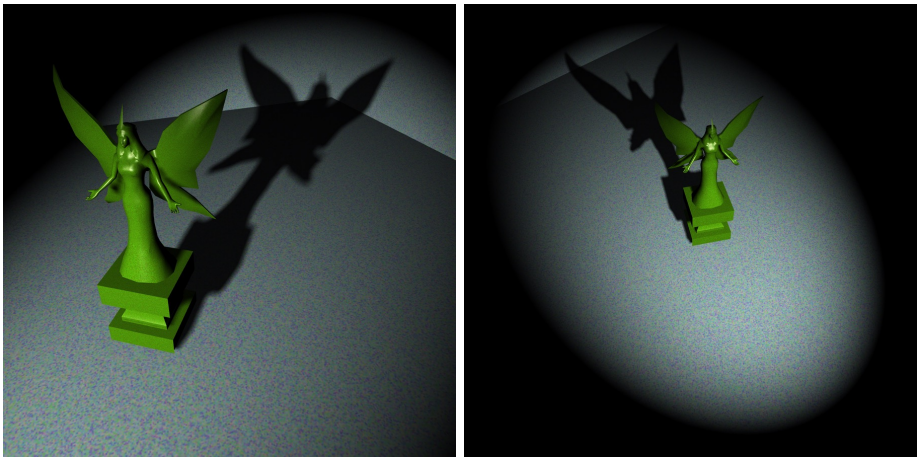
intenzity farby. Pozície, rotácie, veľkosti a farby materiálu sú generované úplne náhodne až na prvý model, ktorého pozícia sa generuje v kruhu (viz 4.2).



Obr. 4.2

Spotlight

Shadow Mapping s jednou hĺbkovou textúrou je tvorený pomocou projekcie, ktorá má obmedzený uhol. Tento pohľad zahŕňa len časť scény, takže zvyšná časť nebude tieňovaná. Hranica je veľmi ostrá, hranatá a neprirodzená. Rozhodol som sa riešiť tento problém výpočtom orezania kvádrového rozmeru textúry na elipsu. S pridaním logaritmického prechodu intenzity k okrajom, dostať výsledok podobný *spotlight* – svetlo z reflektora alebo svietiacej baterky 4.3.

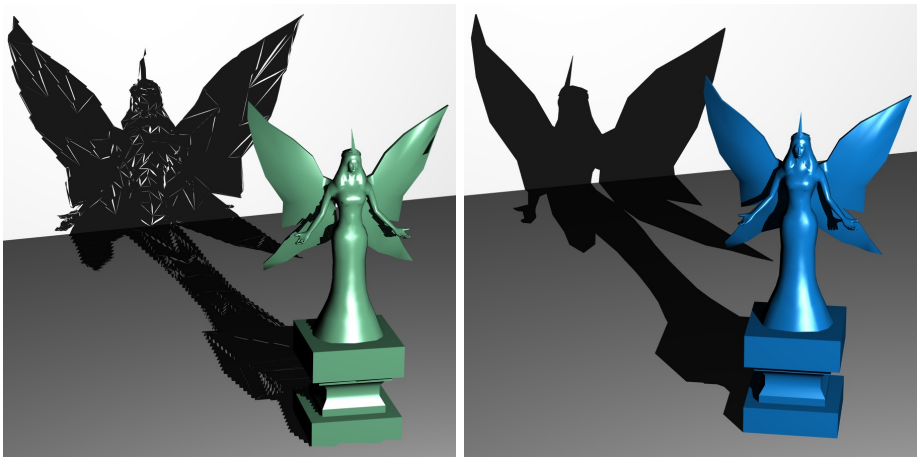


Obr. 4.3: Ukážka *spot light* vyrobeného z jednej hĺbkovej textúry.

Problém roztrieštených tieňových telies

Implementácia tieňových telies pre každý trojuholník priniesla nečakaný a nepriehľadný problém (viz 4.4 vľavo). Výsledný tieň bol pri pohybe roztrasený, plný medzier, ktoré sa zväčšovali vzďaľovaním od kamery smerom od objektu. Problémom boli zlé parametre najbližšieho bodu *near* a najvzdialenejšieho bodu *far* generovania projekčnej matice kamery.

Hodnoty projekcie na obrázku vľavo má $near = 0.001$ a $far = 1000$, čo je pomer $1 : 1000000$. Znížením na hodnoty $near = 0.1$ a $far = 100$ sa vizuálny výsledok z pohľadu



Obr. 4.4: **Vľavo** je fotka roztraseného, nepresného tieňa generovaným telesom v dôsledku veľkého pomeru 1 : 1000000 medzi *near* a *far* parametrami projekčnej matice. **Vpravo** je fotka vyriešeného problému znížením pomeru na 1 : 1000.

orezania scény veľmi nezmení, ale pomer týchto parametrov je výrazne menší 1 : 1000. Zlý pomer spôsobuje nepresnosti vo veľmi malých hodnotách pri výpočte projekcie vrcholov do nekonečna, a tým technika Shadow Volumes prináša obmedzenie aj pre projekciu.

Geometry shader

Niektoré GLSL programy v aplikácii majú *geometry shader* na generovanie trojuholníkov. Ako vstup dostáva pole dát o vrcholoch z *vertex shader*-a a často krát iba kopíruje ďalej ako výstup do *fragment shader*-a. Premenné, ktoré sa prenášajú medzi programami sa nazývajú *varrying*. Prenášané dáta, *varrying* premenné, sa dajú zoskupiť do definovanej štruktúry:

```
// Varying data type
struct Data { vec3 vertex; vec3 normal; }
// Defined input from vertex shader
in Data in_data[];
// Defined output to fragment shader
out Data out_data;
```

Dáta sú prenesené do výstupu jednoduchým priradením štruktúry na indexe vrcholu k:

```
// Copy struct to output
out_data = in_data[k];
```

Jednotlivé programy musia mať zhodné vstupy a výstupy aby bola *pipeline* neporušená. Problém nastal pri kompatibilite s platformou Windows. Implementácia kompilátora GLSL programu sa môže líšiť a v tomto prípade kompilátor vyhodnotil, že definované *varrying* premenné nie sú použité. Nezapísaním do *varrying* premenných nie je prenos konzistentný a linkovanie programov zlyhá. Riešenie je kopírovanie jednotlivých premenných separátne:

```
// Copy every element separately
out_data.vertex = in_data[k].vertex;
out_data.normal = in_data[k].normal;
```

Kapitola 5

Meranie a výsledok

Merania výsledkov boli uskutočnené pomocou demonštračnej aplikácie, ktorá ponúka rozhranie na nastavenie požadovaný počet meraní a uloženie do súboru. Pri implementácii boli použité Query objekty knižnice OpenGL, ktoré dokážu merať čas vykreslenia určitého úseku inštrukcii na grafickej karte. Všetky tieňové módy sú merané zvlášť ako tieňová fáza a normálna fáza s pomocou dvoch objektov Query. V tieňovej fáze sa v Shadow Mapping generuje hĺbková mapa a v Shadow Volumes sa pracuje so stencil buffer-om a prekrýva maskou. Z týchto Query objektov sú extrahované minimum, maximum a priemerná hodnota za niekoľko meraní v rôznych pozíciach scény, aby tieň nebol statický. Výsledky sa ukladajú do súboru s ľahko manipulovateľným formátom *.csv*.

5.1 Výsledok z merania

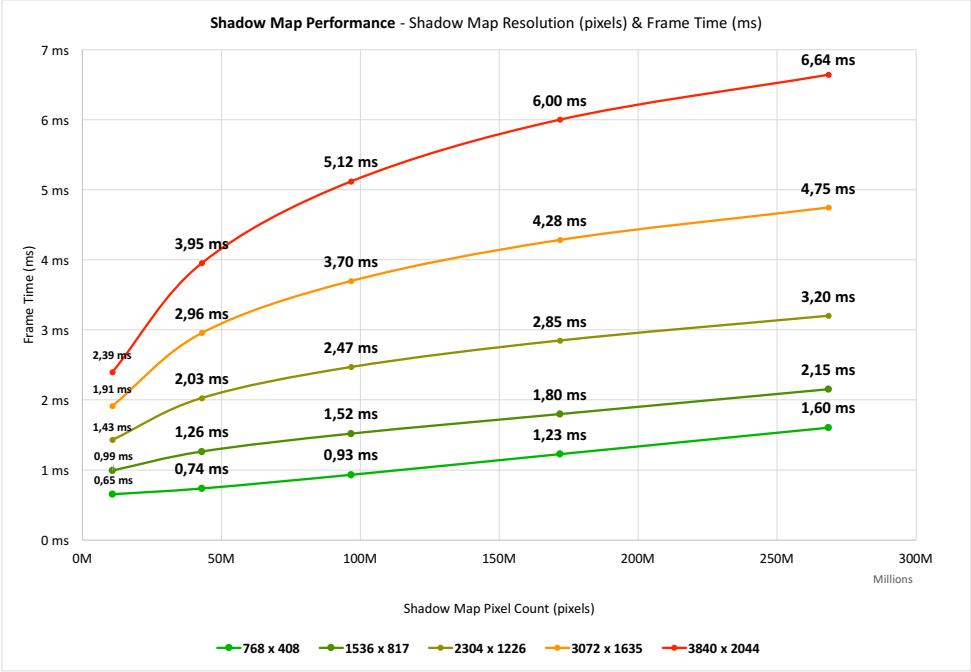
Každá technika má iné atribúty, na ktorých je stavaná kvalita tieňov. Preto je nutné zhodnotiť ako správne merať jednotlivé metódy, aby bolo meranie efektívne.

Shadow Mapping

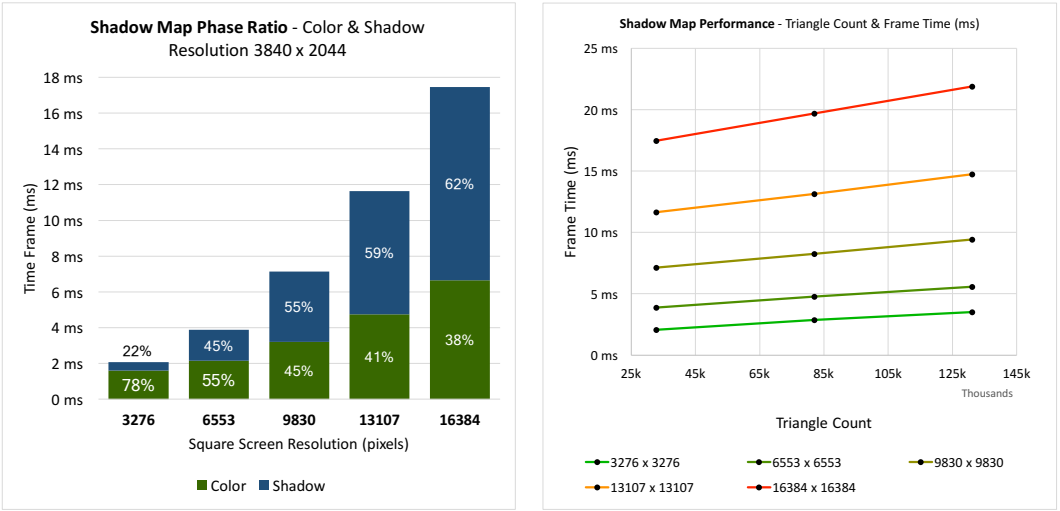
Kvalita tieňovacej techniky Shadow Mapping závisí na rozlíšení hĺbkovej mapy. Na grafe 5.1 je meraná statická scéna rastrovou technikou. Porovnáva sa počet pixelov hĺbkovej mapy k času vyhotovenia obrazu. Veľkosti hĺbkovej mapy boli v rozmedzí 3276 x 3276 do 16384 x 16384. V praxi sa takéto veľké rozlíšenia nepoužívajú, veľké hĺbkové mapy som zvolil z dôvodu vyťaženia karty. Závislosť času a rozlíšenia, ako ukazuje graf, je logaritmická a len mierne stúpa náročnosť vyhotovenia s rastúcou veľkosťou.

Ako je vidieť na grafe 5.2 vľavo, čas vyhotovenia hĺbkovej mapy obrazu je s narastajúcim rozlíšením obrazovky v stále rastúcom pomere oproti fáze, kde sa na tieňovanie mapa používa. Veľkosť textúry je však obmedzená v závislosti na grafickej karte. Na veľké pamäťové úložiská nie je dostatok pamäti alebo nieje navrhnutá tak, aby ich dokázala spracovať.

Na grafe 5.2 vpravo, je porovnávanie rýchlosti vykresľovania oproti počtu trojuholníkov. V algoritme Shadow Mapping nie sú žiadne náročné operácie okrem obyčajného spracovania každého trojuholníka. Podľa očakávania majú len lineárnu závislosť a hodnoty sa veľmi nemenia. Hlavnú úlohu teda hrá veľkosť textúry alebo v prípade viacerých zdrojov svetla počet hĺbkových textúr.



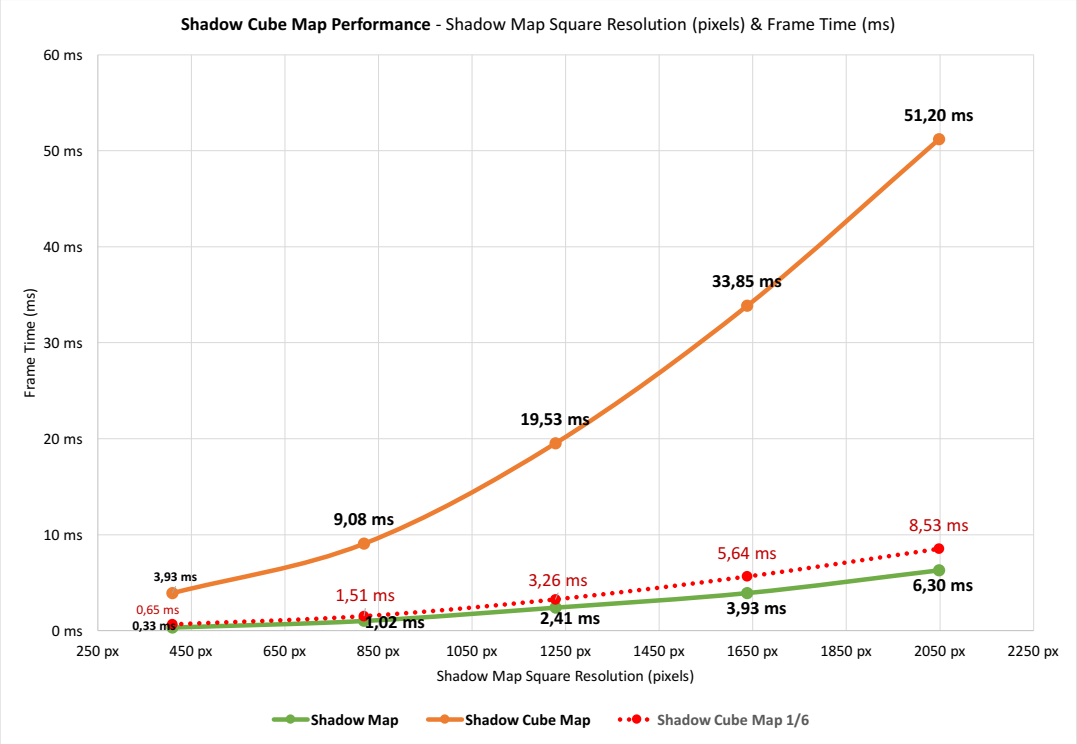
Obr. 5.1: Meranie času vykreslenia jedného obrazu metódy Shadow Mapping v rôznych rozlíšeniach s rôznym rozlíšením hĺbkovej mapy. Meranie bolo na grafickej karte Radeon RX 480.



Obr. 5.2: **Vľavo** je graf zobrazujúci čas vyhotovenia obrazu s metódou Shadow Mapping s rôznymi rozlíšeniami a konštantne veľkou hĺbkovou textúrou. Pomer medzi tieňovou fázou a normálnou je farbne vyznačený. **Vpravo** je graf porovnávajúci čas vyhotovenia obrazu s počtom spracovaných trojuholníkov modelu.

Shadow Cube Map

Shadow Mapping s použitím *cube map* je ekvivalent všestrannému svetlu techniky Shadow Volumes. Na grafe 5.3 je porovnanie času trvania tieňovej fázy vykreslenia medzi jednej hĺbkovej mapy oproti *cube map*. Červená prerušovaná čiara je podelený výsledok *cube map* šiestimi a rozdiel je podľa očakávania približne 6-krát dlhší ako pri jednej hĺbkovej mape.

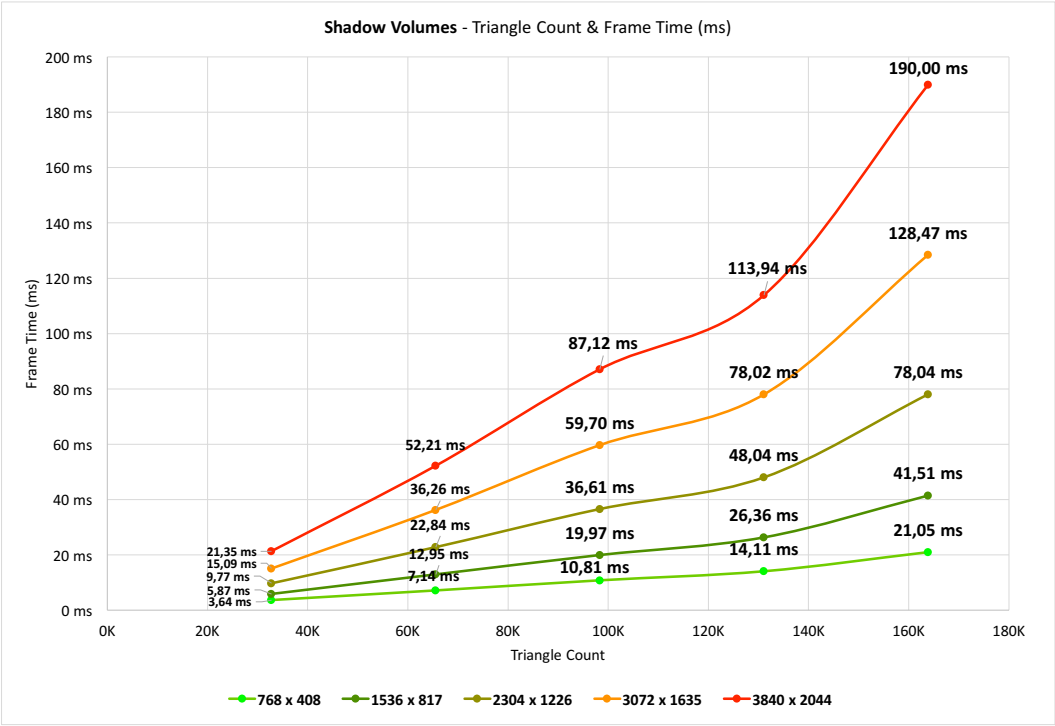


Obr. 5.3: Porovnanie časov trvania tieňovej fázy *shadow map* a *shadow cube map*. Meranie bolo vykonané na grafike Iris Pro Graphics 6200 v rozlíšení obrazovky 2880 x 1800.

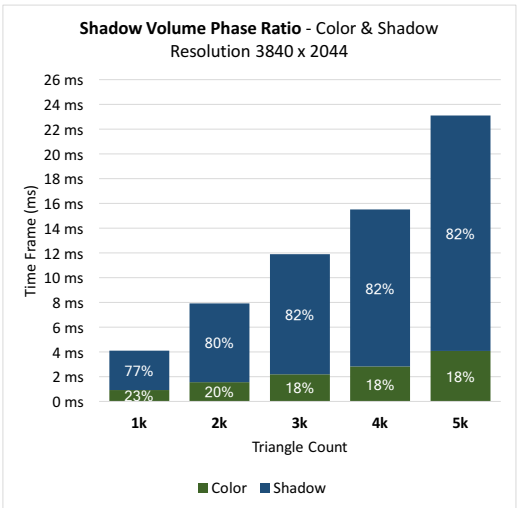
Shadow Volumes

Technika Shadow Volumes je už z teórie náročnejšia oproti Shadow Mapping. Generovanie tieňových telies pridá na počte spracovaných trojuholníkov a pri generovaní telesa pre každý trojuholník sa výkon výrazne spomalí. Graf 5.4 ukazuje závislosť počtu trojuholníkov k času vyhotovenia obrazu v piatich rôznych rozlíšeniach. Ako je vidieť, v tejto technike oproti Shadow Mapping dôležitú rolu vo výkone hrá aj počet pixelov obrazovky.

Graf 5.5 ukazuje ako sa s rastúcim počtom trojuholníkov mení pomer výkonu v rámci jedného vykreslenia medzi fázami. Prvá a tretia fáza Shadow Volumes, kde sa vykresľujú farebné zložky scény, sú spojené s porovnaním fázy, kde sa generujú tieňové telesá. Pomer sa podľa očakávania veľmi nemení, z dôvodu generovania tieňového telesa pre každý trojuholník, čo spôsobí rovnaký pomer rastu náročnosti oboch fáz.



Obr. 5.4: Meranie času vykreslenia obrazu v milisekundách s rôznym počtom trojuholníkov a rôznym rozlíšením. Meranie bolo na grafickej karte Radeon RX 480.



Obr. 5.5: Graf zobrazujúci čas vykreslenia v závislosti na počte trojuholníkov s metódou Shadow Volume. Farebne sú oddelené pomery tieňovej fázy s normálnou fázou s konštantným rozlíšením.

Kapitola 6

Záver

V práci je spomenutých niekoľko techník tieňovania, v čom majú výhodu a v čom nevýhodu. Zaujímavosťou je, že každá sa na problematiku pozerá z iného uhľa pohľadu a rieši vrhanie tieňa rôznymi spôsobmi. Do návrhu demonštračnej aplikácie som vybral dve najpoužívanejšie – Shadow Mapping a Shadow Volumes. Známe problémy týchto techník som vyriešil už v bode návrhu a spolu s knižnicou OpenGL bolo riešenie bezproblémové.

Podarilo sa vytvoriť demo aplikáciu, ktorá umožňuje vizuálne a numericky porovnávať metódy vrhania tieňa Shadow Mapping a Shadow Volumes. Aplikácia ponúka pohodlné ovládanie a užívateľské rozhranie na prispôsobenie scény a nastavenia merania. Obe implementované techniky, aj napriek vyriešeniu zásadných problémov a obmedzení, je možné zlepšiť ku kvalitnejšiemu a realistickejšiemu výsledku.

Výsledky merania ukázali, že technika vrhania tieňov Shadow Mapping je vhodná pre náročnejšiu scénu s veľkým počtom trojuholníkov. Taktiež výhodou je, že scéna nemusí byť zostavená len z trojuholníkov alebo polygónov, ale aj neobjemných čiar, či bodov. Kvalita tieňa sa dá stupňovať a prispôsobiť tak optimálnemu pomeru rýchlosti a kvality.

Technika Shadow Mapping je náročnejšia. Podľa nameraných hodnôt nie je vhodná pre komplikované scény, kde sa s rastúcim počtom polygónov výrazne zvýši náročnosť. Výhodou sú veľmi presné, ostré tiene, ktoré však nemajú možnosť regulovania kvality narozdiel od hĺbkových máp. Výsledky sú merané s generovaním tieňového telesa pre každý trojuholník zvlášť ale možno by sa hodnoty zlepšili zisťovaním obrysových hrán objektov, čím by sa zredukoval počet generovaných telies. Taktiež realistickejší výsledok by priniesla implementácia mäkkých tieňov.

Návrh počíta len s jedným zdrojom svetla, ale myslím, že zaujímavé výsledky by sa dostali s implementáciou viacerých svetiel a kombináciou všetkých ich tieňov.

Literatúra

[1] Beaudoin, P.; Poulin, P.: Compressed Multisampling for Efficient Hardware Edge Antialiasing. In *Proceedings of Graphics Interface 2004*, GI '04, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, 2004, ISBN 1-56881-227-2, s. 169–176.

[2] Burjack, K.: Ray tracing with OpenGL Compute Shaders. 2016.
URL https://github.com/LWJGL/lwjgl3-wiki/wiki/2.6.1.-Ray-tracing-with-OpenGL-Compute-Shaders-(Part-I)

[3] Eisemann, E.; Assarsson, U.; Schwarz, M.; aj.: Casting shadows in real time. In *ACM SIGGRAPH ASIA 2009 Courses*, Yokohama, Japan: ACM, 2009, s. XIII,134, doi:10.1145/1665817.1722963.

[4] Eisemann, E.; Assarsson, U.; Schwarz, M.; aj.: Efficient Real-time Shadows. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-2339-0, s. 18:1–18:54, doi:10.1145/2504435.2504453.

[5] Fernando, R.: *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004, ISBN 0321228324.

[6] Foster, G.: Understanding and Implementing Scene Graphs. 2015.
URL https://www.gamedev.net/resources/_/technical/graphics-programming-and-theory/understanding-and-implementing-scene-graphs-r2028

[7] Jensen, H. W.: *Realistic Image Synthesis Using Photon Mapping*. Natick, MA, USA: A. K. Peters, Ltd., 2001, ISBN 1-56881-147-0.

[8] Jiří Žára, J. S. P. F., Bedřich Beneš: *Moderní počítačová grafika*. Computer Press, 2004, ISBN ISBN 80-251-0454-0.

[9] Lengyel, E.: Projection Matrix Tricks.
URL http://www.terathon.com/gdc07_lengyel.pdf

[10] Lyon, R. F.: *Phong Shading Reformulation for Hardware Renderer Simplification*. Apple Computer, Inc., 1993.

[11] Purcell, T. J.; Donner, C.; Cammarano, M.; aj.: Photon Mapping on Programmable Graphics Hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, ISBN 1-58113-739-7, s. 41–50.

[12] Shirley, P.: A Ray Tracing Method for Illumination Calculation in Diffuse-specular Scenes. In *Proceedings on Graphics Interface '90*, Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 1990, s. 205–212.

[13] Wiki, O.: Multisampling — OpenGL Wiki. 2014, [Online; accessed 10-May-2017]. URL http://www.khronos.org/opengl/wiki_opengl/index.php?title=Multisampling&oldid=11428

[14] Wiki, O.: Framebuffer Object — OpenGL Wiki. 2016, [Online; accessed 10-May-2017]. URL http://www.khronos.org/opengl/wiki_opengl/index.php?title=Framebuffer_Object&oldid=13801

[15] Wiki, O.: Image Format — OpenGL Wiki. 2016, [Online; accessed 10-May-2017]. URL http://www.khronos.org/opengl/wiki_opengl/index.php?title=Image_Format&oldid=13814

Prílohy

Príloha A

Obsah přiloženého paměťového média

- **source/**
  - zdrojové kódy aplikácie a súbory *CMake*
- **latex/**
  - súbory LaTeX z ktorého bola vyhotovená dokumentácia
- **run/**
  - **win-x64/**
    - obsahuje spustiteľnú aplikáciu pre Windows x64 a všetky potrebné dynamické knižnice *.dll*
  - **mac-x86-64/**
    - obsahuje spustiteľnú aplikáciu pre macOS a všetky potrebné dynamické knižnice
- **documentation.pdf**
  - dokumentácia odbornej práce
- **README.txt**
  - inštrukcie na kompiláciu zdrojových kódov, ovládanie aplikácie, obsah média
- **preview.mp4**
  - prezentačné video aplikácie