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

NANÁŠENÍ FOTONŮ NA HIERARCHII OBRAZOVÝCH VZORKŮ

PHOTON SPLATTING USING A VIEW-SAMPLE CLUSTER HIERARCHY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARCEL KISS**
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: **Kiss Marcel, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Nanášení fotonů na hierarchii obrazových vzorků**
Photon Splatting Using a View-Sample Cluster Hierarchy
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte techniky real-time globální iluminace
2. Implementujte metodu Photon Splatting Using a View-Sample Cluster Hierarchy s využitím OpenGL. Dbejte na přenositelnost kódu na různé platformy (Windows/Linux/Mac, NVidia/AMD, ...). Využijte pro implementaci kritických částí compute shadery nebo OpenCL.
3. Navrhněte rozšíření metody.
4. Implementujte navržené rozšíření a matematicky jej dobře popište.
5. Otestujte metodu alespoň na třech různý scénách s animacemi (využijte standardní modely: Sponza, Sibenik, ...).
6. Proměřte metodu i rozšíření na různých platformách (alespoň jedno GPU od NVidia a AMD). Změřte jednotlivé části metody a jak reagují na nastavení.
7. Vytvořte demonstrační video.

Literatura:

- http://www.cse.chalmers.se/~uffe/Photon-Splatting-Using-a-View-Sample-Cluster-Hierarchy.pdf

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2.

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

Táto práca sa zaoberá technikami globálneho osvetľovania scény. V teoretickej časti rozoberá rôzne techniky globálnej iluminácie, pričom je sústredená na osvetľovanie v reálnom čase pomocou rôznych optimalizačných metód. Zameraná je na techniku nanášania fotónov s hierarchiou zhlukovaných obrazových vzoriek. Hlavnou časťou je analýza, implementácia a meranie zameranej techniky.

Abstract

This thesis deals with the techniques of global illumination of the scene. The theoretical part discusses various techniques, focusing on processing in real-time using various optimization methods. It focuses to the technology of photon splatting using view sample cluster hierarchy. The main part is analysis, implenetation and measurement of mentioned method.

Kľúčové slová

globálne osvetlenie, zhlukové tieňovanie, odložené tieňovanie, draždicové tieňovanie, nanášanie fotónov, OpenGL, OpenCL, grafika, vykresľovanie v reálnom čase, fotónové mapovanie, Mortonov kód, hierarchia zhlukov, virtuálne bodové svetlá

Keywords

global illumination, clustered shading, deferred shading, tiled shading, photon splatting, OpenGL, OpenCL, graphics, real time rendering, photon mapping, Morton code, cluster hierarchy, virtual point lights

Citácia

KISS, Marcel. *Nanášení fotonů na hierarchii obrazových vzorků*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Nanášení fotonů na hierarchii obrazových vzorků

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Tomáša Mileta. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Marcel Kiss
19. mája 2019

Poďakovanie

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

Obsah

Kapitola 1

Úvod

Táto práca sa zaoberá globálnym osvetľovaním scén s cieľom dosiahnuť fotorealistické výsledky, ktoré je žiadané hlavne vo filmovom a hernom priemysle. V počítačovej grafike je správne osvetlenie nevyhnutné k dosiahnutiu realistického dojmu a 3D efektu. Dosiahnuť takýto výsledok je vo virtuálnej scéne náročné, pretože je potrebné simulovať fyzikálne vlastnosti správania sa svetelných lúčov šíriacich sa po scéne. Mnoho programov sa k tomuto správaniu snaží len priblížiť, aproximovať na úkor lepšieho výkonu a plynulosti behu aplikácie.

Dôležitým faktorom foto-reality obrazu je ambientné svetlo, ktoré vzniká ako produkt neustáleho rozptyľovania pri interakcii s materiálom po scéne. Inými slovami sa predmety osvetľujú navzájom svojou odrazivosťou svetla, teda globálna iluminácia. V počítačovej grafike je to náročná úloha vzhľadom na dnešný výkon počítačov avšak dokážeme tento problém riešiť rôznymi spôsobmi, ktoré sú zamerané buď na kvalitu alebo rýchlosť. Je viacero algoritmov globálnej iluminácie a mnoho rozšírení ich verzií. Táto práca sa zameriava na jednu z nich s cieľom dosiahnuť globálnu iluináciu v reálnom čase.

Cieľom tejto práce je zanalyzovať a popísať metódy globálnej iluminácie vo všeobecnosti a zameriava sa na metódu *Photon Splatting Using a View-Sample Cluster Hierarchy*. Zameranú metódu implementovať, navrhnúť zlepšenie a zmerať dosiahnuté výsledky na výkonných grafických kartách.

V prvej kapitole je opísané čo znamená globálna iluminácia a vymenované sú niektoré metódy ako ju riešiť. V druhej časti prvej kapitoly je analýza a teória za spomínanou metódou vychádzajúcej z viacerých významných techník, ktoré majú všetky podobný cieľ, dosiahnuť čo najrýchlejšie spracovanie obrazu.

V druhej kapitole je detailnejší návrh analýz a teórii k cielenej metóde, podrobnejšie sa v nej riešia problémy na implementácii požadovaného výsledku. V tretej kapitole sú spomenuté významné, zaujímavé časti implementácie prezentačnej aplikácie, dosiahnuté výsledky a možné ďalšie vylepšenia a v poslednej kapitole je popísaný návrh paralelného spracovania, efektivita algoritmov, zanalyzované meranie výsledku končiac zhrnutím záverečnou kapitolou.

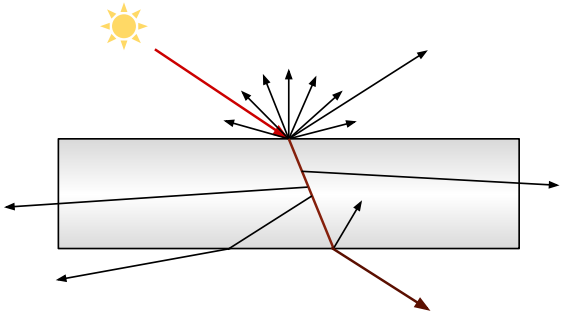
Kapitola 2

Teoretické a analytické riešenia globálneho osvetľovania

Táto kapitola opisuje čo je to globálna iluminácia a čo znamená v počítačovej grafike. Tento druh osvetlenia je všeobecne náročnou implementačnou úlohou, ktorá sa rieši rôznymi spôsobmi a technikami. V prvej časti sú spomenuté niektoré najznámejšie metódy ako dosiahnuť fotorealistický obraz pomocou globálneho osvetlenia riešením fyzikálnych javov svetla. Niektoré sa snažia dosiahnuť čo najkvalitnejší výsledok a iné zas len výsledok odhadnúť, zamerané na rýchlosť vykreslenia. Nižšie sú v tejto kapitole opísané metódy a techniky k dosiahnutiu fotorealistického obrazu pomocou jednej z techník *Photon Splatting Using View-Sample Cluster Hierarchy* zameranej na rýchlosť spracovania v reálnom čase, na ktorú je táto práca zameraná a spolu s ňou sú popísané predošlé techniky, z ktorých teória vychádza.

2.1 Svetlo a materiál

Vo všeobecnosti, keď svetlo interaguje s hmotou, vznikne komplikovaný proces medzi svetlom a materiálom. Táto interakcia je závislá od fyzikálnych charakteristík svetla a taktiež fyzikálnej kompozície a charakteristiky hmoty. Napríklad, drsný nepriehľadný povrch ako papier, odráža svetlo rozdielne ako hladký lesklý povrch ako zrkadlo. Obrázok 2.1 zobrazuje typickú interakciu svetla s materiálom [22].

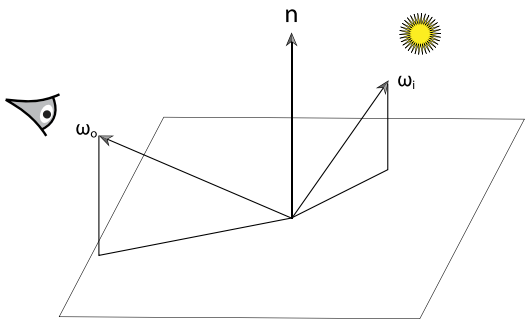


Obr. 2.1: Interakcia prichádzajúceho svetla na plochu materiálu. Časť svetla sa odrazí, iná rozplynie, ďalšia preniká skrz alebo je absorbovaná materiálom. [22]

Z obrázku sa dá urobiť zopár pozorovaní o svetle. Ako prvé, keď sa svetlo dotkne materiálu, môžu sa udiať tri rôzne interakcie: odraz, absorpcia a priepustnosť. Nejaké svetlo je odrazené, iná časť je prepustená a ďalšia je absorbovaná samým materiálom. Keďže svetlo je forma energie, zachovanie energie potom hovorí 2.1 [22].

$$dopadajce\_svetlo = odrazen\_svetlo + absorbovansvetlo + prepusten\_svetlo \tag{2.1}$$

Pre nepriehľadné materiály, je svetlo prioritne transformované na odrazené a absorbované svetlo. Ako výsledok, keď pozorovateľ uvidí osvetlený povrch, to čo je vidieť je odrazené svetlo. Tento jav je definovaný funkciou BRDF *(Bi-directional Transmission Distribution Function)*, ktorá popisuje, koľko svetla je preneseného, keď svetlo interaguje s materiálom.



Obr. 2.2: BRDF je 4D funckcia pre páry smerových vektorov, ktoré opisujú koľko prichádzajúceho svetla je rozptýleného z povrchu v danom smere [12]

V článku [12] je bližšie popísané čo je BRDF a načo sa v počítačovej grafike používa. V príklade ako na obrázku 2.2, je cieľom získať aké množstvo odrazeného žiarenia opúšťa plochu v smere ω_o k bodu pozorovateľa ω_o ako aj výsledok dopadajúceho žiarenia v danom smere. Funkcia BRDF má podobu na na ukážke 2.2

$$f_r(p, \omega_o, \omega_i) = \frac{dL_o(p, \omega_o)}{dE(p, \omega_i)} = \frac{dL_o(p, \omega_o)}{L_i(p, \omega_i) cos\theta_i d\omega_i} \tag{2.2}$$

kde, p je bod dopadu, ω_i je dopadový vektor, ω_o je odrazový vektor, θ je uhol zvierajúci oba smery, E je ožiarenie a L je žiara alebo sila v jednotkách na uhol. Z tejto funkcie potom vychádza vykresľovacia funkcia, ktorá akumuluje všetky žiarenia zo okolitej sféry na základe vlastností povrchu 2.3.

$$L_o(p, \omega_o) = \int_{S^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |cos\theta_i| \omega_i \tag{2.3}$$

Táto funkcia sa nazýva aj rovnica rozptylu, kde je integrovaný definičný obor guľa S^2 a je kľúčová v úlohe vyhodnotenia integrálu svetla v bodoch na plochách scény.

2.2 Globálna Iluminácia

Svetlo sa v scéne neustále odráža až kým nie je úplne pohltené hmotou. Ako výsledok vzniká ambientné osvetlenie, ktoré je zlúčením rôzneho rozptýleného svetla po priestore [21]. Vo fotorealistickom vykresľovaní je dôležitým faktorom a často je tento druh osvetlenia zjednodušený na istý počet odrazov a zvyšok sa počíta ako s konštantným prídavkom

intenzity ako iluminácia priestoru [21]. Simulácia odrazov k dosiahnutiu globálnej iluminácii je veľmi náročná na výkon, poctivým algoritmom by trvalo vyhodnotiť jeden obraz minúty až hodiny [21].

V najstarších modeloch sa považuje ambientné svetlo L_a za konštantné vo všetkých bodoch a smeroch, odráža intenzitu $k_a L_a$ [21]. Tento model ignoruje geometriu scény a výsledný obraz je plochý bez 3D efektu. Fyzikálne správny prístup by bol taký, kde sa zohľadnia všetky chýbajúce faktory v klasickom ambientnom modeli [21]. Tento prístup je však príliš náročný na výpočet v dynamických scénach, ktoré je potrebné vykresľovať v reálnom čase.

Algoritmy globálnej iluminácie sa pokúšajú simulovať fyzikálne založený spôsob, ako je svetlo šírené cez virtuálnu scénu. Cieľom je odhadnúť žiaru dopadajúcu na jednotlivé fragmenty obrazu, ktorá bude spojením všetkých možných trás svetelných lúčov končiacich vo fragmente. Tieto trasy lúčov začínajú vo virtuálnych zdrojoch svetla a odrážajú sa od prekážok, kde sa ich energia môže roztrieštiť alebo absorbovať [14].

Obr. 2.3: Ukážka globálnej iluminácie. Svetlo z farebných závesov sa odráža od podlahy *(Sponza)*. [1]

Existuje veľké množstvo prác o globálnej iluminácii a rôznych prístupov, mnohé matematicky riešia rovnice, ktoré produkujú fotorealistické obrázky. Tieto algoritmy typicky uprednostňujú správnosť a kvalitu pred výpočtovým časom a môže trvať minúty až hodiny, kým vyprodukujú finálny obraz [14].

Odhad intenzity svetla, ktorú daný fragment odráža späť do kamery, je témou integrálu zo všetkých smerov, odkiaľ môže svetlo prichádzať. Sčítaním všetkých vplyvných intenzít pre všetky vykresľované pixly by sa dosiahol obraz blízky realite. Tento integrál je často riešený pomocou Monte Carlo integrálu, ktorý odhaduje výsledky so zvolenou presnosťou v pomere s kvalitou obrazu.

2.2.1 Monte Carlo integrál

Metóda Monte Carlo využíva náhodnosť na výpočet integrálu s mierou konvergencie a je nezávislý na dimenzionalite [13]. V článku [13] autor opisuje, čo je to Monte Carlo integrál a načo sa používa v metódach globálnej iluminácie.

Mnoho integrálov vyskytujúce sa v počítačovej grafike sú náročné alebo ich nemožno počítať priamo. V tomto prípade ide o výpočet množstva odrazeného svetla na ploche v

[1] Zdroj obrázku: http://www.icare3d.org/research/images/givoxels_sponzanew1.png

bode dopadu svetelných lúčov, na ktoré vyčíslenie je nutné vypočítať integrál produktu dopadového žiarenia a BRDF (2.1) nad jednotkovou guľou. Vyhodnocovať integrál funkcie dopadovej žiary analyticky by vo všeobecnosti nebolo možné. Monte Carlo integrál umožňuje odhadnúť odrazovú žiaru jednoduchou voľbou množiny smerov na jednotkovej gule, vypočítať na nich žiaru, vynásobiť hodnotou BRDF pre daný smer a aplikovať váhový faktor. Arbitrárne BRDF, zdroje svetla a geometria objektov sú ľahko spracovateľné, jediné čo stačí je výpočet každej z funkcií na ľubovolnom bode.

Hlavná nevýhoda Monte Carlo je ak je n vzoriek použitých na odhad integrálu, algoritmus konverguje do správneho výsledku v $O(\frac{1}{\sqrt{n}})$. Znamená to, že ak sa má chyba znížiť o polovicu, je nutné vypočítať štvornásobne viac ako je počet vzoriek.

2.2.2 Ray Tracing

V článku [2] je opísaná technika *Ray Tracing* (stopovanie lúčov), alebo *Ray Casting* (vrhanie lúčov), ktorá 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 [2]. 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. 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) a algoritmus tak stráca na výkone.

Ak fotón vyžiarený zo zdroja svetla narazí na plochu, musí sa vyhodnotiť či bude pohltený, ale odrazený a kam bude jeho trajektória 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] (2.1). Výsledky tejto techniky sú veľmi realistické avšak kvalita záleží od počtu vyžarovaných lúčov (2.2.1), pričom veľké množstvo lúčov zaťažuje grafický výkon. Keďže tento typ techniky produkuje veľmi detailné osvetlenie scény, je vhodný na generovanie statických obrazov, kde nezáleží na rýchlosti produkcie výsledku.

2.2.3 Photon Mapping

Globálne osvetľovacie algoritmy majú dlhú históriu v počítačovej grafike od skorých výskumov založených na rádiozite a *Monte Carlo ray tracing*, po novšie algoritmy ako sú *photon mapping* [18]. *Photon mapping* je v súčasnosti najúspešnejší prístup spájajúci rádiozitu a stopovanie lúčov *(ray tracing)* [19].

Rádiozita 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 *(form 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. Prostredie je rozdelené do rovnomerne rozložených častí a

simulovaná vyžarovaná energia sa prenáša medzi nimi až kým nie sú energeticky vyrovnané [8].

2.2.4 Reflective shadow maps

Metóda *Reflective shadow maps* [4] vychádza z myšlienky tieňových máp, ktoré sa používajú na tiene v reálnom čase. Tieňová mapa je vlastne hĺbková mapa, ktorá zachytí scénu z jedného bodu buď vo forme plátna pre svetlomet alebo kocky pre bodové svetlá. Podľa tejto mapy sa určuje, či je pixel v tieni alebo naň vplýva priame svetlo. Každý pixel hĺbkovej textúry, ktorý definuje všetky body ovplyvnené priamym svetlom, sa považuje ako nepriamy zdroj svetla. Tieto sekundárne zdroje vygenerujú jeden svetelný odraz nepriamej iluminácie do scény.

Táto myšlienka je založená na takom pozorovaní, že ak je jednobodový zdroj svetla, všetky jedno-odrazové nepriame iluminácie sú spôsobené viditeľnými plochami v tieňovej mape. Takže v tomto prípade tieňová mapa obsahuje všetky informácie o priamom osvetlení a žiadna štruktúra nepriamej iluminácie nie je potrebná. Metóda je podobná *Translucent Shadow Maps*, kde sú pixly tiež považované ako bodové svetlá.

2.2.5 Voxel cone tracing

Metóda *Voxel cone tracing* z článku [3] pre interaktívne nepriame osvetlenie pracuje v troch hlavných krokoch. V prvom kroku sa vloží energia žiarenia spolu s prichádzajúcim smerom z dynamických zdrojov svetla do listov akceleračnej štruktúry *voxel octree* hierarchie. Je to uskutočnené rasterizáciou scény zo všetkých zdrojov svetla a nanášaním fotónov pre všetky viditeľné fragmenty. V druhom kroku sa prefiltruje prichádzajúca žiara do vyšších úrovní hierarchie, pričom sa dajú použiť mipmap textúry. Táto filtrovacia schéma taktiež berie do úvahy NDF *(Normal Distribution Function)* a BRDF (2.1) v závislosti na smere pohľadu. Konečným krokom je vykreslenie scény z kamery, pre každý viditeľný fragment plochy sa skombinuje priama a nepriama iluminácia. Ako aproximácia je použité sledovanie kužeľa *(cone tracing)* pre finálnu fázu zozbierania, vrhnutím zopár kužeľov na hemisfére a zozbieranie iluminácie z *octree*. Typicky pre Phongové BRDF stačí odhad difúzneho žiarenia so zopár veľkými kužeľmi, zatiaľ čo jeden úzky kužeľ pre reflexiu vzhľadom na smer pohľadu na spekulárnu zložku. Šírka spekulárneho kužeľa vychádza zo spekulárneho exponentu materiálu, s čím sa dajú efektívne vypočítať lesklé materiály.

2.2.6 Light propagation volumes

V práci [9] je spracovaná metóda *Cascaded Light Propagation Volumes for Real-Time Indirect Illumination*, ktorá tiež rieši globálne osvetlenie pre dynamické scény, kde nie sú žiadne prepočítania uskutočniteľné. Je dizajnovaná pre účely v reálnom čase a zameriava sa na vyhodnocovací čas len niekoľko milisekúnd na jeden obraz na súčasných počítačoch a konzolách. Využíva rôzne predošlé prístupy na rýchle vyhodnocovanie ako aj metódu *instant radiosity*.

Metóda rieši predošlé problémy a netrpí žiadnymi preblikávaním. Dosahuje to použitím mriežky ako úložisko svetla a geometrie v scéne. Smerová distribúcia svetla je reprezentovaná použitím sférickej harmonickej funkcie. Používa vzorkovanie *reflective shadow maps* (2.2.4) a inicializuje s týmito informáciami mriežku každý obraz odznova. Na základe tejto reprezentácie je možná paralelná schéma rozširovania svetla. Táto metóda je najlepšia pre nízko-frekvenčné nepriame iluminácie z difúznych plôch.

2.3 Predošlé práce

V tejto kapitole sú bližšie popísané techniky používané v aplikáciách v reálnom čase, ako napríklad v hrách. Jednotlivé pojmy a postupnosť techník sa opiera o bežné tieňovanie jednoduchých dynamických scén priamo počas rasterizácie a zameriava sa na cielenú techniku *Photon Splatting* [14].

2.3.1 Forward Shading

Pod pojmom *Forward Shading*, alebo popredné tieňovanie, sa myslí vyhodnocovanie svetla priamo pri spracovaní fragmentu ako súčasť rasterizácie geometrie scény. Je to bežná technika osvetľovania používaná v aplikáciách s vykresľovaním v reálnom čase. Vyhodnocuje sa v nej iba jednoduché svetlo ako napríklad Phongov alebo Lambertov model, ktoré vo veľa prípadoch postačujúce ako vo vývoji hier [16]. Problémom je redundantný výpočet intenzity osvetlenia fragmentov, ktoré sú orezané napríklad hĺbkovým testom *depth testing* a s narastajúcim počtom svetiel vo virtuálnej scéne, môže spôsobiť výrazné spomalenie vykresľovania.

2.3.2 Deferred Shading

Technika *Deferred Shading*, alebo odložené tieňovanie, má dva hlavné kroky. V prvom sa vykreslí celá scéna do pamäte po jednotlivých zložkách ako pozícia, normála a materiálové vlastnosti, spolu sa nazývajú *Geometry Buffer (G-Buffer)*. V moderných grafických API sa všetky zložky tieto zložky dajú zapísať v jednom vykreslení scény. V druhom kroku sa tieňuje každý pixel za pomoci vytvoreného G-Buffera klasickým modelom, napríklad Phongov model. [16][17]

Deferred Shading má aj svoje nevýhody a limitácie. Závažná limitácia je kombinácia s *Full Screen Anti Aliasing (FSAA)*. Problém vzniká vo veľkosti požadovanej pamäte. Napríklad pri 1080p (1920x1080) s 16x *Multi Sample Anti Aliasing (MSAA)* a 32-bitovou farbou, je potrebných skoro 256Mb na uloženie hĺbky a farby obrazu. Pre dnešné grafické karty, je pridanie ďalších atribútov do G-Buffera takmer nemožné [16]. Ďalšou slabosťou je, že neprinášajú žiadny spôsob spracovania priesvitnosti *(transparency)*, sú však práce, ktoré sa tým zaoberajú buď ako aproximácia [10], [5] alebo je náročná technika [15, Eve01] spomenuté v [16]. Všeobecný algoritmus vyhotovenia jedného obrazu sa môže popísať takto [16][17][14]:

1. Uloženie atribútov počas vykresľovania scény do G-Buffera ako pozície (hĺbka), normála a materiálové vlastnosti jednotlivých fragmentov.

2. Tienenie jednotlivých pixelov na základe vygenerovaných atribútov. Napríklad Phongovým modelom.

2.3.3 Tiled Shading

Tiled Shading, z článku [16], alebo dlaždicové tieňovanie, funguje na základe zhlukovania svetiel scény do dlaždíc obrazovky. Každá dlaždica obsahuje zoznam svetiel, ktoré potenciálne zasahujú alebo ovplyvňujú jej intenzitu svetla. Dlaždice je možné spracovať nezávisle na kalkuláciu osvetlenia. Túto techniku je možné implementovať ako Forward (2.3.1) alebo Deferred shading (2.3.2) [16].

Táto metóda je staršia a vychádza z predošlých, ktoré boli použité ako optimalizácia pre geometrie. Časom bola však nepostačujúca pre scény s miliónmi trojuholníkov, ktoré

sa používajú dnes. Technika je aplikovaná na svetlá miesto geometrie, ktorých je v scéne rádovo v tisíckach aj vo veľmi náročných aplikáciach pracujúcich v reálnom čase. *Tiled Shading* poskytuje rýchle zhlukovanie svetiel do dlaždíc a tým aj výkon do spracovania v reálnom čase [16][17].

Efektivita tejto metódy vychádza z predpokladu, že bodové svetlá majú obmedzený dosah L_r, od ktorého neprispievajú žiadne svetlo. Čím je osvetlený predmet ďalej od zdroja, tým menšia intenzita svetla dopadá na jeho povrch. Tento faktor zmenšenia intenzity g v závislosti na vzdialenosti môžme definovať nepriamou úmerou 2.4:

$$g = max\left(1 - \frac{|F_p - L_p|}{L_r}, 0\right) \tag{2.4}$$

kde F_p je pozícia fragmentu, L_p je pozícia svetla a $|F_p - L_p|$ je vzdialenosť medzi fragmentom a zdrojom svetla. Intenzita prispievaného svetla je potom vynásobená faktorom g. Je zrejmé, že od istej vzdialenosti bude hodnota g nulová, čo tvorí virtuálnu hraničnú guľu okolo zdroja svetla. Tento prístup je očividne fyzikálne nesprávne, ale reprezentuje bežné svetlo v real-time aplikáciách [16].

Rozdelenie obrazovky na dlaždice vytvára dlhé hranoly do priestoru z pohľadu kamery. Priestor z kamery je tak rozdelený na viac menších objemov, ktoré sa dajú jednoducho testovať s hranicami bodových svetiel (guľa). Jedným zo spôsobov, je rasterizácia objemu svetla, čo prináša výhodou ľubovolných hraničných objemov svetiel. Táto technika sa spomína ako netestovaná zaujímavosť v [16]. Vzhľadom na jednoduché geometrie kolíznych tvarov, postačí obyčajný analytický test priesečníku hranola a gule.

Ďalším zlepšením je *Depth Range Optimization*, kde sa vypočíta minimálna a maximálna hĺbka dlaždíc z jednotlivých zasiahnutých fragmentov. Zmenšením hĺbkových hraníc dlaždíc sa môžu vylúčiť svetlá, ktoré nemôžu zasiahnuť do príslušných pixlov obrazovky. Vygenerovanie min/max hĺbky vyžaduje čítanie z hĺbkovej mapy pred tým, ako je skonštruovaná dlaždicová mriežka. To nie je problém ak je vyhotovený *pre-z* alebo *deferred* prechod. V dlaždiciach, ktoré majú hĺbkovú diskontinuitu však môže nastať príliš veľký rozdiel medzi minimálnej a maximálnej hĺbky, čím narastá počet priradených svetiel a stráca sa výkon [16].

Algoritmus *Tiled Deferred Shading* sa dá popísať takto[16]:

1. Vykresli nepriehľadnú geometriu scény do G-Buffera. *(Deferred Shading 2.3.2)*
2. Skonštruuj dlaždicovú mriežku pokrývajúcu rozmery obrazu s fixnou veľkosťou, $t = (x, y)$, napríklad 32×32 pixlov.
3. Pre všetky svetlá: nájdi body objemu svetla (gule) v priestore obrazovky a zapíš ID svetla do každej zasiahnutej dlaždice.
4. Pre každý fragment v obraze, s pozíciou $f = (x, y)$:
   (a) prečítaj G-Buffer na pozícii f
   (b) zhromaždi hodnoty svetla zo všetkých svetiel v dlaždici $\lfloor f/t \rfloor$
   (c) zapíš hodnotu výsledného svetla do obrazu na pozícii f

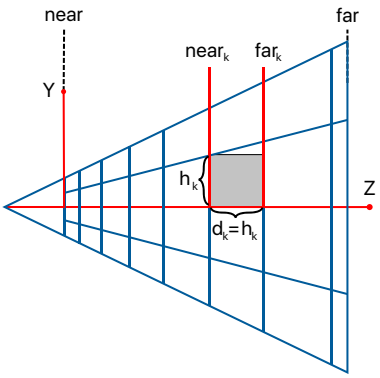
Táto metóda má svoje slabosti, ktoré dedí od predošlých techník (2.3.1) alebo deferred shading (2.3.2), kde je problémová priehľadnosť alebo príliš veľké rozmery obrazu ako pri multi-samplingu. [16].

2.3.4 Clustered Shading

Clustered Shading vychádza z techniky *Tiled Shading* (2.3.3), ktorú rozširuje o spôsob generovania dlaždíc. Nevýhodou popísaného *Tiled Shading* s *Depth Range Optimization* je, že v dlaždiciach sa môže vyskytovať hĺbková diskontinuita. Znamená to, že napríklad pri pohľade na scenériu sa často vyskytujú fragmenty oblohy, teda s maximálnou hĺbkou. Vzniknú tak redundantne dlhé hranice dlaždíc, čím sa stráca na výkone.

Existuje potenciálne nekonečné množstvo spôsobov ako zhlukovať obrazové pixly. V podstate sa vyžaduje, aby navzájom blízko pri sebe boli spojené spolu, kde je viac pravdepodobné, že budú zasiahnuté rovnakou sadou svetiel. Je veľa dynamických zhlukovacích algoritmov, ako napríklad *k-means clustering* ale žiadny nie je dobrý na spracovanie milióny vzorkov, takže sa nasadí obyčajné delenie alebo kvantizácia a poskytovanie odhadu veľkosti zhlukov [17].

Jednoduchá metóda je jednoducho použitie uniformnej mriežky v absolútnych súradniciach *(world coordinates)*. Výhodou je rýchle generovanie kľúča zhluku a všetky zhluky sú rovnakej veľkosti. Avšak mriežka je pod perspektívnou projekciou a vzdialené zhluky budú malé na obrazovke. Tým vo veľkých scénach je možné, že mnoho zhlukov bude mať veľkosť jedného pixlu, čo spôsobí slabý výkon [17].



Obr. 2.4: Exponenciálne rozdelenie pohľadového priestoru. Pre danú časť k, je znázornená najbližšia *(near$_k$)* a najvzdialenejšia *(far$_k$)* hĺbka ako aj výška *(h_k)* a hĺbka *(d_k)* bunky.

V práci [17] je popísané odvodenie generovanie mriežky založené na pozorovateľovi, kde sú zaujímavé body iba v objeme pohľadu zrezaného kužeľa *(view frustum)*. Odvodenie začína z uniformného rozdelenia rozšírením na exponenciálne (Obrázok 2.4). Majme počet delení v Y osi (S_y) daným v obrazovom priestore *(screen space)* (napríklad z veľkosti dlaždíc 32×32 pixlov), uhol pohľadu daným 2ϕ *(field of view)*, hĺbku fragmentu z_{vs} a najbližšia vzdialenosť perspektívnej projekcie *near*. Odvodením v spomenutej práci dostaneme rovnicu [17][20]:

$$k = \left\lfloor \frac{log(-z_{vs}/near)}{log(1 + \frac{2tan\phi}{S_y})} \right\rfloor \tag{2.5}$$

Použitím rovnice 2.5 sa dá poskladať kľúč zhluku *(cluster key)* daný trojicou (i, j, k) z obrazových súradníc (x_{ss}, y_{ss}) *(ss - screen space)* a obrazovej hĺbky z_vs *(vs - view space)*. Koordináty (i, j) sú obrazové súradnice dlaždice, pre veľkosť dlaždice (t_x, t_y) sú $(i, j) = (\lfloor x_{ss}/t_x \rfloor, \lfloor y_{ss}/t_y \rfloor)$.

S využitím dynamickejšej definície zhlukov sa dajú použiť aj iné atribúty na definíciu zhlukového kľúča ako len pozícia. Zo vzorkových normál sa dá generalzovať normálový kužel *(normal cone)* zakódovaním dvoch parametrov: stred a uhol rozpätia kužela. Kľúč môže byť rozšírený na pár bitov a zakódovať kvantizáciu smeru normál do priestoru kocky rozdelenej na 3x3 pod-kocky (niečo ako rubíkova kocka), kde na zakódovanie tejto informácie stačí 6 prídavných bitov. Zhluky tak môžu byť lepšie rozdelené na vzorky s podobnejšími vlastnosťami a výhodou je triviálny akceptačný test na zhluky oproti testovanému svetlu. Zhluky, ktorý normála je úplne odvrátená normálovým kuželom od svetla, môže byť rýchlo zamietnutá aj keby bola v prieniku s oblasti vplyvu svetla. Algoritmus *clustered deferred shading* sa dá popísať takto[17]:

1. Vykresli scénu do G-Buffera. (2.3.2)
2. Priraď fragmentom ich *cluster key*.
3. Nájdi unikátne zhluky.
4. Priraď svetlá k zhlukom.
5. Vytieňuj obrazové vzorky s priradenými svetlami.

Výhodou operácii algoritmu nad vzorkami je možnosť ich implementovať paralelne a keďže sú vykonávané pre každý obraz individuálne, je to pre vykresľovanie v reálnom čase vítané.

2.3.5 View Sample Cluster Hierarchy

V práci [14] je spomínaná práca [20] od ktorej bola inšpirovaná využívajúca hierarchiu, ktorá používa hierarchiu na *per-triangle* tiene testovaním geometrie [14].

Jedným z krokov algoritmu spomínanej práce, je využitie G-Buffer na výstavbu akceleračnej štruktúry, ktorá zoskupuje obrazové vzorky blízko pri sebe. Zvolí sa veľkosť dlaždíc na 8 x 8 pixlov, kde každá dlaždica môže obsahovať maximálne 64 vzoriek. Vzorka, ktorej pixel má koordináty (x, y) a ktorého hĺbka je z, sa vypočítajú koordináty zhluku **x'** ako[20]

$$\mathbf{x'} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \lfloor x/8 \rfloor \\ \lfloor y/8 \rfloor \\ \left\lfloor \frac{log(-z/near)}{log(1+\frac{2tan\theta}{S_y})} \right\rfloor \end{pmatrix} \tag{2.6}$$

kde *near* je vzdialenosť k najbližšej ploche, θ je uhol pohľadu (*field-of-view*) a S_y je počet rozdelení zhlukov vo výške. Koordinát z' bol prevzatý z práce [17] spomínanej aj v sekcii *Clustered Shading* (2.3.4), kde sa hĺbka pozorovacej pyramídy rozdelí exponenciálne tak, aby ohraničujúce boxy dlaždíc mali rovnakú hĺbku ako jeho výška a šírka v absolútnom priestore. Ďalej sa vyberie minimálny počet zhlukov, z ktorých sa bude budovať strom. Pri tvorbe hierarchie nad týmito zhlukmi, je treba zaistiť aby každý uzol mal ukazovateľ na miesto, kde sú uložené jeho deti pomocou bit-masky, ktorá hovorí ktoré deti existujú. Túto hierarchiu je možné vytvoriť veľmi rýchlo (pod 1ms pre rozlíšenie 1024 x 1024)[20], vygenerovanie a prechádzanie stromu je výrazne rýchlejšie ako obeta pamäte a vytvorenie plného stromu [20]. Hierarchia sa skladá z 5 úrovní 32 bitových slov, kde každé slovo je bitová maska pod-uzlov, ktoré existujú. Transformáciou indexu na *mortonov rozklad* sa dostane index do najnižšej, štvrtej vrstvy. Ďalšia vrstva je vygenerovaná podobným spôsobom ale

posunutím kľúča o bitov o 5 (32 bitové slovo). Týmto spôsobom sa pokračuje, až do vrchnej úrovne 0, reprezentovanej jedným 32 bitovým slovom [20].

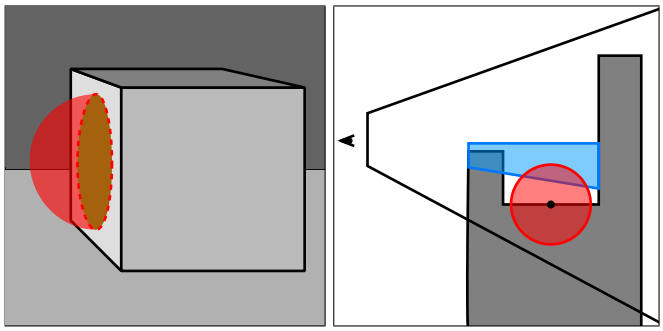
2.3.6 Photon Splatting

Photon splatting je varianta *photon mapping*. Trieda algoritmov *photon splatting*, sú fotóny stopované zo zdrojov a priamo vizualizované akumulovaním jednotlivých príspevkov do výslednej žiary každého fotónu k obrazovým vzorkám *(view samples)*, na ktoré majú vplyv. Fotóny majú teda istú umelú oblasť vplyvu, typicky guľu alebo elipsu. Hlavný rozdiel medzi *photon splatting* algoritmami je ako interpretujú geometrický útvar je testovaný prienikom s obrazovými vzorkami [14].

Photon mapping predstavená v článku [7], je dobré uplatnená technika pre globálnu iluináciu. Fotóny sú stopované zo zdroja svetla do scény a zhromažďované do akceleračnej štruktúry na vyhodnotenie odhadu žiarenia *(radiance estimate)* pri sledovaní lúčov z kamery v druhom prechode. Zvyčajne nie sú zachytávané odrazy v prvom zlome ale sekundárne odrazové lúče a ich zásahy sú zaznamenané do fotónovej mapy. Finálny krok je nazývaný *final-gather*. *Photon mapping* bol rozšírený rôznymi spôsobmi [14].

Pre globálnu iluináciu v reálnom čase môže byť finálny krok, *final-gather*, príliš drahý, ale mapovanie niekoľko fotónov cez scénu na zachytenie efektu priameho osvetlenia môže byť vyhotovené v čase jedného obrazu. Priame osvetlenie môže byť efektívne vyhodnotené so štandardným prechodom. Viditeľnosť môže byť vyhodnotená pomocou tieňovej techniky *shadow mapping* alebo pre bodové svetlá *shadow cube mapping*. Uložené fotóny z odrazov, okrem prvého odrazu, môžu byť použité na odhad nepriamej eliminácie [14].

Tradičný odhad hustoty fotónovej mapy je často príliš drahý. Alternatívny prístup je priradiť fotónom ich oblasť vplyvu, alebo rádius, a nanášať fotóny na obrazové vzorky. Tento spôsob je často nazývaný *photon splatting* alebo nanášanie fotónov, ktoré môže byť implementované na GPU vykreslením fotónov ako geometrických útvarov do hĺbkovej mapy, vypočítať ich vplyv na každej vzorke a zozbierať výsledky [14].



Obr. 2.5: Vľavo je *photon splatting* pomocou rasterizácie do hĺbkovej mapy, vzorky ležiace za fotónom (červená oblasť) sú odrezané. Vpravo je *photon splatting* pomocou dlaždíc, fotón (červená oblasť) zasahuje do oblasti dlaždice (modrá oblasť) i keď nemá na vzorky vplyv.

V práci [11] sú zviditeľnené dve metódy *photon splatting*. Prvý je spôsob rasterizácie fotónov do hĺbkovej mapy, na obrázku vpravo 2.5 je príklad, kde vzorky ležiace za fotónom (červená oblasť), budú odrezané. Druhý priraďuje fotóny k dlaždiciam na základe ich najbližšej a najvzdialenejšej hĺbky, kde redukuje nesprávne priradenie fotónov do dlaždíc

ale stále zlyháva pri nesprávnom priraďovaní v dlaždiciach s veľkým hĺbkovým rozsahom (Obr. 2.5 vľavo).

Photon Splatting Using a View-Sample Cluster Hierarchy

Rozšírením algoritmu, priraďovanie fotónov k dlaždiciam (2.3.6), o zhluky sa zlepší presnosť priradenia fotónov k obrazovým vzorkám. Rozšírenie algoritmu *photon splatting* spočíva v tom, že pre každý obraz je vygenerovaná trojrozmerná akceleračná štruktúra pre aktuálne vzorky (fragmenty). Použitá je hierarchia zhlukov obrazových vzorkov *(view-sample cluster hierarchy* 2.3.5*)*, ktorá je vytvorená zo zhlukov založených na obrazových dlaždiciach *(Tiled Shading* 2.3.3, *Clustered Shading* 2.3.4*)*. V konečnom dôsledku je každý fotón testovaný prienikom s hierarchiou obrazových vzorkov a vložený do zoznamu každého uzlu, ktorého úplne alebo čiastočne obsahuje. Doplnením metadát k uzlom hierarchie je možné dosiahnuť rýchle triviálne testy na vylúčenie prieniku fotónov s obrazovými vzorkami a vytriediť tak veľké množstvo zhlukov, ktoré ležia mimo fotónovej oblasti vplyvu pred spracovaním jednotlivých fragmentov [14].

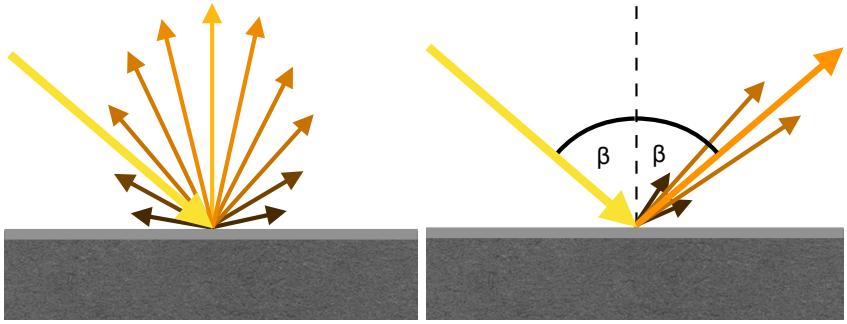
Kapitola 3

Návrh riešenia

V tejto kapitole je popísaný spôsob riešenia implementácie techník k dosiahnutiu cielenej metódy *Photon Splatting Using a View-Sample Cluster Hierarchy* (2.3.6). Popisuje analyticky bližší návrh riešení jednotlivých častí a implementačných problémov.

3.1 Primárne svetlo

V počítačovej grafike sa primárne svetlo dá vypočítať relatívne jednoducho a rýchlo. Primárne svetlo vidíme na obraze ako prvý odraz fotónov vyžiarených zo zdroja svetla od materiálu smerujúci do miesta pozorovateľa. Difúzne a spekulárne zložky (Obr. 3.1), ktoré sú súčasťou primárneho svetla, nevyžadujú náročné podmienky a počítajú sa aj v triviálnych sveteľných modeloch. Efektivita vyhodnocovania primárneho svetla je veľkou výhodou vo výkone a je dobré túto výhodu zachovať. V tejto technike sa spôsob vyhodnocovania týchto dvoch zložiek nelíši od Clustered Shading za pomoci hĺbkovej mapy a materiálových vlastností z G-Buffera.



Obr. 3.1: Vľavo rozptyl difúzneho svetla po interakcii s materiálom a vpravo odraz spekulárneho svetla do kužeľa po interakcii s materiálom.

V demonštračnej aplikácii je použitý Phongov model, u ktorého sa výsledná farba difúzneho svetla pre daný fragment počíta pre každú zložku farby RGB v rovnici (3.1)

$$I_d = M_d \cdot max(\vec{F_n} \cdot (\vec{L_p} - \vec{F_p}), 0) \tag{3.1}$$

kde I_d je RGB zložka difúzneho svetla, M_d je RGB zložka materiálových vlastností odrazu difúzneho svela, F_n je normálový vektor fragmentu, L_p je vektor pozície zdroja svetla

a F_p je vektor pozície fragmentu v priestore scény alebo v rovnakom priestore ako pozícia svetla. V rovnici sa nachádza funkcia $max()$, ktorá zaručuje aby intenzita nedosahovala záporné hodnoty, vo výslednom sčítaní intenzít by vznikali artefakty, ktoré nie sú realistické.

Pre spekulárnu RGB zložku je rovnica (3.2)

$$I_s = M_s \cdot M_{si} \cdot max(reflect(\vec{F_n}, \vec{L_p} - \vec{F_p}) \cdot \vec{E}, 0)^{M_{sp}} \tag{3.2}$$

kde I_s je intenzita RGB zložky spekulárneho svetla, M_s je materiálová vlasnosť intenzity odrazu spekulárneho svetla pre danú RGB zložku, M_{si} je materiálová vlasnosť sily odrazu spekulárneho svetla, $\vec{F_n}$ je normálový vektor fragmentu, $\vec{L_p}$ je polohový vektor zdroja svetla, F_n je pozícia fragmentu v priestore scény, $\vec{E}$ je smerový vektor z fragmentu do bodu pozorovateľa a M_{sp} je materiálová vlasnosť faktor rozptylu. Silu odrazu spekulárneho svetla je možno zakomponovať do materiálových vlastností intenzity, niektoré modely separujú túto vlasnosť od farby potom $M_{si} = 1$. Funkcia $reflect(\vec{i}, \vec{n})$ je vypočítaná ako (3.3)

$$reflect(\vec{i}, \vec{n}) = \vec{i} - 2\vec{n}(\vec{n} \cdot \vec{i}) \tag{3.3}$$

a vracia odrazený vektor $\vec{i}$ od normálny $\vec{n}$. Táto funkcia je často natívne implementovaná v príslušnej API. Spekulárnu zložku má význam počítať iba ak fragment prejde triviálnym testom, kde difúzny faktor intenzity $\vec{F_n} \cdot (\vec{L_p} - \vec{F_p}) > 0$. Znamená to, že fragment je odvrátený od svetla a nemôže odrážať žiadne priame svetlo.

Výhodou výpočtu primárneho svela s *clustered shading* je rozdelenie zápisu pozícii, normál a materiálu a následne počítanie intenzity svetla. Tým sa zredukuje redundantný, náročný výpočet pre fragmenty, ktoré sú zahodené cez hĺbkový *(depth test)* alebo iný filter. Výsledné priame svetlo je možné uložiť ako intenzitu $I = I_d + I_s$.

V realite zdroj svetla vyžaruje svetlo s nejakou intenzitou žiarenia, ktoré sa delí na dopadajúcu plochu objektov v scéne. S rastúcou vzdialenosťou intenzita svetla na meter štvorcový klesá a tým sú vzdialenejšie objekty od svetla tmavšie. Do priameho svetla prichádza nový faktor vzdialenosti, ktorého intenzita klesá kvadraticky s rastúcou vzdialenosťou ale nikdy nezanikne. Používajú sa rôzne rovnice ako aproximácia tohoto javu, najčastejšie je cieľom odrezať tento faktor pri veľmi malej intenzite alebo naopak má svetlo definovaný rádius, za ktorým je intenzita nulová.

V aplikácii je použité svetlo s rádiusom vplyvu L_r a faktor viditeľnosti fragmentu od svetla je vypočítaný ako (3.4)

$$I_f = max(1 - \frac{|F_p - L_p|^2}{L_r^2}, 0) \tag{3.4}$$

kde I intenzita svetla vzhľadom na vzdialenosť, F_p je pozícia fragmentu a L_p je pozícia zdroja svetla. Výsledná intenzita svetla je po zahrnutí vzdialenosti $I = (I_d + I_s) \cdot I_f$.

3.1.1 Tiene

Súčasťou primárneho osvetlenia je taktiež jeho nedostatok, ktorý sa javí vo forme tieňov. Tiene, tvorené objektami v rámci scény vrhané medzi sebou sú v počítačovej grafike samostatným problémom. Je viacero metód ako riešiť tento problém, najznámejšie sú *shadow mapping*, *shadow volumes* a ich deriváty. V aplikácii je použitá metóda *shadow mapping* pre bodové svetlá - *shadow cube map*. Na začiatku vyhodnocovania obrazu sa z bodového svetla scéna vykreslí v samostatnom prechode do textúrovej kocky. Podstatou vykreslenia je hĺbka nepriesvitných geometrií, ktorá je použitá na testovanie fragmentov zatienenia lúča

zo zdroja svetla. Ak je hodnota hĺbkovej mapy menšia alebo rovná skutočnej hĺbky k zdroju svetla, fragment je osvetlený a v opačnom prípade na fragment nedopadá žiadne priame svetlo, je zatienený.

Hĺbková mapa v skutočnosti zbiera hodnoty hĺbky rasterizovaného obrazu, čo sú tvrdé tiene s artefaktami. Výhodou je variabilná kvalita, ktorá sa dá zjemniť obrazovými filtrami a vzniknú tak umelé mäkké tiene. Majme faktor viditeľnosti fragmentu zo zdroja svetla I_m, ktorého hodnoty sa pohybujú medzi $< 0; 1 >$. Hodnota 0 znamená, že fragment je plne zatienený, 1 je plne viditeľný a hodnoty v rozsahu $(0; 1)$ znamenajú intenzitu zatienenia mäkkým tieňom, ktoré môžu vzniknúť napríklad lineárnou interpoláciou medzi vzorkami hĺbky. Potom intenzita výsledného svetla je počítaná ako $I = (I_d + I_s) \cdot I_f \cdot I_m$.

V tejto metóde dáva riešeniu zatienenia priameho svetla volnú ruku, vo výsledku je dôležitá intenzita priameho svetla pre fragment.

3.2 Generovanie fotónov

Každým odrazom lúča od geometrie vznikajú ďalšie lúče, ktoré závisia na uhle dopadu, orientácie plochy a odrazových vlastností materiálu plochy. Iteratíve sa vyhodnocujú odrazy lúčov, ktoré odrazmi strácajú energiu. Ideálne by tieto odrazy mali byť vyhodnocované do nekonečna, podstatou je ale dosiahnutie obrazu v konečnom čase. Zvyčajne sa generujú lúče do určitého počtu odrazov alebo sa lúče uzatvárajú ak ich energia sa po odrazoch dostane pod nejakú hodnotu ϵ. Lúče vzhľadom na závislosť na dopade, môžu byť vyhodnocované po úrovniach odrazov.

Každý lúč je definovaný ako polohový vektor jeho začiatku $\vec{o}$, smerovým vektorom $\vec{d}$ a maximálnou vzdialenosťou prieniku. Prvá úroveň lúčov má počiatok v zdroji svetla a jeho smer je zvolený náhodne. Metóda Monte Carlo vyžaduje, aby voľba smerového vektoru bola dostatočne náhodná. Na dostatočnú kvalitu odhadu žiarenia je treba mnoho lúčov, ktorých nastavenie môže byť vygenerované nezávisle čo umožňuje paralelnú implementáciu. Vzhľadom na neexistenciu implicitnej funkcie náhodných čísiel alebo vektorov, aplikácia používa vlastný pseudonáhodný generátor.

3.2.1 Vrhanie lúčov

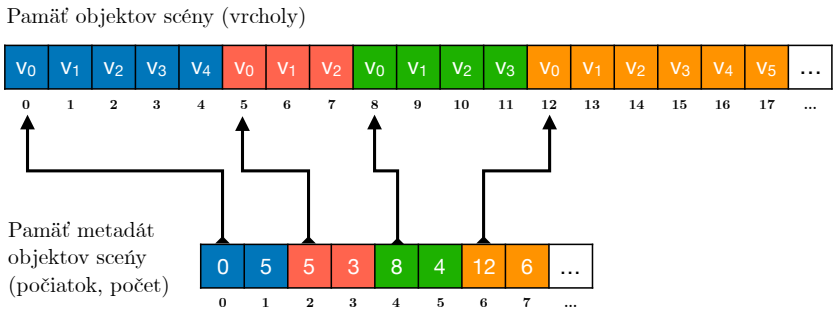
Vrhanie lúčov do scény je náročná operácia, ktorá musí byť implementovaná paralelne na grafickej karte. Optimalizácia vrhania a prieniku lúčov s geometriou v scéne, je samostatným problémom, ktoré rieši mnoho dostupných knižníc. V demonštračnej aplikácii je použitá knižnica *RadeonRays*[1], ktorá dokáže efektívne vyhodnotiť prieniky lúčov s geometriou scény ale je možné použiť ľubovoľne inú knižnicu, ktorá dokáže priniesť podobné dáta. *RadeonRays* bola zvolená pre jej implementáciu na grafickej karte, jednoduché rozhranie a ľahké použitie. Geometriu objektov v scéne je potrebné predať knižnici, ktorá z dát následne vytvorí jeden z podporovaných druhov BVH *(Bounding Volume Hierarchy)* stromu. Knižnica vracia triviálny výstup vo forme identifikátoru objektu, identifikátoru trojuholníka, polohového vektoru prieniku, vzdialenosť od počiatku lúča a barycentrické koordináty trojuholníka prieniku. Tieto atribúty sú minimalistické a nevyhnutné pre ďalšie spracovanie.

3.2.2 Vzorkovanie materiálu

Ďalším krokom po vyhodnotení prienikov lúčov, je vzorkovanie materiálových vlastností a keďže jednotlivé prieniky sú nezávislé, je možné ho vykonávať paralelne. Vstup do procesu

hľadania prienikov vrhnutých lúčov musí ísť scéna ako celok v uzavretej množine objektov. Rozhranie knižnice RadeonRays umožňuje pridávať a odoberať objekty z kontextu ale množina sa musí pred spracovaním uzavrieť. Vzhľadom na prenos dát medzi grafickou kartou a RAM, je vhodné brať geometriu všetkých objektov a scénu ako jeden celok.

Ako výstup z procesu vrhanie lúčov je k dispozícii identifikátor objektu. Ak scénu postavíme tak, aby bol identifikátor poradový index v globálnom poli objektov, môžu sa všetky geometrické dáta vložiť do jednej veľkej vyrovnávacej pamäte, čo by zefektívnilo vzorkovanie vrcholových vlastností. Pre toto globálne pole je potrebné si uchovať začiatočnú pozíciu poľa vrcholov a počet vrcholov pre daný objekt. Toto globálne pole je možné použiť na vykreslenie scény a zároveň vzorkovanie vrcholových vlastností priamym indexom s efektívnym využitím pamäte. Štruktúra vyrovnávacej pamäte je znázornená na obrázku 3.2.



Obr. 3.2: Pamäť scény využívaná na vykresľovanie pomocou OpenGL príkazov a vzorkovanie vrcholov pomocou kernelov OpenCL.

Z globálneho poľa sa pre každý úspešný prienik s povrchom vezme jeho pozícia, normála a BRDF pre difúzne svetlo. Pre difúzne svetlo je BRDF konštantná a jej vzorka je vlastne vzorka difúznej textúry objektu. Každý vrchol má atribúty pozície, normály a textúrových koordinátov, ktorých vzorka prieniku $\vec{p}$ na trojuholníku s troma vrcholovými vektormi $\vec{v_1}, \vec{v_2}, \vec{v_3}$ a s dvojrozmernými barycentrickými koordinátami t_u, t_v sa vypočíta ako v rovnici 3.5.

$$\vec{p} = \vec{v_1}(1 - \vec{t_u} - \vec{t_v}) + \vec{v_2}\vec{t_u} + \vec{v_3}\vec{t_v}; \tag{3.5}$$

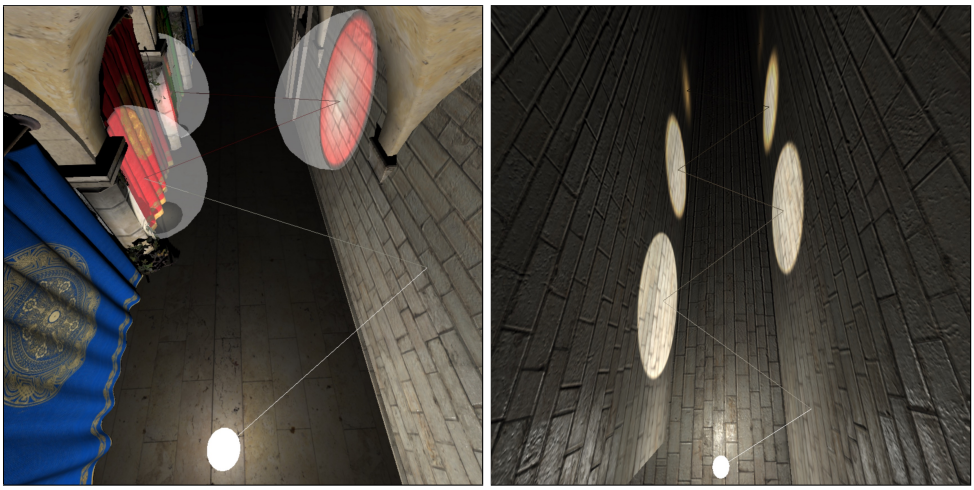
Vzorka odrazeného žiarenia materiálu je potom vzorka difúznej textúry objektu na normalizovaných textúrových kooridnátoch $\vec{p_{uv}}$.

Vzorkovanie vrcholov je vyhotovené efektívne v jednom kerneli ale textúry môžu narážať na limitácie. Zvyčajne je jedna difúzna textúra pre jeden objekt ale jedna textúra môže byť spoločná pre viac objektov. Každá textúra má iba jednu inštanciu a je potrebné im priradiť identifikátor, podla ktorého budú objekty referencovať svoju textúru do globálneho poľa textúr v scéne. Každý lúč môže trafiť do ľubovolného objektu a môže vzorkovať z inej difúznej textúry. Ideálne by sa do kernelu priložilo pole textúr, kde každý prienik navzorkuje z textúry na ktorú referencuje. Pole textúr je však v OpenCL problematické, má limitácie na počet predaných textúr do jedného kernelu a preto je nutné vzorkovať textúry postupne po dávkach. V modernejších rozhraniach OpenGL je možné použiť takzvané *bindless textures*, kde sa k textúram pristupuje priamo a je možné ich predať grafickej karte v poli.

[1]https://gpuopen.com/gaming-product/radeon-rays/

3.2.3 Fotóny

Z každého odrazu, okrem prvého, vzniká virtuálne bodové svetlo, ktoré osvetľuje zhluky v jeho okolí. Každý fotón má svoj smer žiarenia, z ktorého bol vyžiarený, pozíciu kde sa nachádza a oblasť vplyvu. Oblasť, ktorú ovplyvňuje, môže byť ľubovolného tvaru, podmienkou je výpočet prieniku s ohraničujúcimi boxami zhlukov. V tomto prípade ide o difúzne materiály a oblasť vplyvu je zjednodušená na guľu, ktorá je ľahko opísateľná jedným skalárom, rádiusom. Každý fotón nesie informáciu o intenzitách rôznych vlnových dĺžok, čiže farbách, ktoré môže byť odrazmi pohltené. Vygenerované fotóny z jedného lúča majú rovnomerne rozdelenú energiu, ktorú lúč nesie ako je popísané v práci [14].



Obr. 3.3: Ukážka odrazu lúčov a vygenerovania fotónov z odrazov.
Intenzity fotónov sú neprirodzene veľké pre názornú ukážku.

3.3 Clustered shading

Metóda vychádza z myšlienky zhlukov (*Clustered Deferred Shading* 2.3.4), ktorá je popísaná v práci [17]. Podstatou je zefektívniť výpočet intenzity dopadajúceho svetla na fragmenty tým, že sa rozdelia do zhlukov v ktorých majú fragmenty podobné vlastnosti ako pozícia alebo normála. Práca 2.3.6 sa taktiež odkazuje na spomínaný *Clustered Deferred Shading* rozšírený o hierarchiu vzoriek referencovanú na prácu [20]. Najskôr spomeniem prístup použitý touto prácou a neskôr opíšem trochu odlišný prístup použitý v aplikácii.

3.3.1 Generovanie kľúča

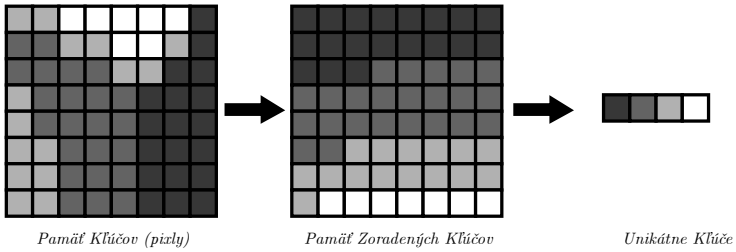
V prvom kroku sa uložia atribúty fragmentov do G-Buffera (2.3.2) z ktorých sa vypočíta kľúč zhluku, do ktorého patrí. Výpočet kľúča závisí na zvolenej veľkosti dlaždice, zvolený počet bitov na jednotlivé súradnice a zahrnutie ďalších atribútov. Veľkosť kľúča sa vo všetkých spomínaných zdrojoch zmestí do 32bitov, čo je efektívnou veľkosťou pre registre na GPU. Kľúč primárne pozostáva z polohového vektoru fragmentu v scéne, ktorého jednotlivé komponenty nutné istým spôsobom kvantizovať, tak aby ich hodnoty zabrali čo najmenej bitov. V sekcii 2.3.4 a 2.3.5 je znázornený spôsob spracovania dvojrozmernej pozície podľa veľkosti dlaždíc a dobre spracovaná hĺbka. V práci [14] a [17] používajú veľkosť dlaždíc 32×32. V práci [20], kde sú použité zhluky na tiene je použitá veľkosť 8×8.

Každým unikátnym vygenerovaným kľúčom vzniká nový zhluk a delia sa tak fragmenty s odlišnými vlastnosťami. Generovanie kľúča nie je však obmedzené na polohový vektor ale je možné doň zahrnúť rôzne atribúty. V sekcii 2.3.4 je spomenuté rozšírenie kľúča o ďalšie bity, kde je doplnený o kvantizovanú normálu fragmentu do 6-tich bitov. Pridaním nových atribútov pribúda nová dimenzia rozdelenia, čo prispieva k menším zhlukom.

3.3.2 Unikátne zhluky

V každej dlaždici má fragment priradený svoj kľúč zhluku. Cieľom je dostať celkový počet zhlukov v obraze, týmto zhlukom sa v ďalšom kroku budú generovať príslušné metadáta. Jednou z možností ako dostať celkový počet vygenerovaných zhlukov, spomínanú v práci [17] je ich zoradenie hlukových kľúčov a sčítanie sekvenčných zmien.

Výhodou je možná implementácia na GPU pomocou *compute shaders* nazývaných v OpenGL alebo *kernels* v OpenCL. Podstata je v paralelnom spracovaní obrazových vzorkov, ktorá je veľkým prínosom do výkonu. Zoradiť všetky kľúče obrazu by bolo príliš náročné aj na paralelné spracovanie, výhodou tohoto zoradenia je, že sú vopred roztriedené do dlaždíc. Nemôže existovať jeden kľúč v dvoch rôznych dlaždiciach a je teda možné tieto kľúče zoradiť lokálne v dlaždici (Obr. 3.4).



Obr. 3.4: Zoradenie a zlúčenie kľúčov v pamäti. Kľúče uložené v pamäti sú zoradené a zlúčené aby sa získali unikátne kľúče v dlaždici. Zoradenie je vykonané napríklad podľa pozície a normály.

Môže sa implementovať tak, každé vlákno zoradí jednu dlaždicu, no pri paralelnej implementácie na GPU je vhodné využitie lokálnej pamäte pre optimálne čítanie z pamäte. S veľký počtom porovnávaní elementov narastá náročnosť výpočtu ak sa čítajú kľúče z globálnej pamäte. Paralelná implementácia s lokálnou pamäťou má svoje limity, napríklad algoritmus *selection sort* vyžaduje, aby bol podporovaný počet vlákien v jeden pracovnej skupine dostatočný na každý fragment v dlaždici. Táto podmienka limituje veľkosť dlaždice v závislosti na hardvérovej implementácii grafiky. Znamená to, že napríklad pre dlaždice 32×32 aspoň 1024 vlákien v jeden pracovnej skupine, kde každé vlákno hľadá deterministický index v rámci dlaždice pre daný element. Je nutné, aby jeho pozícia bola deterministická aj v prípade dvoch identických kľúčov, pretože by vznikli kolízie pri paralelnom zapisovaní do pamäte. Výpočet deterministického indexu p v lokálnej pamäti sa môže vypočítať sekvenčne s počiatočnou hodnotou $p = 0$ pre každý kľúč na pôvodnej pozícii i porovnaním s kľúčom na pozícii j. Ak je kľúč k_i menší ako k_j deterministickou podmienkou 3.6

$$(k_j < k_i) \wedge (k_j = k_i \wedge j < i) \tag{3.6}$$

pričíta sa pozícia $p = p + 1$. Následne sa paralelne zapíše kľúč k_i do pamäte na pozíciu p. Treba však brať do úvahy, že dlaždice nemusia pokrývať celý obraz, okrajové políčka by zasahovali mimo pamäte.

Na zoradených kľúčoch v dlaždici sa sekvenčne prevedie operácia XOR ako detekcia zmeny kľúča a ak bude výsledok nenulový, nastala zmena. Počet zmien v dlaždici určuje počet unikátnych kľúčov v dlaždici a uloží sa do pamäte počtov unikátov. Paralelným sčítaním všetkých počtov v dlaždiciach sa dosiahne celkový počet unikátnych zhlukov, napríklad paralelným sčítaním a s počtom všetkých zhlukov sa môže alokovať pamäť pre potrebné metadáta.

3.4 Alternatívny prístup k unikátnym zhlukom

V práci [14] bola na optimalizáciu použitá hierarchia zhlukov, ktorá je popísaná v práci [20]. Alternatívnym prístupom k hľadaniu unikátneho počtu zhlukov, ktorý je spojený s hierarchiou, je spomenutý v práci [20], kde autor hovorí o rýchlejšom prechode stromu za cenu alokácie pamäte.

3.4.1 Predošlý prístup

Metóde spomenutej v sekcii 3.3.2 pre unikátny počet zhlukov, je použité lokálne zoraďovanie kľúčov, súčet zmien a prefixový súčet *(*prefix scan) nad všetkými počtami dlaždíc. Okrem náročného zoraďovania je nutné zachovať pôvodnú pozíciu vzorky v rámci dlaždice rozšírením metadát kľúča o pár bitov. Počas generovaní ohraničujúceho boxu sa zapíše identifikátor zhluku pre každú vzorku na zoradenej pozícii ale po zoradení sa vzorky poprehadzujú a nemajú tak spätnú referenciu na pôvodnú pozíciu fragmentu. V práci [17] je tento problém popísaný a ako riešenie používa pár bitov ako referenciu na pôvodnú pozíciu fragmentu rozšírením metadát kľúča zhluku. Spätná pozícia je relatívna k dlaždici, to znamená, že ak bude dlaždica o veľkosti 32×32 pixlov, pôvodnú pozíciu je možné zapísať do 10-tich bitov (5 pre os X, 5 pre os Y, $2^5 = 32$) [17].

3.4.2 Idea a limity

V práci [14] je použitá podobná metóda s rozšíreným kľúčom o pár bitov pre kvantizovanú normálu (3.3.1). Cieľom je vyplniť prázdne bity v kľúči pre efektívnejšie rozdelenie do menších zhlukov. V práci s inšpiráciou hierarchie [20] je použitý iný prístup k štruktúre kľúča. Miesto doplňovania kľúča o ďalšie bity, je ďalšou možnosťou použiť alternatívnu metódu, kde sa kľúč snaží zabrať čo najmenší priestor a rozšíriť ju pre účely nanášania fotónov.

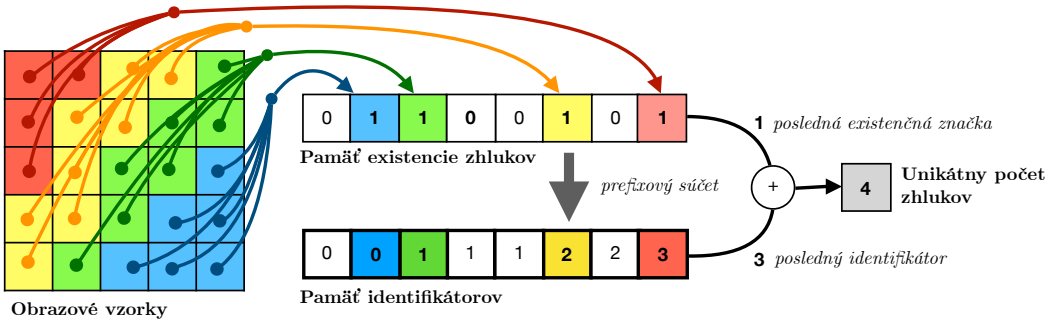
Idea vychádza z myšlienky, kde sa autor rozhodol alokovať do pamäte celý strom hierarchie. V hierarchii je najkritickejšia posledná vrstva, ktorej veľkosť závisí na celkovom rozsahu kľúča a cieľom je teda zredukovať priestory súradníc trojrozmerného kľúča na minimum. Podstatou je, že každá vzorka zapíše do globálneho poľa uzlov svoje atribúty na priamy index svojím kľúčom. To znamená, že veľkosť globálneho poľa musí byť taká veľká ako je počet možností v kľúči.

Zatiaľ čo v práci [14] je použitých 8 bitov pre súradnice X a Y, 10 bitov pre hĺbku a 6 bitov pre normálu (spolu 32 bitov), v práci [20] je použitých 7 bitov pre X a Y a 9 bitov pre hĺbku (spolu 23 bitov). To by znamenalo, že v prvom prípade by bolo nutné alokovať poslednú vrstvu hierarchie s počtom 2^{32} a v druhom 2^{23} uzlov. Majme veľkosť uzla

s ohraničujúcimi dátami a maskou existujúcich pod-uzlov 28 bajtov, 6×4 bajty *(*float) pre box a 4 bajty *(*uint) pre masku. V prvom prípade by musela byť alokovaná pamäť uzlov s veľkosťou približne 120 GiB a v druhom prípade 234 MiB. Je zrejmé, že výrazná časť pamäte nie je využitá, pretože zhluky sú prítomné iba zriedkavo a taktiež, že je nevyhnutný kompaktný zoznam poslednej vrstvy hierarchie a unikátnym zhlukom sa tak vyhnúť nedá.

3.4.3 Princíp

Vlastnosť minimálneho kľúča sa dá využiť tak, že každá vzorka po vygenerovaní kľúča zapíše existenciu príslušného zhluku priamym indexom do globálneho poľa. Do globálneho poľa stačí zapísať jednotku pokiaľ zhluk existuje a ponechať nulu ak neexistuje. Pole sa musí pred každým generovaním novej hierarchie vynulovať a zápis nevyžaduje atomickú ochranu do zdieľaného globálneho priestoru. Nad týmto poľom existencii zhlukov sa prevedie operácia prefixového súčtu, čím sa pridelí každému existujúcemu zhluku unikátne poradové číslo, ktoré slúži ako aj referencia vzorky k danému zhluku. Súčtom posledného poradového čísla a poslednej existenčnej značky je dostať celkový počet zhlukov. Po tomto kroku je možné alokovať kompaktný zoznam iba existujúcich zhlukov.



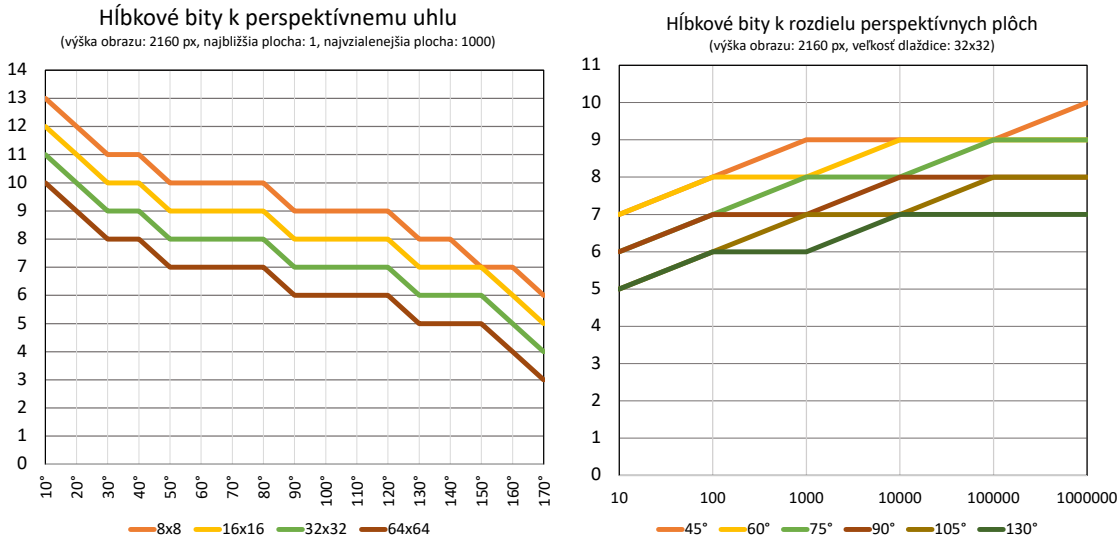
Obr. 3.5: Alternatívny postup kalkulácie unikátnych zhlukov do kompaktného listu pomocou prefixového súčtu.

V aplikácii je použitý ako dátový typ poradové číslo alebo teda identifikátor do kompaktného listu 32 bitové číslo, čo by znamenalo pre 32 bitový kľúč použitý v práci [14] alokáciu približne 17 GiB pamäte. Znížením priestoru kľúča sa výrazne znižuje požadovaná veľkosť pamäte.

3.4.4 Bitová analýza kľúča

Bližšou analýzou generovaných súradníc kľúča sa dajú ušetriť nevyužité drahocenné bity. Hĺbková súradnica je vypočítaná logaritmicky podľa rovnice v sekcii 2.3.4 alebo 2.3.5. Analýzou a dosadením do premenných sa môže obmedziť hĺbková súradnica na istý počet postačujúcich bitov zobrazených na grafoch 3.6.

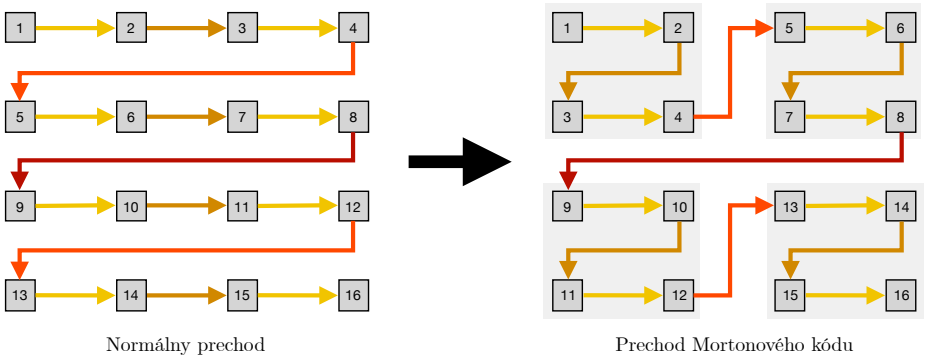
Hĺbková súradnica je závislá na perspektívnom uhle, veľkosti dlaždice a rozdielu medzi najbližšou plochou a najvzdialenejšou hĺbkovou plochou *(z-near, z-far)*. V aplikácii je postačujúcich 9 bitov pre perspektívny uhoľ 90°, hĺbkový rozdiel 10000 a veľkosť dlaždíc 32×32 alebo 16×16. Pre súradnice X a Y postačuje 7 bitov na pokrytie 4K (3840×2160) rozlíšenie s 32×32 dlaždicami alebo 2K (2048×1080) rozlíšenia s dlaždicami 16×16.



Obr. 3.6: Vľavo graf závislosti hĺbkových bitov na perspektívnom uhle s rôznymi veľkosťami dlaždíc. Vpravo hĺbkové bity v závislosti od najbližšej a najvzdialenejšej plochy *(z-near, z-far)* s *rôznymi perspektívnimi uhlami.*

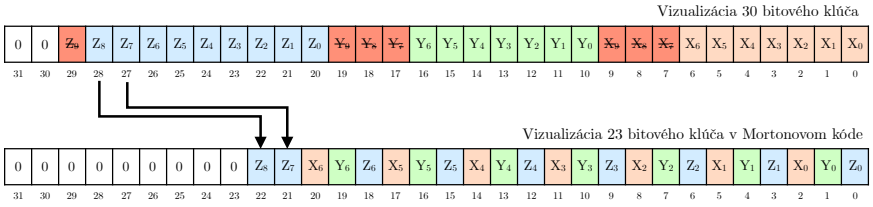
3.4.5 Morton code

V práci [20] je v hierarchii použitá transformácia kľúčových bitov na Mortonov kód. Preskladanie bitov jednotlivých súradníc vytvorí lepší sekvenčný prechod v priestore. Porovnanie prechodu sekvencie kľúča je znázornený na obrázku 3.7 s rozdielom, že v kľúči sa pracuje s trojrozmernými súradnicami.



Obr. 3.7: Vľavo je vizualizácia prechodu kľúča s obyčajnou konkatenáciou súradnicových bitov a vpravo je znázornený prechod kľúča v Mortonovom kóde.

Počet navrhnutých bitov je zhodný s použitými počtami v práci [20], kde je taktiež použitých 7 bitov pre osi X a Y a 9 bitov pre os Z. Pre nezhodnosť počtov bitov je potrebné kľúč po transformácii upraviť ako je zobrazené na obrázku 3.8. Rozdiel od práce [20] je, že os Z, ktorá má viac bitov ako ostatné, je zoradená ako prvá (Obr. 3.8).



Obr. 3.8: Vizualizácia transformácie 30 bitového kľúča s 10 bitovými súradnicami do 23 bitového kľúča v Mortonovom kóde s 7 bitmi pre osi X, Y a 9 bitov pre os Z.

3.4.6 Zhrnutie a vlastnosti

Znížením bitov ostáva kľúč s veľkosťou 23 bitov, čo je zhodná s počtami v práci [20]. Alokovaná pamäť sa rapídne znížila na požadovaných 34MiB pre existenčnú pamäť a pamäť identifikátorov. Veľkosť alokovanej pamäte je závislý na návrhu aplikácie a nie na dynamických dátach a stavu aplikácie. Táto pamäť je síce veľká ale je možné ju ponechať celú počas celého behu aplikácie. Rýchlosť výpočtu unikátnych zhlukov je závislý iba na rýchlosti prefixového súčtu veľkého globálneho poľa a ostáva približne konštantná nezávisle na počte zhlukov alebo veľkosti dlaždíc. Výhodou je aj nezávislosť na veľkosti dlaždíc a tým aj na veľkosti pracovnej skupiny pri optimalizácii. Odhliadnuc na iné okolnosti, pri tejto technike je možné zvoliť ľubovolnú veľkosť dlaždíc podla potrieb.

3.5 Budovanie zhlukov a hierarchie

Každý zhluk pokrýva niekoľko vzoriek, ktoré svojou pozíciou tvoria jeho ohraničujúci kváder *(bounding box)*. Ich pozícia je ale zachytená ako hĺbka v hĺbkovej mape, ktorá musí byť transformovaná projekčnou maticou T_{proj}. Matica vychádza z najbližšej Z_{near} a najvzdialenejšej vzdialenosti Z_{far} perspektívneho pohľadu. Hĺbková mapa ukladá hodnoty v logaritmickej krivke, čo nie je vhodné pre test prieniku s využitím Euklidovskej vzdialenosti. Hĺbka z_{proj} tak musí byť transformovaná do priestoru pohľadu z_{view} vzťahom 3.7.

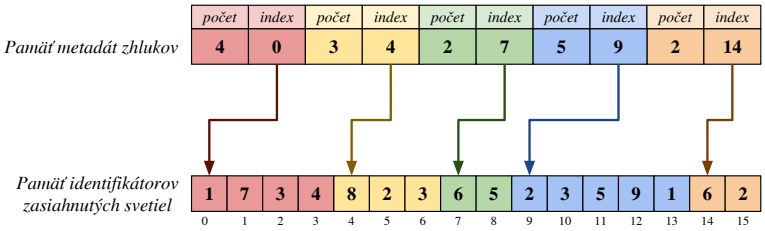
$$z_{view} = \frac{2Z_{near}Z_{far}}{(Z_{far} + Z_{near} - (2z_{proj} - 1) * (Z_{far} - Z_{near}))} \tag{3.7}$$

Pre každý zhluk sa vygeneruje ohraničujúci box (AABB) z pozícii vzoriek (Clustered Shading 2.3.4) a normálový kužeľ *(normal cone)*. Do jednotlivých zhlukov sa budú priraďovať svetlá vygenerované fotónmi s oblasťou vplyvu v tvare gule. Svetlo je teda potrebné testovať prienikom s ohraničujúcim boxom zhluku. Na boxe, definovanom ako bod minima b_{min} a bod maxima b_{max}, sa nájde najbližší bod c ku stredu gule s_{center}. Ak je vzdialenosť medzi bodom c a stredu gule s_{center} menšia ako polomer gule s_r, guľa je v prieniku s boxom (výraz 3.8).

$$|s_{center} - max(v_{min}, min(s_{center}, b_{max}))| < s_r \tag{3.8}$$

Zhluky môžu mať dynamický počet zasiahnutých svetiel, ideálnym spôsobom by bol dynamický zoznam, do ktorého sa len pripíšu ďalšie zdroje, to je však na vzhľadom na architektúru a možnosti GPU nereálne a je potrebné vytvoriť statický zoznam z informácii o celkovom počte elementov.

Všetky svetlá sú zapísané do jedného zoznamu, kde má každé svetlo v scéne atribúty ako farba svetla, pozícia, polomer vplyvu atď. Ako ich identifikátor stačí uložiť index daného svetla v tomto globálnom zozname. V prvom prechode cez všetky zhluky sa spočíta, koľko svetiel zasahuje na vzorky pod zhlukom. Cez všetky počty svetiel sa spočíta celkový počet svetiel a ich index v globálnom poli. Počet a index začiatku zoznamu vplyvných svetiel sú súčasťou dát metadát zhluku, znázornená štruktúra referencii je na obrázku 3.9. V druhom kroku sú známe počty zoznamov a je tak možné vyhradiť pamäť pre zasiahnuté svetlá. K jednotlivým zhlukom je potrebné zapísať identifikátor svetla vo forme indexu.



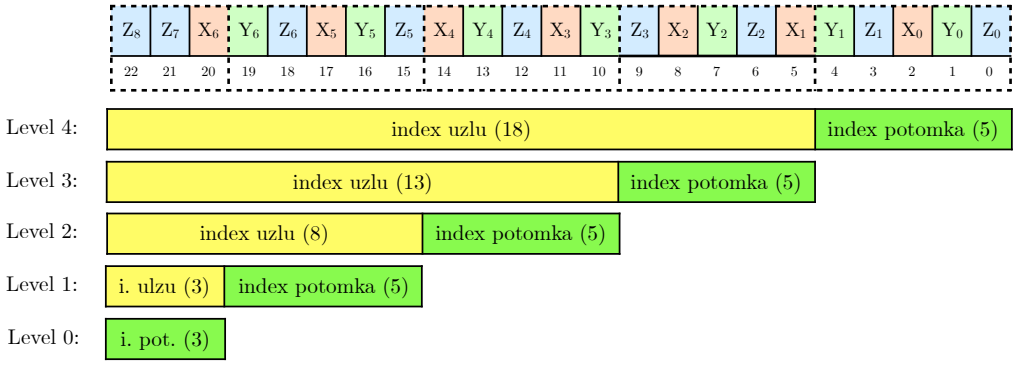
Obr. 3.9: Pamäť metadát zhlukov a referencia do pamäte zoznamu zasiahnutých svetiel.

Je nutné dopredu odhadnúť maximálnu možnú veľkosť pamäte identifikátorov aby jej rozsah obsiahol dátový typ indexu. Problémom je, že nie je dopredu známe aký je maximálny počet zhlukov a s tým ani maximálny počet prienikov so svetlami. Ak je počet zhlukov m a každý je v prieniku s n svetlami, pamäť musí byť veľká m x n identifikátorov svetla.

3.5.1 Hierarchia a nanášanie fotónov

Hierarchia pozostáva z uzlov so štruktúrou podobnou s zhlukmi, kde každý uzol má pod sebou maximálne 32 poduzlov. Toto maximálne číslo je odporúčané v prácach [14] a [20] a je zvolené podla veľkosti registra aby sa s ním dobre pracovalo a nezaberá redundantnú pamäť. Každý uzol má 32 bitovú masku potomkov, ktorá hovorí, či daný potomok existuje alebo nie. V návrhu aplikácie je kľúč o veľkosti 23 bitov v Mortonovom kóde, ktorý je rozdelený na masku poduzlov a priamy index do globálneho poľa úrovne. Posledných 5 bitov ($2^5 = 32$) slúži ako index potomka daného uzlu, čiže hierarchia vychádza na 5 úrovní ($\lceil 23/5 \rceil = 5$) Tento prístup je možný iba s alokáciou celej hierarchie, inak by index zasahoval do pamäte. Rozdelenia kľúča do hierarchie je vizualizované na obrázku 3.10, inšpirovaným z práce [20].

Nanášanie fotónov na uzly alebo zhluky je vykonávané jednoduchým prechodom cez hierarchiu ako je opísané v práci [14] a [20]. Spustenie jedného vlákna pre každý fotón, ktoré prejde cez existujúce uzly hierarchie a testuje, či má fotónová oblasť vplyvu prienik s ohraničujúcim boxom. Ak úplne alebo čiastočne obsahuje box, priradí sa do zoznamu zhluku. Prechod hierarchiou je implementovaný s pomocou malého zásobníka miesto rekurzie, ktorá je často v príslušných API nepodporovaná. Do zásobníka sa najskôr vloží maska poduzlov a spracovávaný kľúč inicializovaný na nulu (koreňový uzol). Prechodom v cykle sa hľadá v maske existujúci poduzol, teda jednotka, ktorú z masky vymaže aby sa neprechádzala znova. Do zásobníka vloží masku poduzla a uloží kľúč spacovávaného uzla. Rekurzívne prechádza poduzly, až kým nenarazí na poslednú úroveň, ktorá má referenciu do kompaktného zoznamu zhlukov. K poduzlom sa pristupuje priamym indexom, ktorý pozostáva z aktuálne spracovávaného kľúča zo zásobníka a indexu poduzla. Aktuálny kľúč sa posunie o 5 bitov doľava a pripočíta sa index aktuálneho poduzla vychádzajúci do rekurzie.



Obr. 3.10: Rozdelenie bitov kľúča do hierarchie. Každá úroveň rozdeľuje kľúč po 5-tich bitoch, ktoré slúžia ako index na potomka.

3.5.2 Vedrá

Druhým spôsobom, ako je opísaný v práci [14], je nepriraďovať fotóny k zhlukom pre ďalšie spracovanie ale vyhodnocovať intenzitu svetla priamo do zhlukov. Ideou predošlého prístupu je, že každá vzorka spracuje všetky priradené svetlá k zhluku, ku ktorému patrí. Vzhľadom na počet priradený fotónov, môže byť tento prístup neefektívny a je v silnej závislosti tejto hodnote.

Prístup s vedrami alebo *buckets*, je opísaný v práci [14] ako ďalší vývoj riešenia globálnej iluminácie v reálnom čase. Prechodom hierarchie sa nevytvára ďalší globálny zoznam priradený fotónov ale sa priamo vyhodnocuje intenzita svetla do jednotlivých uzlov. Každý uzol obsahuje štruktúru, do ktorej sa akumuluje svetlo z rôznych smerov. Táto štruktúra ukladá intenzity v diskrétnych smeroch, prerozdeľuje ich do vedier. Vznikne tak priestorová mapa intenzít pre každý zhluk. Funkcia kvantizácie smerového vektoru môže mať rôzne podoby, v návrhu je použitý spôsob kvantizácie normál spomínaný v práci [14], kde vzniká index kvanta v rozsahu 0 až 53. Pre každú zo 6-tich stranu priestorovej kocky je unikátny index v dvojrozmernej ploche so stranou 3 ($3 \times 3 \times 6 = 54$).

Každý zhluk na použitie kvantizácie svetla, potrebuje štruktúru o veľkosti 648 bajtov (RGB float), čo by spotrebovalo približne ďalších 6 MiB pre 10k zhlukov. Pri prechode hierarchiou je akumuláciu žiarenia nutné vykonávať atomicky. Výsledné fragmenty potom akumulujú svetlo v závislosti na jeho normále z fixného poľa s 54 prvkami prenásobením odporúčaného v práci [14] kosínusového faktoru $2 \cos \vec{n}, \vec{d}$, kde $\vec{n}$ je normála a $\vec{d}$ je kvantizovaný smer fixného poľa. Multiplikácia konštantou 2 je odôvodnená rozdielom kasínového integrálu (π) a integrálu hemisféry (2π) [14].

3.6 Odhad žiarenia

Podstatou globálnej iluminácie je odhadnúť žiarenie odrazeného svetla od objektov v scéne. Poslednou fázou je finálne tieňovanie fragmentov obrazu. Pre každý fragment sa spočítajú prídavky všetkých svetiel v zozname jeho zhluku, do ktorého patrí. Ako je spomínané v sekcii 3.1, výpočet odrazeného difúzneho svetla z prvého odrazu fotónov do miesta pozorovateľa je relatívne efektívne. Tejto princíp je použitý aj pre sekundárne zdroje svetla, plochy, ktoré odrážajú priame lúče späť do scény s rozdielom, že fotóny definujú smer prichádzajúceho žiarenia.

Pre každú vzorku sa prejdú všetky fotóny a akumuluje sa intenzita žiarenia pre daný fragment. Z každej vzorky je potrebné získať jej normálu pre výpočet intenzity difúzneho žiarenia pod uhlom dopadu. Výpočet intenzity je ovplyvnený faktorom vzdialenosti fragmentu od pozície fotónu ako je opísaný v sekcii 3.1.

V technike vedier je tento krok konštantný, kde sa akumuluje prechodom cez všetky kvanty akumulovaného svetla v zhluku. Bližší postup je opísaný v sekcii 3.5.2.

Kapitola 4

Implementácia

V tejto kapitole sú detailnejšie popísané zaujímavé časti implementácie aplikácie. Aplikácia bola vyvíjaná na grafickej karte Intel Iris pod operačným systémom macOS. Časti spracované na grafickej karte sú pod knižnicami OpenGL verziu 4.1 spolu s OpenCL 1.2.

4.1 OpenGL a OpenCL súdržnosť

Táto časť sa venuje prepojeniu knižníc OpenGL a OpenCL, ktoré je kľúčové k implementácii aplikácie. Počas implementácie som narazil na konkrétnejšie problémy, ktoré sa pokúsim vysvetliť pomocou kódu.

Skupina Khronos[1] vytvorila modul *(Khronos Shared Platform Header)*, ktorý spája pamäť kontextov tak aby ich bolo možné používať medzi sebou. Podstatou použitia sú kernely na paralelné spracovanie dát a keďže je aplikácia z návrhu obmedzená na verziu OpenGL 4.1, nie je možné použiť moderný *compute shader*. Spomínané optimalizačné techniky vyžadujú paralelné spracovanie obrazu na grafickej karte s využitím zdielaných pamäťových zdrojov ako textúry a vyrovnávacej pamäte *(buffer)*. Použitie zdielaných zdrojov musí zariadenie tento modul podporovať *(CL_KHR_gl_sharing)*, čo môže spôsobiť problémy s kompatibilitou. Zdieľanie zdrojov vyžaduje vytvoriť špeciálny kontext, ktorý umožní požičiavanie OpenGL pamäte.

```
/* cl_device_id device; cl_platform_id platform; cl_int err; */
cl_context_properties clContextProperties[] = {
    CL_GL_CONTEXT_KHR, (cl_context_properties)wglGetCurrentContext(),
    CL_WGL_HDC_KHR, (cl_context_properties)wglGetCurrentDC(),
    CL_CONTEXT_PLATFORM, (cl_context_properties)platform, 0
};
cl_context ctx = clCreateContext(clContextProperties, 1, &device, NULL, NULL, &err);
```

Prvý krok *clustered shading* je naplnenie G-Buffera (sekcia Deferred Shading 2.3.2 a Clustered Shading 2.3.4) jedným vykresľovacím prechodom do samostatného framebuffera. V modernom OpenGL sa viac-cieľové vykresľovanie jedným prechodom scény robí pomocou pripojovania textúr k framebuffer-u *(FBO)*. Tieto cieľové textúry, nazývané *attachments*, sú bežne vytvorené textúry OpenGL príkazmi a pripájajú sa ako hĺbkový alebo farebné ciele. Fragment shader zaistí zápis dát do týchto pripojených textúr, ktorých obsah je použitý neskôr na spracovanie obrazu v OpenCL kerneloch. Po naplnení FBO dátami je potrebné aby OpenCL nadobudla práva na pamäťe pripojených textúr. Mapovanie textúry na CL objekty vyžaduje mapovanie interného formátu, ktoré závisí od podpory zariadenia.

[1] https://www.khronos.org

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);

cl_mem t = clCreateFromGLTexture(ctx, CL_MEM_READ_ONLY, GL_TEXTURE_2D, 0, tid, &err);
```

Mapovanie hĺbkovej textúry je však problematické, pretože hĺbkový komponent v internom formáte textúry nie je často podporovaný. Novšie verzie OpenCL, ktoré hĺbkové formáty podporujú, nie sú často rozšírené, preto je zvolená relatívne dobre podporovanú verzia 1.2. V článku [1] je písané o verzii 1.1/1.2, kde sa aj okrem iného spomínajú limitácie hĺbkových formátov. Ideálnym prípadnom je mapovanie priamo 24bit hĺbkovej textúry ale treba hľadať iné spôsoby. Jedným zo spôsobov je rozšíriť G-Buffer o ďalšiu farebnú textúru s 32bit formátom, kde sa nejedná už o hĺbkový formát. Fragment shader zaistí zápis hĺbky tiež do tejto textúry, aby bol zhodný s hĺbkovou textúrou. Problém môže nastať, keď zariadenie nepodporuje ani mapovanie tohoto formátu.

4.2 Implementačné riešenia

V tejto kapitole sú rozpísané implementačné problémy a riešenia, na ktoré sa prišlo počas tvorby aplikácie.

4.3 Optimalizácia

Vo fázach, kde sa vyhodnocuje kompaktný zoznam objektov ako zhluky a priradené fotóny je potrebné zistiť ich unikátny počet. V oboch prípadoch sa používa prefixová operácia, ktorá vyhodnotí identifikátory, indexy jednotlivých objektov (3.4.3). Z týchto informácii sa dá získať celkový počet objektov, sčítaním posledného indexu a posledného súčtu. Celkový počet objektov je následne použitý na určenie počtu pracovných jadier alebo prípadnú realokáciu pamäte kompaktného zoznamu.

Táto operácia prebieha na hlavnom vlákne a je potrebné synchronizovať pamäť z grafickej karty do programu. Blokované funkcie čítania z pamäte predlžujú čas vyhodnotenia obrazu. Jednou s možností ako sa tejto blokácii vyhnúť, je predpoklad maximálnej veľkosti zoznamov, ktorú nemôže prekročiť.

Počet unikátnych zhlukov sa počas interakcie pohybuje okolo rovnakých hodnôt, pretože je závislý hlavne na rozložení a zvolenej veľkosti dlaždice. Z viacerých behov aplikácie je vidieť, že počet neprekročí hodnoty pár desiatok tisíc. Vopred alokovaná pamäť sa bude teda pohybovať okolo 1 MiB aj s normálovými dátami a s použitím vedier, aj s rezervou, okolo 20 MiB. Priradené fotóny k zhlukom sú viac nepredvídateľné ale ich dáta sú veľmi kompaktné. Pre každé priradenie fotónu k zhluku stačí 32 bitové číslo. Na alokáciu 10 miliónov priradení je potrebné približne 40 MiB. Vypočítané počty sa teda nemusia synchronizovať a je dostatok pamäte na beh aplikácie. Aj keď optimalizácia priniesla mierne zrýchlenie za cenu pamäte, nie je to výrazná zmena.

4.3.1 Prefixový súčet

Problém prefixového súčtu je samostatná téma, ktorú rieši mnoho knižníc. V článku [6, Chapter 39., Parallel Prefix Sum] je rozpísaná implementácia a optimalizácia paralelného prefixového súčtu. Algoritmus sa prispôsobuje behu grafickej karty a efektívne využíva výpočetné zdroje. Efektívna implementácia zahŕňa využitie zdielanej pamäte v rámci jednej pracovnej skupiny. Idea súčtu nad arbitrárnym polom je vykonávať prefixovú operáciu hierarchicky.

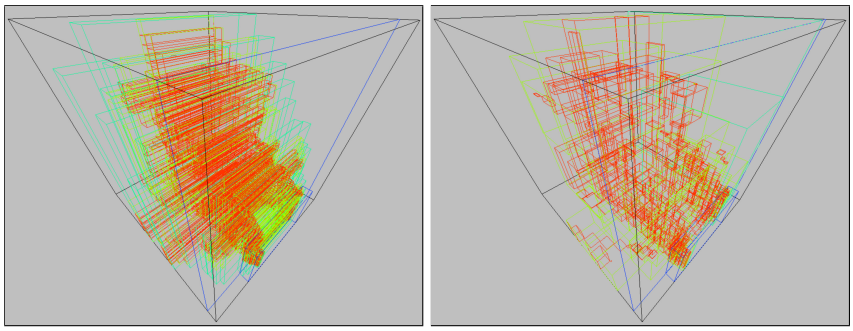
V aplikácii je použitá implementácia prefixového súčtu z knižnice RadeonRays (CLW), ktorá podporuje tri úrovne prefixovej operácie. Maximálna veľkosť jednej pracovnej skupiny je definovaná ako 512 elementov. Na účely globálnej iluminácie je táto implementácia postačujúca, žiadna z pamätí kompaktných zoznamov nepresahuje 512^3 elementov.

4.3.2 Hierarchia

Všetky úrovne hierarchie sú uložené do jedného globálneho poľa. Pri prístupe k jednotlivým úrovniam je potrebné si uchovať počiatočný index prvého uzlu a pričítať jeho hodnotu k relatívnemu indexu úrovne. Posledná úroveň je uložená v kompaktnom zozname zhlukov, ktorých index je taktiež potrebné čítať z poľa unikátnych identifikátorov (3.4).

Na traverz hierarchie je použitý zásobník opísaný v sekcii 3.5.1 a v prácach [14] a [20].

Transformácia kľúča na Mortonov kód má svoje opodstatnenie, bez použitia je hierarchia priestorovo neefektívna ako je vidieť na ukážke 4.1 vľavo. Rozdelenie binárnych číslic rovnomerne medzi dimenzie spôsobí rovnomernejšie rozdelenie zhlukov a uzlov do priestoru ako na obrázku 4.1 vpravo.



Obr. 4.1: Vľavo je hierarchia vytvorená pomocou pôvodného kľúča vpravo vytvorená pomocou kľúča po transformácie do Mortonovho kódu.

4.3.3 Finálne mixovanie svetla

Výstupy osvetlenia scény sú vo forme textúr s rovnakou veľkosťou. Nepriame osvetlenie má výstup iba ako intenzita, ktorú je potrebné vynásobiť difúznou materiálovou vlastnosťou. Výsledná nepriama intenzita pohltená a odrazená materiálom sa akumuluje spolu s intenzitou priameho svetla. Jednou s možností je vytvoriť kernel, ktorý hodnoty príslušne spojí. Všetky výstupy pixelov P_{rgb} sa dajú spojiť vykreslením cez seba s využitím OpenGL mixovania fragmentov (4.1).

$$P_{rgb} = (S_{rgb} \cdot S_f) + (D_{rgb} \cdot D_f) \tag{4.1}$$

Najskôr sa vykreslí nepriama iluminácia preložená materiálovým difúznym odrazom v móde pre-násobenia. S mixovacou funkciou súčtu a kombináciou nastavení zdrojového faktoru S_f na D_{rgb} a cieľového faktoru D_f na 0 zaručí násobenie fragmentov ako v rovnici 4.2.

$$P_{rgb} = (S_{rgb} \cdot D_{rgb}) + (D_{rgb} \cdot 0) \tag{4.2}$$

V ďalšom kroku výsledok prekryje textúra s priamym svetlom s nastavením S_f a a D_f na hodnotu 1 ako v rovnici 4.1.

$$P_{rgb} = (S_{rgb} \cdot 1) + (D_{rgb} \cdot 1) \tag{4.3}$$

Výsledné vrstvy osvetlenia sú zobrazené v ukáže 4.2. Pomocou OpenGL príkazov sa výsledné vykreslenie osvetlenia scény dá zapísať takto:

```
glEnable(GL_BLEND);
// Vykreslenie nepriameho osvetlenia
glBlendFunc(GL_DST_COLOR, GL_ZERO);
// Vykreslenie difuzneho materialu (multiplikativny mix)
glBlendFunc(GL_SRC_ALPHA, GL_ONE);
// Vykreslenie priameho osvetlenia (aditivny mix)
glDisable(GL_BLEND);
```

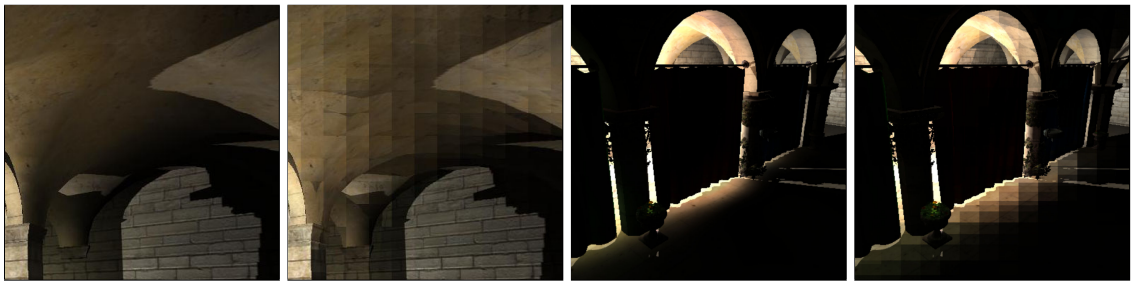
Rozdelenie úrovní svetla má výhody pri ďalšom spracovaní. Izolovaná intenzita žiarenia môže prejsť dodatočnými úpravami pre zjemnenie alebo odstránenie niektorých artefaktov. Finálne mixovanie podporované OpenGL môže byť efektívnejším riešením ako spracovanie v kerneli.

Obr. 4.2: Vľavo je scéna osvetlená iba priamym svetlom, v strede je scéna s nepriamou iluináciou a vpravo je spojenie oboch osvetlení.

4.4 Dosiahnuté výsledky

Návrh a implementácia zahŕňa dva prístupy odhadu žiarenia. Prvý je kalkulácia žiarenia pre každú vzorku zvlášť a druhý ako aproximácia žiarenia pre všetky vzorky cez príslušný zhluk. Druhý spomenutý spôsob je opísaný v práci [14] ako použitie akumulačných vedier *(buckets)*, ktoré uchovávajú intenzitu prichádzajúceho svetla z rôznych strán (v sekcii 3.5.2). Tento prístup je síce výrazne rýchlejší ale jeho kvalita je v surovom stave neporovnateľná, pretože vznikajú veľké skoky intenzít medzi zhlukmi (Obrázok 4.3). Aplikácia dokáže vizualizovať jednotlivé použité časti ako hierarchiu, priradenie zhlukov, priradenie svetiel, vrhnuté lúče alebo oblasti vplyvu na obrázkoch 4.7, 4.8 a na záver kapitoly pár obrázkov výstupu 4.4, 4.5 a 4.6.

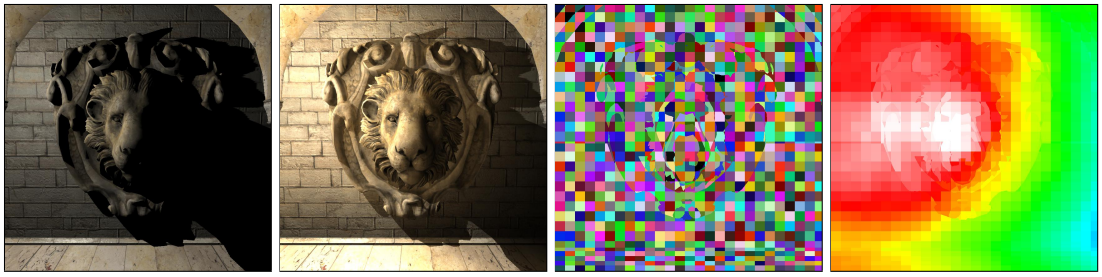
[1]Zdroj: https://casual-effects.com/data/



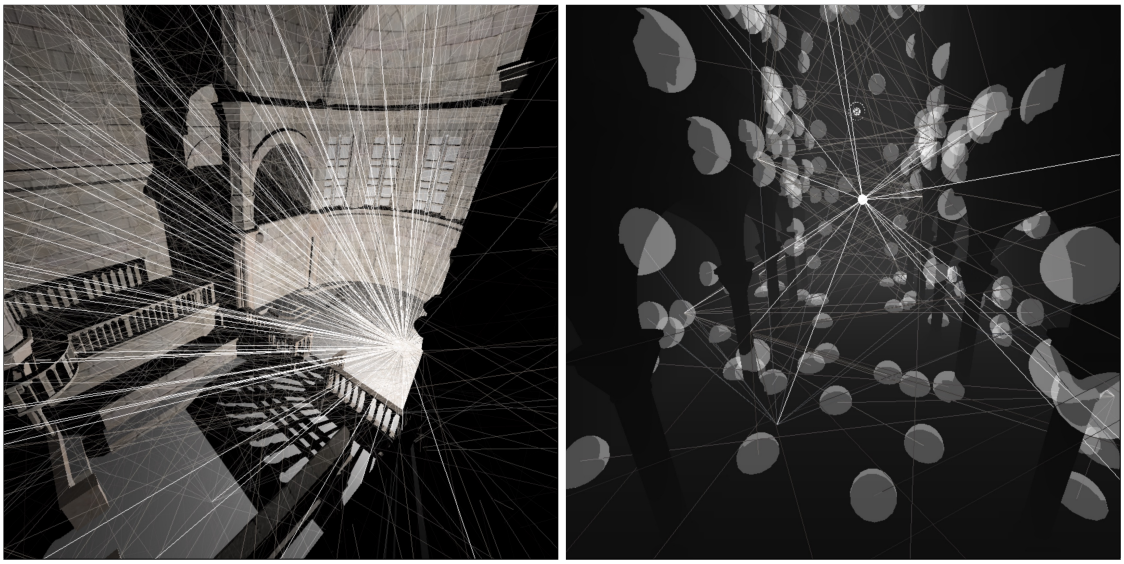
Obr. 4.3: Porovnanie metódy pracujúcej s každou vzorkou zvlášť a metódy s použitím vedier intenzít.

Obr. 4.4: Ukážka osvetlenia scény **Sponza**[1] (Crytek), 17 tisíc fotónov s rádiusom 15.0 bodov cez 4 odrazy na rozlíšení 1920 × 1080.

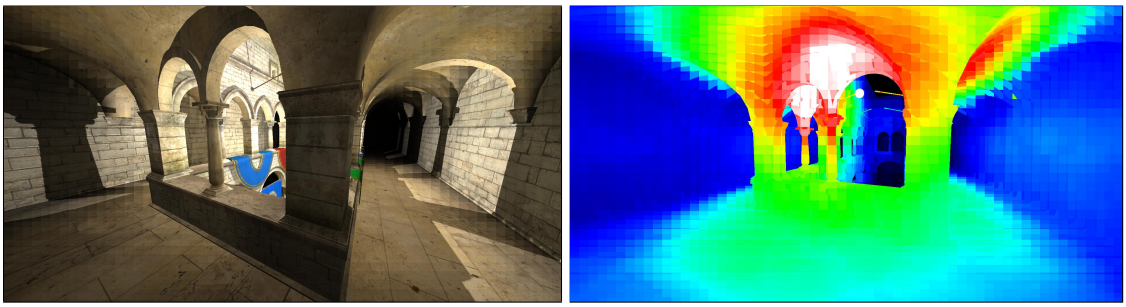
Obr. 4.5: Ukážka osvetlenia scény **Sibenik**[1] (Marko Dabrovic), 17 tisíc fotónov s rádiusom 20.0 bodov cez 4 odrazy na rozlíšení 1920 × 1080.



Obr. 4.6: Vľavo je scéna bez globálneho osvetlenia, na nasledujúcom obrázku s použitím globálneho osvetlenia s 6000 fotónov a rádiusom 12.0, na treťom je vizualizácia zhlukov obrazu a vpravo vizualizácia histogramu fotónov.



Obr. 4.7: Ukážka aplikácie s vizualizáciou trajektóriou vyžiarených fotónov a ich oblasť vplyvu. Slúži na overenie správnosti výpočtov a znázornenie vnútornej štrukúry programu.



Obr. 4.8: Vľavo je výsledok osvetlenia pomocou vedier a vpravo je ukážka histogramu svetiel.

4.5 Budúca práca

V téme *photon splatting* s využitím akceleračnej štruktúry je toho veľa čo vylepšovať. Implementovaná aplikácia má viacero oblastí, kde by sa dala lepšie optimalizovať. Snaha aplikácie bola o dosiahnutí približne podobného výkonu ako v práci [14], čo sa mierne líši od referenčných hodnôt.

Podstatnou zmenou bolo použitie alternatívneho rozdelenia zhlukov, ktoré vynecháva normálové bity v kľúči. Vynechaním tejto informácie je rozdelenie zhlukov iba vďaka pozícii a môže sa stať, že v jednom zhluku budú fragmenty s opačnými normálami. Delenie podla normál a triviálny test s normálovým kužeľ implementované neboli ale môžu znížiť počet priradených fotónov k zhlukom, čo by zrýchlilo poslednú fázu odhadu žiarenia.

Ďalšou zmenou bola implementácia svetla ako bodového zdroja. V porovnaní s prácou [14], kde sú použité ako svetlomety s uhlom odrezania, je potrebné vyžiariť viac lúčov zo zdroja na dosiahnutie menšej chyby. Keď zdroj umiestnený pod nebom, mnoho lúčov zasiahne do prázdneho priestoru a skončí v nekonečne alebo neurčito, čo zbytočne zaťažuje separáciu a spracovanie neplatných lúčov.

Použitie vedier tvorí hranaté artefakty a aj keď nestráca príliš na kvalite, vizuálne nie je prijateľný. Ako je spomenuté v práci [14], tento prístup si žiada dodatočné spracovanie obrazu. Akumulovaná intenzita žiarenia (Obrázok 4.2 v strede), ktorá tvorí kvantizované zhluky osvetlenia sa dá rôznymi spôsobmi zjemniť. Spomína sa rozmazanie obrazcu s ohľadom na hĺbku *(depth-aware blue)*, ktoré zachová hĺbkové rozdiely ale zjemní prechody medzi vzorkami v priestore blízkych pri sebe. Táto operácia môže byť náročná, možno by bolo postačujúce inak interpolačne znížiť kvalitu obrazu. Keďže sa neberie du úvahy vzdialenosť vzorky od polohy fotónu je v tejto metóde dôležitý veľký počet vyžiarených fotónov aby sa chyba osvetlenia znížila na minimum. Na okrajoch oblasti vplyvu vznikajú veľké rozdiely intenzít, ktoré je treba zjemniť zvýšením počtu vrhnutých fotónov (Obrázok 4.3).

Ďalším možným vylepšením je dynamický rádius vplyvu, ktorý je závislý na pravdepodobnosti trajektórie alebo dopadovej vzdialenosti lúča tak ako intenzita. V práci [14] je spomenutá táto zmena ako možná ale s cieľom iluminácie v reálnom čase a prechádzaniu prežiarených oblastí je menej podstatná.

Tento algoritmus rieši iluminániu difúznych materiálov. V práci [14] je spomenuté použitie aj pre iné materiály ale niektoré optimalizácie by sa stávali neplatnými kvôli BRDF závislé na smere pohľadu.

Počas implementácie vznikli problémy súvisiace so zvolenou staršou technológiou, ktoré by sa dali efektívnejšie riešiť modernými, novšími implementáciami. Nahradením týchto kritických častí by sa výkon mohol zlepšiť.

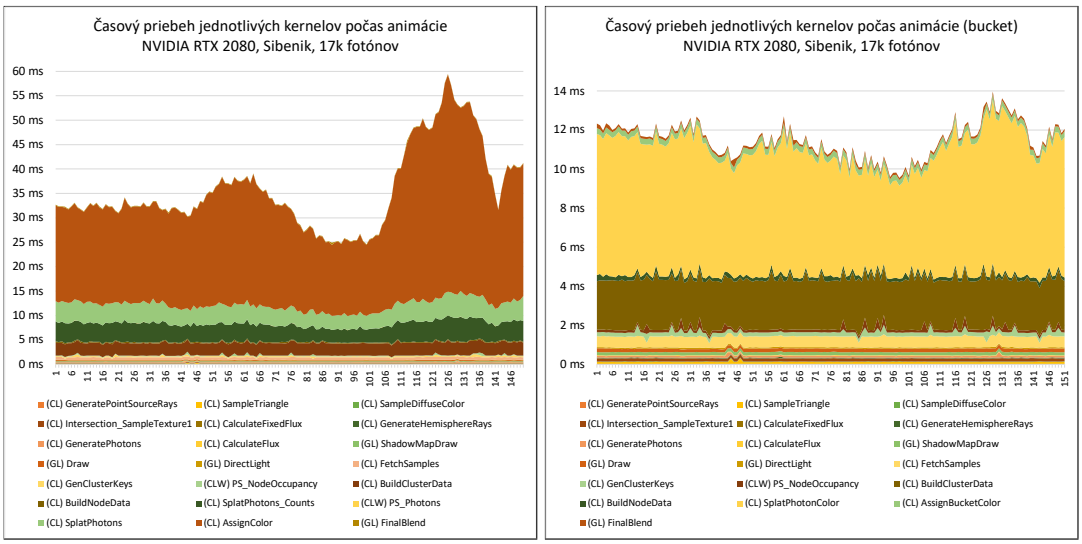
Kapitola 5

Meranie a výsledky

V tejto kapitole je súhrn meraní, výkonnostných a vizuálnych výsledkov implementovanej aplikácie. Rozpísané sú podrobnejšie použité kernely a ich návrh pre presnejšiu identifikáciu problematických a kritických častí algoritmu. Aplikácia bola testovaná na operačnom systéme Windows 10 na grafický kartách Intel Iris, NVIDIA GTX 1080, NVIDIA RTX 2080 a Radeon RX 480.

5.0.1 Výkon a efektivita

Kernel *FetchSamples* má na starosti čítanie z G-Buffera ako hĺbku, normálu a spracuje ich do použiteľného formátu. Normálový vektor z textúrového úložiska n_t je nutné transformovať späť do pôvodného stavu $\vec{n} = \vec{n_t} \cdot 2 - 1$. Hĺbku je potrebné transformovať do priestoru projekcie *(projection space)* a priestoru pohľadu *(view space)*. Tieto dáta sú použité viacerými kernelmi a transformácia vektorov maticami je náročná operácia, preto je lepšie vyhradiť pamäť pre spracované dáta, potom čítanie z textúr a spracovanie prebehne iba raz.

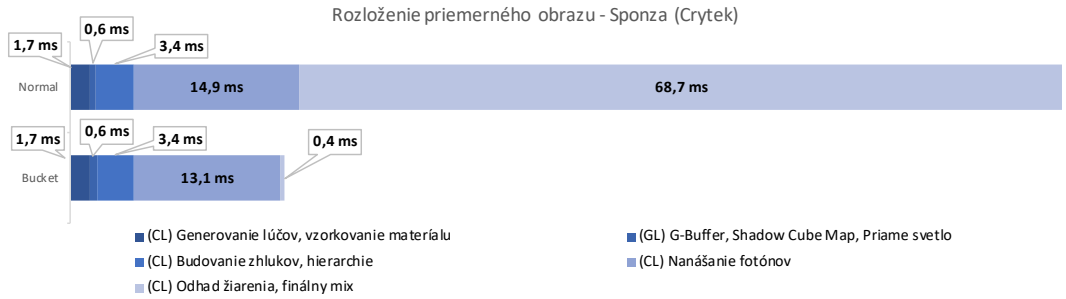


Obr. 5.1

Druhý krok je *GenClusterKeys*, kde sa pre každú vzorku vygeneruje príslušný kľúč a zapíše sa existencia zhlukov. Oba spomenuté kernely sú závislé na počte obrazových

vzoriek, ich zložitosť narastá zvýšením rozlíšenia. Ďalším krokom opísaný v sekcii 3.4.3 je spracovanie existenčného poľa prefixovým súčtom, ktorého zložitosť je počas behu aplikácie približne konštantná. Spracovanie doposiaľ spomenutých kernelov je relatívne efektívne, pri testovaní 2 milióna pixelov sa priemerne vyhodnotí tento krok za 0,68 ms (NVIDIA RTX 2080, 1920×1080), čo je v porovnaní s celkovým časom obrazu zanedbateľné.

Po vygenerovaných pamätí pre zhluky nasleduje vytvorenie hierarchie a naplnenie uzlov dátami cez *(BuildClusterData, BuildNodeData)*. V tomto kroku je potrebné zo vzoriek vytvoriť ohraničujúce boxy atomickými operáciami *min* a *max* nad desatinnými hodnotami. Rozdelenie tohto kroku je odporúčané v práci [20], kde sa rozdelí spracovanie hierarchie na medzi-stupeň *(intermediate level)*. Prístup atomickými operáciami z každej vzorky do štruktúry zhluku spôsobí pravdepodobne priveľa kolízii, čo môže výpočet výrazne zdržať. Implementáciou priameho spracovania vyšlo, že rozdelenie na medzi-krokový kernel zrýchli výpočet približne 8-10 násobne.

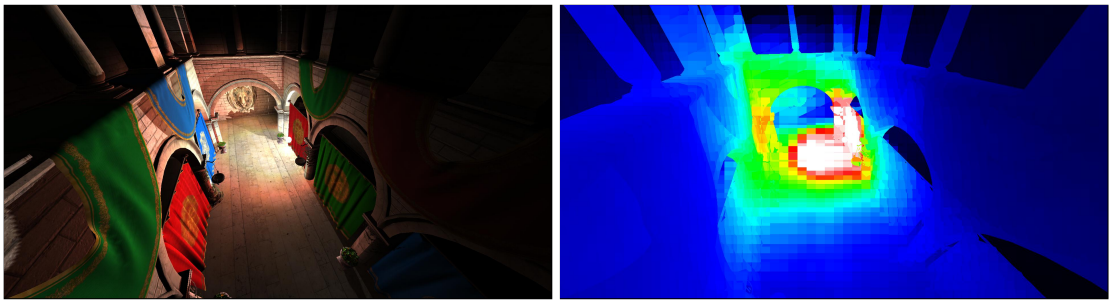


Obr. 5.2: Graf rozdelenia priemerného vyhodnotenia obrazu na jednotlivé úrovňe spracovania. Horný graf je technika spracúvajúca všetky vzorky so všetkými priradenými fotónmi a dolný graf je rozdelenie vyhodnotenia obrazu pre techniku vedier *buckets*.

Ďalším krokom je samotné nanášanie fotónov prechodom cez vytvorenú hierarchiu. Tento krok je výrazný v oboch použitých technikách, kde sa počítajú prieniky fotónov s ohraničujúcimi boxami a v prípade techniky s vedrami *(buckets)* ide o kľúčový krok v efektivite. V prvej technike je potrebné prejsť hierarchiu dvoma prechodmi. Prvý prechod hierarchie každým fotónom zapíše počet pridelených fotónov do zhluku pre celkový počet fotónových pridelení a v druhom po alokácii kompaktného zoznamu sa pre každý zhluk priradí fotón do globálneho zoznamu. Atomickou operáciou subtrakcie z počtu pridelených fotónov pre zhluk sa dá zistiť unikátny index, ktorý predchádza paralelným kolíziám prístupu do poľa.

Ďalším krokom je prefixový súčet nad všetkými počtami ako inicializácia kompaktného globálneho zoznamu priradení. Pre techniku s vedrami platí len zápis intenzity fotónu atomickými operáciami do vedier kandizovaných smerov. V oboch prípadoch sa v testovacej vzorke vyhodnotil prechod hierarchiou približne za 10-15 ms vrátane dvojitého prechodu prvej techniky.

Posledným, finálnym krokom je odhad žiarenia, ktorý je v prípade vedier takmer zanedbateľný v porovnaní s poctivou prvou technikou. Komplexnosť algoritmu vedier finálneho kroku je závislá iba na počte kvantov pri kvantizácii smerového vektoru. Naopak pri prvej technike je algoritmus silne závislý na počte fotónov. Tento krok je v tejto technike najkritickejší, jeho zložitosť je priamo úmerná počtu spracovaných fotónov k zhluku a má



Obr. 5.3: Vizualizácia počtu priradených fotónov do zhlukov vykreslenej scény. Na obrázku vpravo je priradená čierna farba pre počty blízke nule cez modrú, zelenú, červenú až bielu, ktorá dosahuje v tomto prípade 3750 fotónov na zhluk. Vzniká tak nevyváženie výkonu pri paralelnom psracovaní.

dominantné zastúpenie v čase vyhodnotenia obrazu ako je vidieť na obrázku 5.2. Na grafoch 5.1 je rozpis všetkých kernelov a ich zastúpenie v testovacom prechode scény.

Je zrejmé, že jeden zhluk spracováva viac fotónov ako druhý a vzniká tak problém vyvážaného výkonu, kde sa v paralelizácii musia vlákna na seba čakať. Na obrázku 5.3 vpravo, je vizualizácia pridelených počtov fotónov do zhlukov. Čierna farba znamená počet blízky nule a biela farba dosahovala do hodnôt 3750 fotónov na zhluk. Riešením by bolo rozdelenie jednotlivých fotónov v spojení so vzorkou ako podúlohou do globálneho poľa a potom spustiť ich paralelné spracovanie, kde si každé vlákno vyzdvihne jednu pod úlohu až kým nie je zoznam prázdny [14].

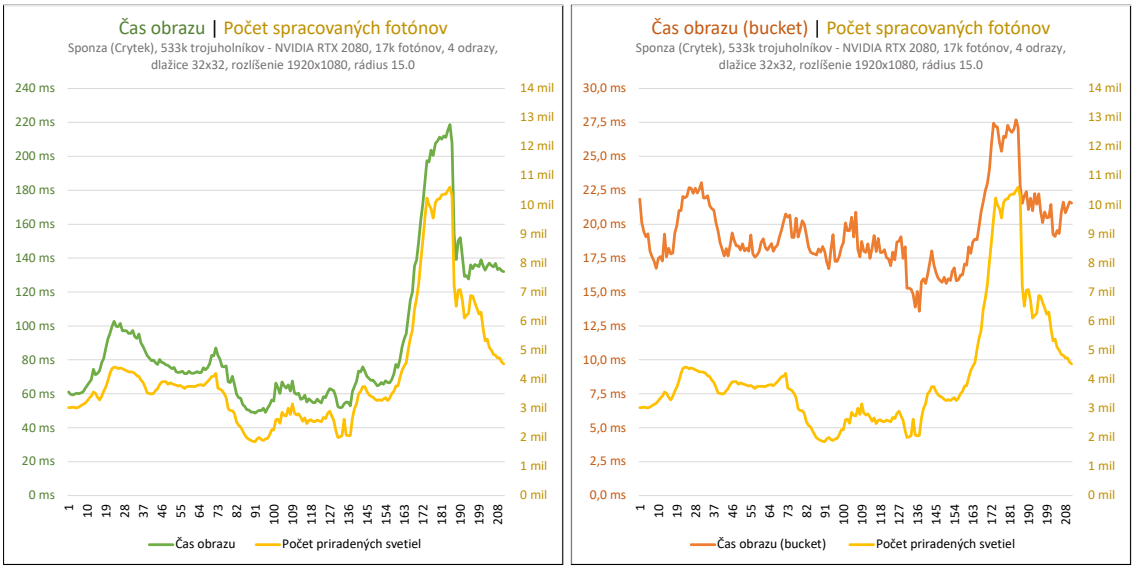
5.0.2 Vizuálne výsledky

Každá scéna je iná a treba zvoliť pri každej správne parametre. Rádius je pre všetky fotóny konštantný a závisí na ňom kvalita a chyba globálneho osvetlenia. V prvej technike s použitím vzdialenostného faktoru je možné pri väčšom rádiuse vrhať menej lúčov, to však tvorí artefakty ako na obrázku 5.4, ktoré nie sú realistické. Väčší rádius je možné použiť pri scénach s veľkým otvoreným priestorom so zriedkavejšími detailami. V opačnom prípade je vhodnejšie zvoliť menší rádius a vyrovnať to väčším počtom fotónov.

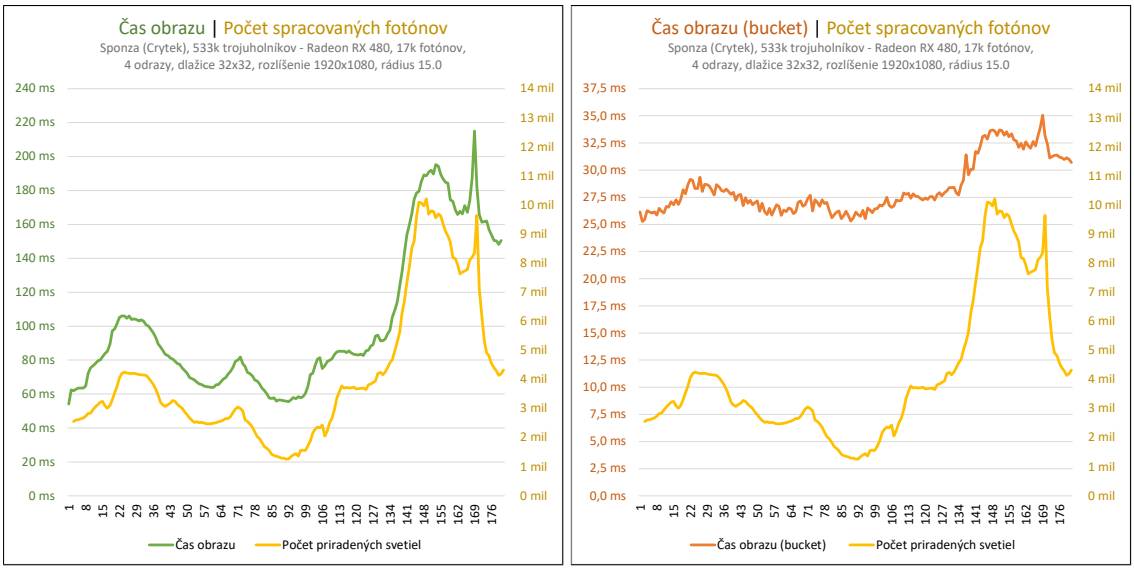
Obr. 5.4: Ukážka vizuálnej chyby pri zvolení príliš veľkého fotónového rádiusu.

Na grafike Radeon RX 480 obraz globálnej iluminácie vytvára "duchov", teda zanecháva za sebou artefakty pravdepodobne z predošlého obrazu. Predpoklad je, že je to spôsobené

vo finálnom kroku mixovania textúr, čo je pravdepodobne spôsobené tým, že má inú implementáciu synchronizácie textúr ako zvyšné testované grafické karty. Finálna fáza tak omylom používa pamäť z vrstvy predošlého obrazu.

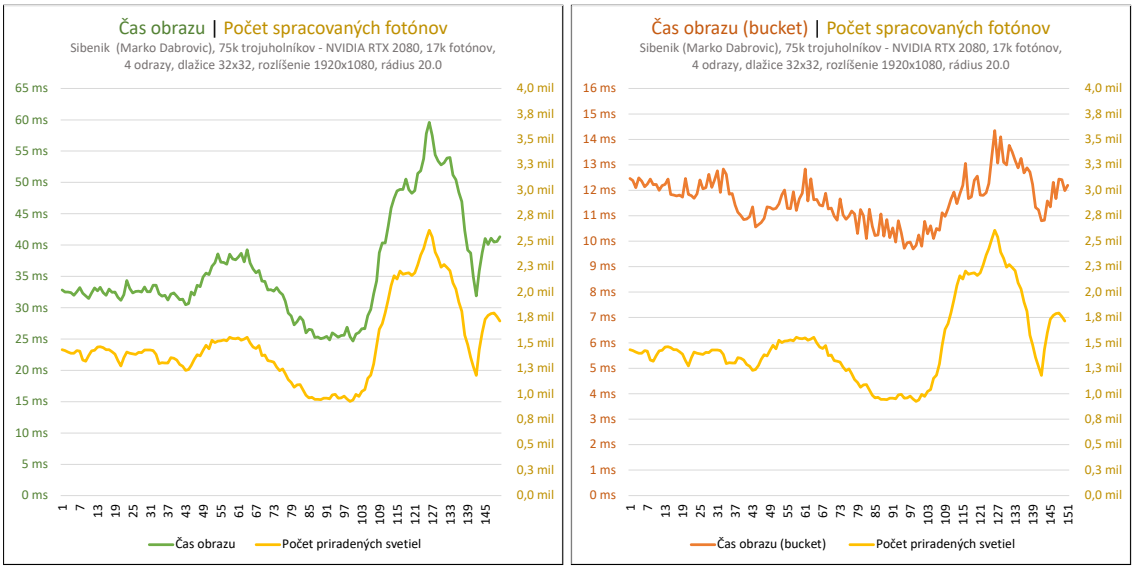


Obr. 5.5: Meranie na grafike **NVIDIA RTX 2080**, scéna **Sponza**[1] (Crytek), 553 tisíc trojuholníkov, 17 tisíc fotónov s rádiusom 15.0 bodov cez 4 odrazy na rozlíšení 1920×1080. **Vľavo** je technika cez všetky vzorky a **vpravo** je s použitím optimalizačných vedier. *Časy obrazu sú v porovnaní s celkovým počtom vyhodnocovaných svetiel, priradených k zhlukom v jednom obrazci.*

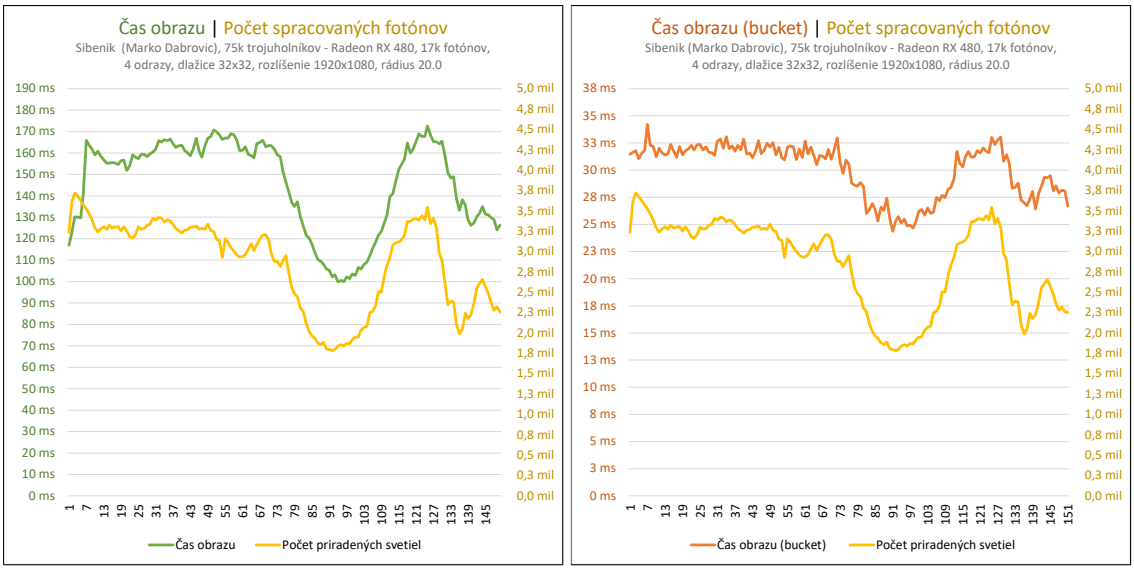


Obr. 5.6: Meranie na grafike **Radeon RX 480**, scéna **Sponza**[1] (Crytek), 553 tisíc trojuholníkov, 17 tisíc fotónov s rádiusom 15.0 bodov cez 4 odrazy na rozlíšení 1920×1080. **Vľavo** je technika cez všetky vzorky a **vpravo** je s použitím optimalizačných vedier. *Časy obrazu sú v porovnaní s celkovým počtom vyhodnocovaných svetiel, priradených k zhlukom v jednom obrazci.*

[1] Zdroj: https://casual-effects.com/data/



Obr. 5.7: Meranie na grafike **NVIDIA RTX 2080**, scéna **Sibenik**[1] (Marko Dabrovic), 75 tisíc trojuholníkov, 17 tisíc fotónov s rádiusom 20.0 bodov cez 4 odrazy na rozlíšení 1920×1080. **Vľavo** je technika cez všetky vzorky a **vpravo** je s použitím optimalizačných vedier. *Časy obrazu sú v porovnaní s celkovým počtom vyhodnocovaných svetiel, priradených k zhlukom v jednom obrazci.*



Obr. 5.8: Meranie na grafike **Radeon RX 480**, scéna **Sibenik**[1] (Marko Dabrovic), 75 tisíc trojuholníkov, 17 tisíc fotónov s rádiusom 20.0 bodov cez 4 odrazy na rozlíšení 1920×1080. **Vľavo** je technika cez všetky vzorky a **vpravo** je s použitím optimalizačných vedier. *Časy obrazu sú v porovnaní s celkovým počtom vyhodnocovaných svetiel, priradených k zhlukom v jednom obrazci.*

[1] Zdroj: https://casual-effects.com/data/

Kapitola 6

Záver

V tejto práci boli spomenuté niektoré globálne iluminačné metódy, ich výhody a nevýhody. Detailnejšie popísané boli optimalizačné techniky predchádzajúce metódu *photon splatting* zameranej na spracovanie v reálnom čase. Cieľom bolo analyzovať, implementovať, merať a navrhnúť rozšírenie zameranej techniky.

Implementovaná aplikácia demonštruje algoritmus *photon splatting using a view sample cluster hierarchy* s využitím knižnice OpenGL a OpenCL. Dokáže vykresliť scény v dvoch iluminačných prístupoch opísané v sekciách 3.6 a 3.5.2. Prvý prístup generuje vizuálne kvalitnejšie výsledky ale s výpočetne náročnejším algoritmom. Druhý prístup generuje výsledok so zhoršenou kvalitou, ktorý je možné zlepšiť dodatočnými úpravami ale vhodnejší pre aplikácie v reálnom čase. S vizualizačnými nástrojmi je prehľadná vnútorná štruktúra techniky pracujúca so vzorkami, hierarchiou a fotónmi. Metóda sa dá vylepšiť ďalšími prístupmi spracovania opísaných ako budúca práca v sekcii 4.5.

V sekcii 3.4 je navrhnutý alternatívny prístup zisťovania unikátnych zhlukov do kompaktného zoznamu. S využitím poznatkov z práce [20] sa podarilo implementovať tento spôsob využívajúci priestor hierarchie na úkor niektorých zlepšení z práce [14]. Navrhnuté riešenie šetrí výpočetný čas a využitú pamäť.

Metódy boli otestované na viacerých grafických kartách na viacerých scénach. Aplikácia má isté oblasti, kde by sa dala vylepšiť z pohľadu optimalizácie kódu alebo spracovania dát, čo by mohlo prispieť k lepšiemu výkonu spracovania obrazu. Metódy boli odmerané a analyzované v sekcii 5, kde sú rozpísané jednotlivé časti a návrh ich zlepšenia.

Literatúra

[1] Anteru: OpenCL for realtime rendering: What's missing? 2011.
URL https://anteru.net/blog/2011/opencl-for-realtime-rendering-what-s-missing/

[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] Crassin, C.; Neyret, F.; Sainz, M.; aj.: Interactive Indirect Illumination Using Voxel Cone Tracing: A Preview. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, New York, NY, USA: ACM, 2011, ISBN 978-1-4503-0565-5, s. 207–207, doi:10.1145/1944745.1944787.
URL http://doi.acm.org/10.1145/1944745.1944787

[4] Dachsbacher, C.; Stamminger, M.: Reflective Shadow Maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, New York, NY, USA: ACM, 2005, ISBN 1-59593-013-2, s. 203–231, doi:10.1145/1053427.1053460.
URL http://doi.acm.org/10.1145/1053427.1053460

[5] Enderton, E.; Sintorn, E.; Shirley, P.; aj.: Stochastic Transparency. *IEEE Transactions on Visualization and Computer Graphics*, ročník 17, č. 8, August 2011: s. 1036–1047, ISSN 1077-2626, doi:10.1109/TVCG.2010.123.
URL http://dx.doi.org/10.1109/TVCG.2010.123

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

[7] Jensen, H. W.: Global Illumination Using Photon Maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, London, UK, UK: Springer-Verlag, 1996, ISBN 3-211-82883-4, s. 21–30.
URL http://dl.acm.org/citation.cfm?id=275458.275461

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

[9] Kaplanyan, A.; Dachsbacher, C.: Cascaded Light Propagation Volumes for Real-time Indirect Illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, New York, NY, USA: ACM, 2010, ISBN 978-1-60558-939-8, s. 99–107, doi:10.1145/1730804.1730821.
URL http://doi.acm.org/10.1145/1730804.1730821

[10] Kircher, S.; Lawrance, A.: Inferred Lighting: Fast Dynamic Lighting and Shadows for Opaque and Translucent Objects. In *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, Sandbox '09, New York, NY, USA: ACM, 2009, ISBN 978-1-60558-514-7, s. 39–45, doi:10.1145/1581073.1581080.
URL http://doi.acm.org/10.1145/1581073.1581080

[11] Mara, M.; Luebke, D.; McGuire, M.: Toward Practical Real-time Photon Mapping: Efficient GPU Density Estimation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-1956-0, s. 71–78, doi:10.1145/2448196.2448207.
URL http://doi.acm.org/10.1145/2448196.2448207

[12] Matt Pharr, W. J.; Humphreys, G.: Surface Reflection. ročník 5.6, 2004-2018.
URL http://www.pbr-book.org/3ed-2018/Color_and_Radiometry/Surface_Reflection.html

[13] Matt Pharr, W. J.; Humphreys, G.: Surface Reflection. ročník 5.6, 2004-2018.
URL http://www.pbr-book.org/3ed-2018/Monte_Carlo_Integration.html

[14] Moreau, P.; Sintorn, E.; Kämpe, V.; aj.: Photon Splatting Using a View-sample Cluster Hierarchy. In *Proceedings of High Performance Graphics*, HPG '16, Goslar Germany, Germany: Eurographics Association, 2016, ISBN 978-3-03868-008-6, s. 75–85, doi:10.2312/hpg.20161194.
URL https://doi.org/10.2312/hpg.20161194

[15] NVIDIA, C. E.: Interactive Order-Independent Transparency. 2001, [Online; navštíveno 08.01.2019].
URL https://pdfs.semanticscholar.org/99b8/940b5a6dab8527198e966c0eb7e2a02ee28c.pdf?_ga=2.211794325.1830523783.1546992215-221913375.1546992215

[16] Olsson, O.; Assarsson, U.: Tiled Shading. *Journal of Graphics*, ročník GPU, 11 2011: s. 235–251, doi:10.1080/2151237X.2011.621761.

[17] Olsson, O.; Billeter, M.; Assarsson, U.: Clustered Deferred and Forward Shading. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, Goslar Germany, Germany: Eurographics Association, 2012, ISBN 978-3-905674-41-5, s. 87–96, doi:10.2312/EGGH/HPG12/087-096.
URL https://doi.org/10.2312/EGGH/HPG12/087-096

[18] 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.
URL http://delivery.acm.org/10.1145/850000/844181/p41-purcell.pdf

[19] 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.
URL https://pdfs.semanticscholar.org/d81c/63e0b38b26401621adf8edcc53ce8e0849d7.pdf

[20] Sintorn, E.; Kämpe, V.; Olsson, O.; aj.: Per-triangle Shadow Volumes Using a View-sample Cluster Hierarchy. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '14, New York, NY, USA: ACM, 2014, ISBN 978-1-4503-2717-6, s. 111–118, doi:10.1145/2556700.2556716.
URL http://doi.acm.org/10.1145/2556700.2556716

[21] Tomás Umenhoffer, L. S.-K., Balázs Tóth: *Efficient Methods for Ambient Lighting*. [Online; navštíveno 08.01.2019].
URL https://dl.acm.org/citation.cfm?id=1980482

[22] Wynn, C.: *An Introduction to BRDF-Based Lighting*. [Online; navštíveno 16.01.2019].
URL http://vision.ime.usp.br/~acmt/hakyll/assets/files/wynn.pdf

Prílohy

Príloha A

Obsah přiloženého paměťového média

- **bin/**
  - **mac-x86-64/** - *Zložka pre binárne súbory macOS Mojave*
    - **run/...** - *Zložka pre binárne súbory macOS Mojave k okamžitému spusteniu aplikácie*
      - **xkissm01-photon-splatting** - *Preložená aplikácia pre macOS Mojave*
  - **win-x64/** - *Zložka pre binárne súbory Windows x64*
    - **cmake_app/...** - *Program CMake*
    - **libraries/...** - *Skompilované potrebné knižnice na preloženie aplikácie*
    - **run/...** - *Zložka obsahujúca potrebné súbory (.dll, .exe) k okamžitému spusteniu aplikácie*
      - **xkissm01-photon-splatting.exe** - *Preložená aplikácia pre Windows x64*
- **latex/...** - *Projekt textovej časti, dokumentácie (LaTeX)*
- **proj/**
  - **CMakeLists.txt** - *Súbor na preloženie cez CMake*
  - **CMakeModules/...** - *Prídavné moduly pre CMake*
  - **include/...** - *Zložka obsahujúca podzložky knižníc tretích strán zahrnutých do projektu vo forme hlavičkových súborov*
  - **libs/...** - *Zložka obsahujúca podzložky knižníc tretích strán zahrnutých do projektu vo forme zdrojových súborov*
  - **src/...** - *Zdrojové súbory aplikácie*
- **video/...** - *Videá z implementovanej aplikácie*
- **README.txt** - *Inštrukcie na kompiláciu zdrojových kódov, ovládanie aplikácie, obsah média*
- **xkissm01-photon-splatting.pdf** - *Dokumentácia odbornej práce*