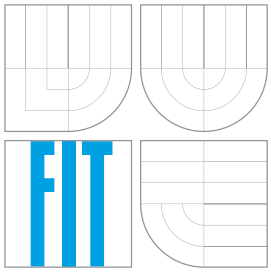
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

LIGHT PROPAGATION VOLUMES

LIGHT PROPAGATION VOLUMES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ RŮŽIČKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MILET

BRNO 2016

Zadání diplomové práce/18581/2015/xruzic42

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Růžička Tomáš, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Light Propagation Volumes**
Light Propagation Volumes
Kategorie: Počítačová grafika

Pokyny:

1. Nastudujte techniky real-time globální iluminace
2. Implementujte metodu Light Propagation Volumes s využitím OpenGL. Dbejte na přenositelnost kódu na různé platformy (Windows/Linux, NVidia/AMD, ...). Využijte pro implementaci kritických částí compute shadery nebo OpenCL.
3. Navrhněte rozšíření metody Kaskádových Light Propagation Volumes (například: zachování tvaru sférické harmonické, buňky ve tvaru pravidelného čtyřstěnu, ...).
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 a práci zveřejněte na internetu.

Literatura:

- http://dl.acm.org/citation.cfm?id=1730821
- http://www.crytek.com/download/Light_Propagation_Volumes.pdf

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a 3.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí: **Milet Tomáš, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

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

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem diplomové práce je popsat různé metody výpočtu globálního osvětlení scény včetně techniky Light Propagation Volumes. Pro tuto metodu jsou podrobně popsány všechny tři kroky výpočtu: injekce, propagace a vykreslení. Dále je navrženo několik vlastních rozšíření zlepšující grafickou kvalitu metody. Části návrhu a implementace jsou zaměřeny na popis scény, zobrazovacího systému, tvorby stínů, implementace metody Light Propagation Volumes a navržených rozšíření. Práci uzavírá měření, porovnání výsledných obrázků aplikace při různých parametrech výpočtu a shrnutí práce se zhodnocením dosažených výsledků s návrhy pro budoucí vylepšení.

Abstract

The aim of master thesis is to describe different calculation of global illumination methods including Light Propagation Volumes. All three steps of LPV calculation are widely described: injection, propagation and rendering. It is also proposed several custom extensions improving graphics quality of this method. Two parts of design and implementation are focused on scene description, rendering system, shadow rendering, implementation of LPV method and proposed extensions. As conclusion, measurement and several images of application are presented, followed by comparison in environment with diffenent parameters, thesis summary with evaluation of achieved results and suggestions of further improvements.

Klíčová slova

propagace světla mřížkou, počítačová grafika, mapování stínů, Reflective Shadow Maps, sférické harmonické, globální iluminace, nepřímé osvětlení, engine, zobrazování, paralelní výpočty, Compute Shader, OpenGL, GLSL

Keywords

Light Propagation Volumes, computer graphics, Shadow Mapping, Reflective Shadow Maps, spherical harmonics, indirect lighting, engine, rendering, parallel computing, Compute Shader, OpenGL, GLSL

Citace

RŮŽIČKA, Tomáš. *Light Propagation Volumes*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Milet Tomáš.

Light Propagation Volumes

Prohlášení

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

.....
Tomáš Růžička
18. května 2016

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Tomáši Miletovi za jeho podporu a odbornou pomoc, kterou mi poskytl při vedení této diplomové práce.

Obsah

Kapitola 1

Úvod

Techniku *Light Propagation Volumes* (obr. 1.1) lze v oblasti počítačové grafiky považovat za relativně novou metodu, jak v reálném čase vypočítat globální osvětlení scény a tím se více přiblížit reálnějším výsledkům při vykreslování trojrozměrné grafiky. Přesněji řečeno se jedná o metodu, která počítá globální osvětlení scény pomocí *virtuálních světel* vytvořených technikou *Reflective Shadow Maps*, kdy je vykreslována scéna z pozice okolních světel do sady *textur*. Tato světla se poté přemění na koeficienty *sférických harmonických funkcí*, které jsou vloženy do mřížky a v ní iterativně propagovány. Podobně je vytvářena mřížka zastínění a výstupem těchto výpočtů je sekundární osvětlení, které je využito spolu s klasickými technikami přímého osvětlení scény.

Obrázek 1.1: Porovnání vizualizace scény vlevo s přímým nasvícením a stíny s obrázkem vpravo s přidaným nepřímým nasvícením vypočteným pomocí metody *Light Propagation Volumes*.

Globální osvětlení se v počítačové grafice řadilo a stále řadí mezi výpočetně náročné procedury, a proto se v minulosti stínovací metody při výpočtu v reálném čase uchylovaly k různým aproximacím a kritickým zjednodušováním technik. Mezi ně patří například lokální osvětlovací modely nebo předpočítání osvětlení do světelných map. Později, s výkonnějšími grafickými adaptéry a možností programovat grafický řetězec bylo možné provádět další výzkum s výsledkem nových stínovacích mechnik.

V dnešní době jsou grafické čipy na tak obecné úrovni, že jsme je schopni využít pro

výpočet libovolného paralelizovatelného problému, který by byl pro klasický procesor obtížně řešitelný. Toho právě využívá většina grafických technik zmíněných v této práci. V blízké budoucnosti pak lze očekávat nástup *grafických API nové generace*, mezi které patří například *Vulkan*, *DirectX 12* nebo *Mantle*. Hlavním přínosem těchto knihovem je nízkoúrovňový přístup ke grafickému hardwaru, možnost snížit vytížení procesoru a naplno využít výpočetní potenciál grafických čipů. Proto lze v nejbližších letech v této oblasti očekávat mnoho nových výzkumů a grafických technik.

Tato práce je zaměřena na již objevené techniky výpočtu globálního osvětlení v reálném čase a především na metodu *Light Propagation Volumes*. Proto je ve druhé kapitole pouze stručně představen pojem osvětlovacích modelů. Ve třetí kapitole jsou popsány real – time techniky globálního osvětlení, které se v menší či větší míře používají v dnešní trojrozměrné počítačové grafice – zejména ve videohrách. Výběr metod je omezen pouze na tzv. interaktivní metody, které jsou plně dynamické, tedy okamžitě reagují na změnu scény. Mezi tyto metody patří také výše zmiňovaná technika *Light Propagation Volumes*, která je podrobně popsána ve čtvrté kapitole. V páté kapitole jsou nastíněny možné rozšíření této techniky, v šesté kapitole je popsán návrh demonstrační aplikace a v sedmé kapitole její implementace. Nakonec jsou v osmé kapitole změřeny výsledky metody a ukázány výstupy práce. V deváté kapitole jsou shrnuty dosažené výsledky diplomové práce s návrhy pro možná vylepšení do budoucna.

Kapitola 2

Osvětlovací modely

Za osvětlovací modely lze v počítačové grafice považovat množinu algoritmů, které určují způsob výpočtu osvětlení ve scéně. Tato kapitola se pouze stručně zaměří na základní rozdělení osvětlovacích modelů, jejich hlavní rysy, výhody a nevýhody. Lze říct, že osvětlovací modely určují výslednou barvu daného bodu modelu, přičemž tento bod je poté zobrazen na projekčním plátně. Barva bodu závisí na mnoho faktorech, například typu materiálu objektu, vlastnostmi prostředí a pozorovatele. Samotný výpočet pak může být pro různé druhy osvětlovacích modelů diametrálně odlišný. Metody je možné v základu rozdělit na lokální a globální, případně na interaktivní a neinteraktivní.

2.1 BRDF a zobrazovací rovnice

Většina osvětlovacích metod pro realistické zobrazování vychází z tzv. *zobrazovací rovnice*, která využivá distribuční funkci *BRDF* (neboli *Bidirectional Reflectance Distribution Function*). Tato funkce udává vizuální vlastnosti povrchu objektu, kdy pro každý bod povrchu je definována míra odrazivosti dopadajícího světla ke směru pozorovatele [17]. Pro funkci jsou důležité tři podmínky: *linearita*, *reciprocita* a *zachování energie*.

Funkce *BRDF* je dána zápisem:

$$f_r(x, \omega_i \rightarrow \omega_o) = \frac{\mathrm{d}L_r(\omega_o)}{\mathrm{d}E(\omega_i)} = \frac{\mathrm{d}L_r(\omega_o)}{L_i(\omega_i)\cos(\theta_i \mathrm{d}\omega_i)}[sr^{-1}] \tag{2.1}$$

kde:

$\mathrm{d}L_r(\omega_o)$	je odražená zář
$\mathrm{d}E(\omega_i)$	je ozáření povrchu
$\mathrm{L}_i(\omega_i)$	je světlo dopadající ze směru ω_i
θ_i	je úhel dopadajícího světla a normály povrchu

Samotná *zobrazovací rovnice* pak definuje přenos světla scénou jako součet emitované a odražené záře. Základní rovnice v úhlové formě je definována zápisem:

$$L_o(\mathrm{x}, \vec{\omega_o}) = L_e(\mathrm{x}, \vec{\omega_o}) + \int_{\Omega} f_r(\mathrm{x}, \vec{\omega_i}, \vec{\omega_o}) L_i(\mathrm{x}, \vec{\omega_i}) \cos(\Theta) \mathrm{d}\vec{\omega_i} \tag{2.2}$$

kde:

x	je bod v prostoru
$\vec{\omega_o}$	je vektor odcházející záře
$\vec{\omega_i}$	je vektor přicházející záře
$L_o(\mathrm{x}, \vec{\omega_o})$	je celková odcházející zář
$L_e(\mathrm{x}, \vec{\omega_o})$	je emitovaná zář
$f_r(\mathrm{x}, \vec{\omega_i}, \vec{\omega_o})$	je *BRDF* přeneseného světla z ω_i do ω_o
$L_i(\mathrm{x}, \vec{\omega_i})$	je přicházející zář

Nutno podoknout, že kvůli nutnosti integrace přes celou hemisféru Ω je analytické řešení této rovnice, v konečném čase, prakticky nemožné. Proto si osvětlovací modely tuto rovnici různě zjednodušují, vzhledem ke schopnostem daného algoritmu, času, visuálnímu výsledku a jiných kritérií. Grafický výstup pak nemusí postihnout všechny jevy, jako například světelné odlesky, odrazy, lom světla, přenos světla povrchem, atd...

2.2 Lokální a globální osvětlovací modely

Lokální osvětlovací modely patří mezi jednodušší techniky, protože řešení zobrazovací rovnice zjednodušují pouze na přímé osvětlení. Nepřímé osvětlení – *globální iluminaci* ignorují. Mezi klasické zástupce lze zařadit *Gouraudovo* a *Phongovo* stínování, které lze jednoduše implementovat v *shaderech* a z toho důvodu jsou interaktivní.

Nicméně globální osvětlovací modely počítají také se světelnými odrazy, a proto je výpočet takového osvětlení řádově složitější. Kromě metod zmíněných v kapitole 3, zde patří také různé varianty metod *sledování paprsku*.

2.3 Interaktivní a neinteraktivní metody

Pokud osvětlovací modely rozdělíme podle časové náročnosti, je možné, podle jejich přístupu, je nazvat interaktivní nebo neinteraktivní (resp. *online* nebo *offline*). Interaktivní techniky jsou zaměřeny na rychlý výpočet a okamžitou odezvu při změně scény (např. odlišné osvětlení), přičemž výsledek nemusí být vždy zcela přesný. Do této kategorie lze zařadit techniky implementovatelné v *shaderech*, například již zmíněné lokální metody: *Goraudovo*, *Phongovo* stínování a globální metody, které jsou popsány v kapitole 3.

Neinteraktivní metody se obvykle liší svým přístupem k vykreslování a scénu nekreslí po objektech ale po pixelech. Takový výpočet v minulosti probíhal čistě softwarově pomocí procesoru, avšak v dnešní době je implementace možná i pomocí grafické karty. Tyto techniky jsou zaměřeny především na realisticky vypadající grafický výstup a samotný výpočet pak může trvat minuty, hodiny nebo dny. Do této kategorie patří např.: metody *sledování paprsku* a případně by se zde dala zařadit, v minulosti ve videohrách hojně využívaná, technika *světelných map* (*Lightmap*).

Světelná mapa je textura obsahující mj. světelnou informaci a tato textura je nanesena na trojrozměrný objekt ve scéně [18]. Vypočtení světelné informace závisí čistě na zvolené metodě a obvykle se volí technika *sledování paprsku* případně *radiozita*. Tímto způsobem je možné zobrazit předpočítané lokální nebo globální osvětlení včetně stínů na předmětech scény a samotné zobrazení scény lze vykonat v reálném čase. Hlavním problémem této metody je její statičnost, případně dlouhá doba výpočtu světelných map a svázanost s konkrétní geometrií. Pokud ve scéně nastane nějaká změna, která ovlivní nasvícení okolních

objektů, potom je potřeba světelné mapy vypočítat znova. Což není, z důvodu složitého výpočtu, v krátkém časovém intervalu možné. V minulosti, např. enginy videoher tento problém řešily předpočítáním několik úrovní světelných map pro různé situace ve scéně.

Kapitola 3

Interaktivní metody globální iluminace

Tato kapitola se zabývá popisem metod výpočtu globální iluminace ve scéně, přičemž výběr metod je omezen na tzv. interaktivní techniky. Tyto techniky jsou schopny reagovat na dynamické scény, nemusí vycházet z žádných předpočítaných dat a pro výpočet nového snímku scény využívají buď aktuálně dostupná data nebo data z několika předchozích snímků.

3.1 Instant Radiosity

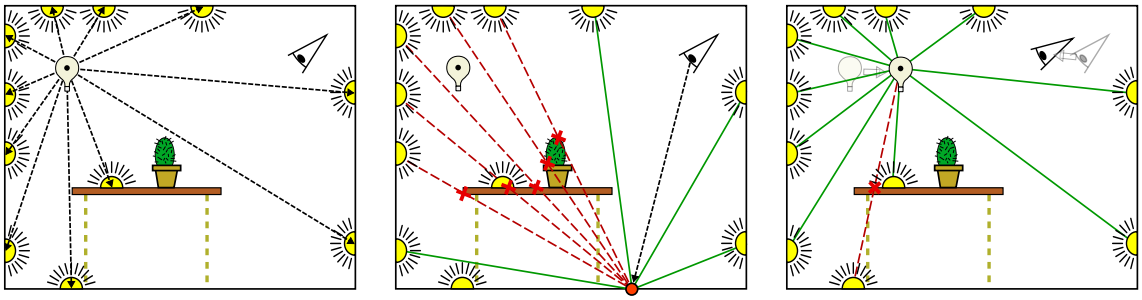
Jedná se o základní metodu globální iluminace, kterou uvedl Alexander Keller v roce 1997 [12]. Zavádí v ní koncept tzv. *Virtual Points Lights* (dále jen *VPL*), které představují sekundární zdroje světla. Cílem metody je z pozice zdroje světla trasovat částice pomocí *kvazi – náhodné procházky*, dokud nenarazí na objekt ve scéně. V těchto místech jsou pak vytvořena nová *VPL*. Následně je z pozice všech *VPL* vykreslena scéna se stíny a výsledek je dán součtem těchto kreslení.

Techniku dále rozšiřuje metoda *Incremental Instant Radiosity* [14], která používá pro určení viditelnosti *VPL* prokládané stínové mapy. Metoda dále optimalizuje situaci, kdy se zdrojové světlo pohybuje (obr. 3.1). V tom případě se znovu negenerují stínové mapy ze všech *VPL*, ale každé *VPL* se prověří, zda není v zákrytu ke zdrojovému světlu. Pokud je zastíněné, odstraní se. Případný pohyb kamery nemá vliv na generování stínových map.

Bohužel tato metoda potřebuje velký počet *VPL*, aby dokázala dostatečně simulovat globální iluminaci. Navzdory tomu, koncept *VPL* využívají i další metody, mezi které patří i technika *Reflective Shadow Maps*. Ta je popsána v sekci 3.2.

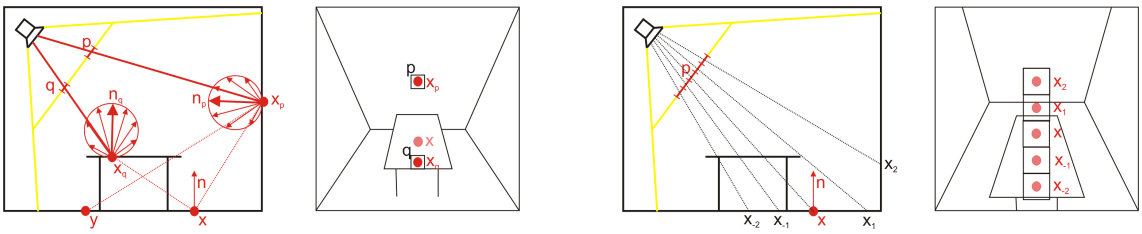
3.2 Reflective Shadow Maps

Pokročilejší technikou je *Reflective Shadow Maps*, která svým principem vychází z metody *stínových map*, kdy je scéna vykreslována z pozice světelného zdroje do textury – *stínové mapy*. Na rozdíl od stínových map, kde jsou body textury využity k určení, zda je vykreslovaný bod osvětlen nebo ne, u této techniky každý bod přestavuje sekundární (virtuální) světelný zdroj (*VPL*) [4]. Pro toto použití je potřeba vycházet z předpokladu, že jsou využity pouze bodové světelné zdroje. Kromě textury hloubky, tato metoda využívá také textury pro světové souřadnice bodů, normálu a odraženou záři. Ve výsledku tato sada textur může připomínat tzv. *G – buffer* pro techniky *odloženého stínování*.



Obrázek 3.1: Ukázka metody *Incremental Instant Radiosity*. Vlevo: vytvoření *VPL* (žluté) na objektech scény. Uprostřed: určení viditelných a zakrytých *VPL* z pozice bodu pixelu kamery ve scéně. Vpravo: prověření všech *VPL*, zda jsou viditelné po pohybu světla. Převzato z [14].

Po získání potřebných textur je možné pro určitý bod x (obr. 3.2 vlevo) spočítat odraženou intenzitu světla jako sumu intenzit všech *VPL*. Nicméně počet všech *VPL* může být díky vyššímu rozlišení textur poměrně značný, proto je možné zmenšit rozlišení textur nebo vzorkovat texturu nízkým a pevně stanoveným počtem *VPL*. S využitím druhé možnosti je pak cílem vybírat nejvhodnější body pomocí předpokladu: body, jež jsou v textuře blízko sebe, jsou také ve scéně nedaleko (obr. 3.2 vpravo). V tom případě je textura vzorkována postupem, kdy je nejprve vypočtena projekce bodu $x \rightarrow x_p$ do textury s *VPL* a poté jsou náhodně vybírány konkrétní *VPL* kolem bodu x_p při snižující se hustotě vůči vzdálenosti od x_p. *VPL* není třeba vybírat zcela náhodně, ale je možné vytvořit předpočítanou masku pro vzorkování textury. Hodnoty zvolených bodů jsou také pomocí vzdáleností nakonec váhovány.



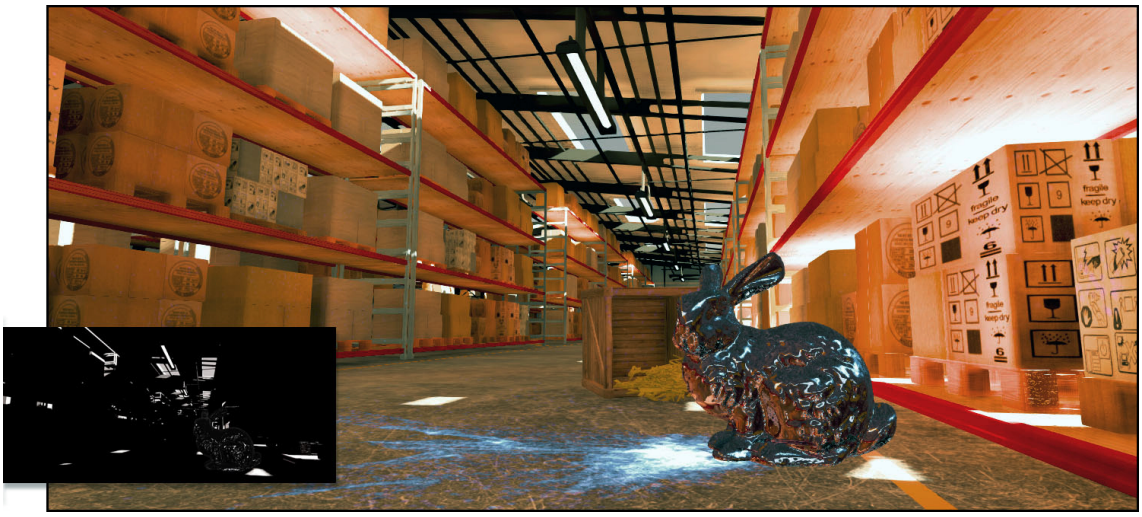
Obrázek 3.2: Vlevo: princip *Reflective Shadow Maps* – každý z bodů textury *RSM* (např. p a q) představuje nový světelný zdroj – *VPL* (x_p a x_q), kterými mohou být osvětlené body x a y. Vpravo: při výpočtu osvětlení bodu x je vzorkováno okolí tohoto bodu v textuře pro získání bodů x_{-2} až x_2. Převzato z [4].

Dalším možným rozšířením metody, pro urychlení vykreslování v reálném čase je postup výpočtu nepřímého osvětlení pouze pro podvzorkované výstupní rozlišení, přičemž při finálním vykreslování jsou pixely, v kladném případě, interpolovány mezi nejbližšími vzorky. V opačném případě, kdy jsou normály nejbližších bodů velmi rozdílné, je potřeba vypočítat hodnotu z plného rozlišení. Nicméně tato optimalizace je vhodná pouze pro jednoduché a rovné scény. Pro komplexní scény není možné použít podvzorkované nepřímé osvětlení.

3.3 Photon Mapping

Původní metodu představil Henrik Wann Jensen v roce 1995 [8], která se skládá ze dvou částí: v první části jsou ze světelných zdrojů emitovány fotony scénou a při dopadu fotonu na objekt je tento foton, vč. jeho pozice, uložen do fotonové mapy. Ve druhé části jsou vrhány paprsky z kamery a po dopadu na objekt je vypočtena intenzita světla pomocí nejbližších nalezených fotonů ve fotonové mapě.

V případě rychlé GPU implementace je možné pro první část algoritmu použít buď rasterizaci nebo paralelizované sledování paprsku optimalizované například pomocí k – D stromu [22]. Díky tomu lze sledovat téměr miliardu paprsků za sekundu. Druhou částí se zabývají autoři článku [15], kde popisují různé metody shromažďování fotonů pro výpočet intenzity (obr. 3.3). Podle přístupu k výpočtu jsou tyto metody rozděleny na dvě třídy: *rozptylu* a *shromažďování* (obr. 3.4).

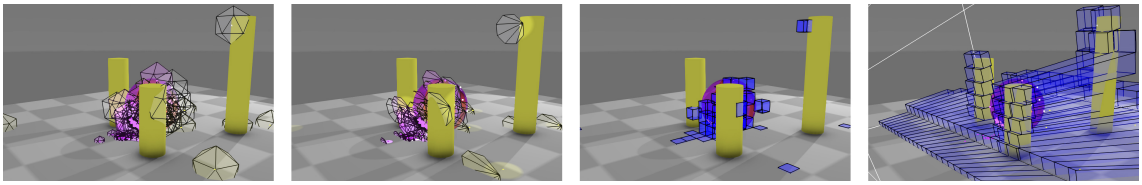


Obrázek 3.3: Scéna skladiště stínovaná metodou *Photon Mapping* s rozlišením 1920 × 1080 při 36 snímcích za sekundu pomocí grafické karty: *NVIDIA GeForce 670*. Výřez zobrazuje přímé nasvícení s prioritizovanými oblastmi. Převzato z [15].

První třída využívá zavedený grafický řetězec pro vykreslování 3D primitiv, jež reprezentují ohraničení fotonů. Toto primitivum může být buď 3D mnohostěn nebo 2.5D hraniční těleso (*billboard*). Po rasterizaci lze získat všechny viditelné body blízko fotonů, které se využijí pro výpočet příspěvku intenzity daného fotonu.

Druhá třída je založena na *shromažďování* fotonů pro daný pixel – algoritmus je tedy postaven obráceně. Pro výpočet je již nutné využít *compute shadery* (případně *CUDA*) a při dotazování na blízké fotony v prostoru může být využita 3D *hašovací tabulka* nebo 2D dlaždice, které odpovídají pozici fotonu v oblasi obrazovky.

Z výsledků vyplývá, že nejefektivnější jsou přístupy kreslení 2.5D hraničních primitiv a při zjednodušení stínovacího modelu pouze na Lambertův osvětlovací model, je nejvhodnější metoda hledání fotonů pomocí 2D dlaždic díky *compute shaderům*.



Obrázek 3.4: Vizualizace různých přístupů k fotonům (kamera je postavena nalevo směrem doprava). Zleva doprava: 3D mnohostěny, 2.5D hraniční tělesa, buňky hašovací tabulky, pohledová frusta 2D dlaždic. Převzato z [15].

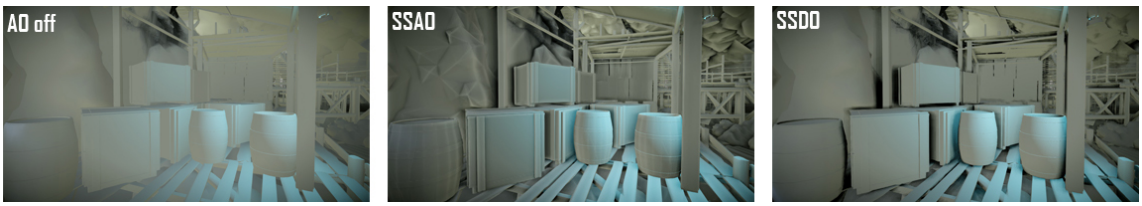
3.4 Screen Space Ambient Occlusion

Ambient Occlusion je metoda, která určuje míru zastínění povrchu nebo bodu povrchu okolními předměty. Taková oblast se pak jeví tmavší, než je okolní ambientní osvětlení [23]. Analyticky je možné zastínění pro konkrétní bod vypočítat jako integrál funkce viditelnosti přes hemisféru okolo bodu, která je určená jeho normálou. Z výše uvedeného je zřejmé, že výpočet je třeba zjednodušit, například na sumu náhodně vystřelených paprsků.

Nicméně pro interaktivní grafiku je výpočet stále složitý, především pro rozsáhlé scény, a proto lze výpočet zastínění provádět pouze v oblasti obrazovky. Z toho vychází metoda *Screen Space Ambient Occlusion* (*SSAO*) [16], která po rasterizaci využívá pro výpočet paměť hloubky (obr. 3.5 uprostřed). *SSAO* pro daný pixel čte několik náhodně vybraných okolních vzorků s paměti hloubky a porovnává jejich vzdálenost s referenční vzdáleností. Pro výběr okolních vzorků, může být použito rotační jádro se sadou pseudonáhodných hodnot a vysokofrekvenční šum lze odstranit pomocí rozmazání s odchylkou stejnou jako je velikost rotačního jádra. Techniku uvedl *Crytek* v roce 2007.

3.5 Screen Space Direct Occlusion

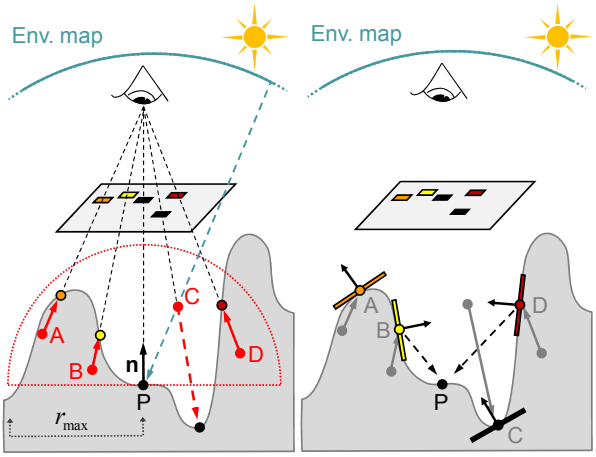
Aby oblasti blízkých ploch nebyly pouze ztamevné, ale také aby dokázaly odrážet difuzní barvu přilehlých ploch (obr. 3.5 vpravo), nastupuje technika *Screen Space Direct Occlusion* (*SSDO*) [19], představená v roce 2009. Metoda vychází z použití $G-buffer$u, který mj. obsahuje pozice a normály rasterizované scény, přičemž výsledek stínování počítá ve dvou krocích: nejprve pro přímé osvětlení a poté pro nepřímé.



Obrázek 3.5: Ukázka použití technik zatínění v herním enginu *CryEngine 3* (*Crytek GmbH*). Vlevo: bez zastínění, uprostřed: *Screen Space Ambient Occlusion*, vpravo: *Screen Space Direct Occlusion*. Převzato z webu[2].

[2] http://docs.cryengine.com/display/SDKDOC1/CryENGINE3+SDK+DX11+Features

Při výpočtu přímého osvětlení je pro každý pixel s pozicí a normálou ve scéně vypočtena záře a viditelnost pomocí několika okolních bodů scény. Pro určení viditelnosti vzorků je využita zpětná projekce bodů a narozdíl od předešlé metody, se pro výpočet využijí pouze viditelné body (obr. 3.6 vlevo). Ve druhé části výpočtu nepřímého osvětlení představují okolní body (z předchozího kroku) malé orientované plošky, které obsahují přímé nasvícení. Díky normále dané plošky pak lze určit, které plošky budou přispívat svou intenzitou do původního bodu (obr. 3.6 vpravo).



Obrázek 3.6: Postup výpočtu *SSDO*. Vlevo: pro výpočet přímého nasvícení bodu P je vygenerováno několik náhodných bodů v okolí: A až D. Zpětnou projekcí, lze zjistit, že je osvícen pouze bod C, a proto je bod P osvícen pouze tímto bodem. Vpravo: body A až D představují plošku, která uchovává přímé nasvícení bodu, jež je využito při výpočtu nepřímého nasvícení. Převzato z [19].

Jeden z možných problémů této metody je omezení určení zastínění, kdy lze získat informaci pouze o nejbližším zastiňujícím objektu a tím pádem může být nekorektně určeno sekundární osvětlení. Tento problém může vyřešit technika *depth peeling* nebo výpočet z několika odlišných pozic kamery. Díky tomu lze získat více informací o prostoru scény a dosáhnout lepších výsledků, avšak tato rozšíření znatelně degradují výkonnost metody.

3.6 Subsurface Scattering

Na rozdíl od předešlých metod, které se zabývaly osvětlením objektů scény pomocí nepřímých odrazů, se tato část zaměřuje na případy, kdy světlo po dopadu na objekt není odraženo, ale prostoupí jím skrz a rozpýlí se (obr. 3.7). Efekt je možné simulovat několika způsoby [7]. Pro základní použití je možné upravit stínování tak, aby difuzní osvětlení ze zdroje osvítilo povrch i v případě, že úhel mezi normálou povrchu a dopadajícím světlem je větší než 90°. Nicméně tento jednoduchý způsob nevytváří příliš kvalitní výsledky.

Pokročilejší alternativou je použití hloubkové mapy, díky které lze lépe simulovat absorbci rozptýleného světla. Svým principem je tato metoda podobná mechanizmu stínových map a pro správný výpočet je třeba scénu vykreslit dvakrát. Nejprve je scéna vykreslena z pozice světla pro zaznamenání hloubky scény do textury. Poté je textura projektována zpět a scéna je vykreslena standardním způsobem z pozice kamery. Při druhém vykreslování je

pro každý pixel získána hloubka, která přestavuje bod výstupu paprsku světla z objektu. Z pozice bodu je pak vyhledán bod v textuře z předchozího kroku ve formě vzdálenosti od světla. Získané hodnoty představující vzdálenosti od sebe je nakonec nutné odečíst, díky čemuž lze získat přibližnou délku, jakou by musel paprsek urazit skrz objekt scény.

Obrázek 3.7: Příklad z WebGL aplikace demostrující průsvitnost postav. Převzato z webu[4].

Problémem metody je, že dokáže dobře pracovat pouze s konvexními objekty, ale předměty se záhyby a dírami nemusí být stínovány korektně. Tomu lze zabránit například pomocí techniky *depth peeling*.

3.7 Voxel Cone Tracing

Jedná se o relativně novou metodu z roku 2012 určenou k výpočtu globální iluminace s difuzními odrazy a neostrými odlesky (obr. 3.8). Základním principem je voxelizace scény do formy hierarchické struktury, kterou představuje oktalový strom (*Sparse Voxel Octree*), kde voxely obsahují data reprezentující zář. Po vytvoření stromu jsou do listových uzlů vloženy informace o dopadajícím světle ze světelných zdrojů a tyto informace jsou následně propagovány do vyšších úrovní stromu. Nakonec, při vykreslování scény z pohledu kamery je pro každý viditelný pixel, z pozice objektu, vrženo několik kuželů scénou, které posbírají světelnou informaci z různých úrovní stromu [3].

Výhodou hierarchické struktury je její nezávislost na komplexnosti scény a díky tomu poskytuje poměrně rovnoměrnou složitost při výpočtu. Také lze při výpočtu použít nejnižší – tedy nejpodrobnější úroveň stromu pro blízké voxely a při zvyšující se vzdálenosti od kamery využít úrovně vyšší. Samotná konstrukce stromu probíhá pomocí rasterizace, kdy je scéna vykreslena ze směrů os X, Y a Z s rozlišením ekvivalentním počtu voxelů na nejnižší úrovni stromu. Pro každý vzorek pak *fragment shader* postupuje stromem shora dolů a rozgenerovává konkrétní uzly. Voxely nejsou uloženy ve stromové struktuře přímo, ale ve 3D textuře spolu s ostatními voxely, které představují tzv. *bricks* (cihly). Pro statickou scénu je dostačující provádět postup generování stromu pouze při inicializaci scény, avšak

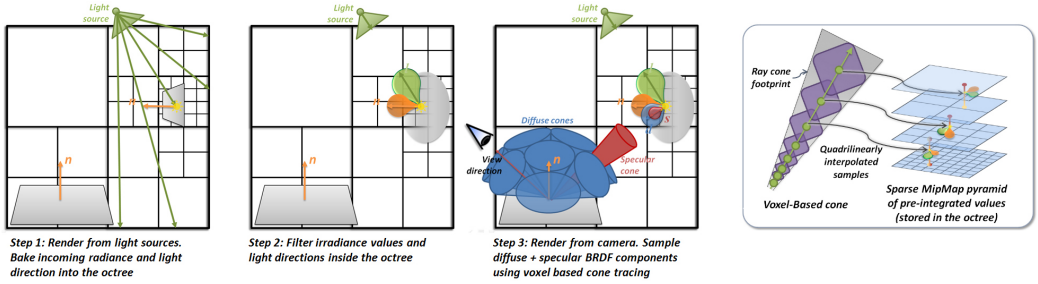
[4] http://www.pixelnerve.com/webgl/sss/

Obrázek 3.8: Ukázka globálního osvětlení a neostrých odlesků počítaných pomocí metody *Voxel Cone Tracing*. Převzato z [3].

dynamické objekty je potřeba voxelizovat při výpočtu každého snímku. Nutno podotknout, že statické i dynamické objekty scény využívají stejný oktalový strom.

Po vytvoření struktury stromu následuje krok vložení žáře světelných zdrojů do listových voxelů stromu (obr. 3.9 vlevo). To se provádí pomocí techniky *Reflective Shadow Maps*, která byla blíže popsána v sekci 3.2. Jakmile je žář v listových voxelech uložena, je propagována (filtrována) do vyšších úrovní stromu (obr. 3.9 druhý zleva). Daný voxel pak provede vážený průměr hodnot voxelů z nižší úrovně.

Po propagaci světla je možné scénu vykreslit z pozice kamery. V tu chvíli je pro každý pixel projekčního plátna, který reprezentuje určitou pozici a normálu bodu ve scéně, emitováno několik kuželů s různou orientací, které sbírají zář ze stromové struktury (obr. 3.9 druhý zprava). Tento sběr je postaven na principu postupného procházení struktury pomocí kužele po určitém kroku, kdy v každém kroku má kužel zvyšující se průměr. Tato hodnota pak určuje úroveň stromu pro vyhledání žáře voxelů, jež spadají do oblasti kužele (obr. 3.9 vpravo). Získaná žář je pak sečtena s výsledky ostatních kuželů.



Obrázek 3.9: Postup metody *Voxel Cone Tracing*. Zleva doprava: vložení žáře z přímého osvětlení do listových uzlů oktalového stromu, filtrace světla do vyšších úrovní stromu, vykreslení a získání osvětlení – kdy z pozice pixelů ve scéně je vrženo několik kuželů, princip sběru záře z voxelů. Převzato z [3].

Kapitola 4

Light Propagation Volumes

Tato kapitola je věnována metodě *Light Propagation Volumes* určenou pro výpočet globálního osvětlení v reálném čase. V sekcích níže je metoda podrobněji popsána a dále je využita v implementaci aplikace (kapitola 7). Tuto techniku představili pracovníci v roce 2009 z německého studia *Crytek*, které se věnuje vývoji her, jež jsou postaveny na herním enginu *CryEngine*. Aktuálně studio nabízí jeho třetí verzi, která je multiplatformní s podporou různých grafických a jiných prvků, mezi které patří i technika stínování *Light Propagation Volumes*.

Jak název napovídá, tato metoda je založena na prograpaci nepřímého světla objemem nebo lépe řečeno trojrozměrnou mřížkou [9]. V základu je tato technika schopna zobrazit druhý odraz světelného toku (první nepřímý) včetně měkkých stínů. Nicméně je možné metodu rozšířit o výpočet dalších nepřímých odrazů, zobrazení neostrých odlesků pro věrnější imitaci kovů nebo využít výsledky při zobrazení mlhy. Přímé osvětlení lze přidat standardními metodami, například pomocí kombinace *Phongova osvětlovacího modelu* se *stínovými mapami*. Celý výpočet se skládá ze tří částí:

1. **inicializace** mřížek světla (LPV) a zastínění (GV)
2. **iterativní propagace** světla v LPV s využitím GV
3. **vykreslení** scény s využitím LPV při stínování

4.1 Mřížky světla a zastínění

Celá metoda je postavena nad dvěmi trojrozměrnými mřížkami, které jsou umístěny přes celou nebo část scény:

- **Light Propagation Volume** (LPV) – obsahuje světelný tok scénou
- **Geometry Volume** (GV) – představuje hrubou aproximaci objektů scény

Nicméně pro každý ze dvou typů není nutné použít pouze jednu mřížku, ale je možné vytvořit kaskádu několika mřížek v různých velikostech. Tomuto rozšíření se věnuje sekce 5.1. Použití mřížek vychází z předpokladu, že nepřímé osvětlení je nízkofrekvenčního rázu, jinak řečeno lze očekávat pouze mírné změny světelné intenzity v prostoru scény. Což je vhodné pro buňky mřížek, které nepředstavují konkrétní bod, ale určitou oblast scény.

Buňky LPV pak obsahují sadu parametrů světla a jeho šíření scénou, ne v přímé formě, ale ve formě koeficientů *sférické harmonické funkce* (SH) pro každý RGB kanál barvy zvlášť.

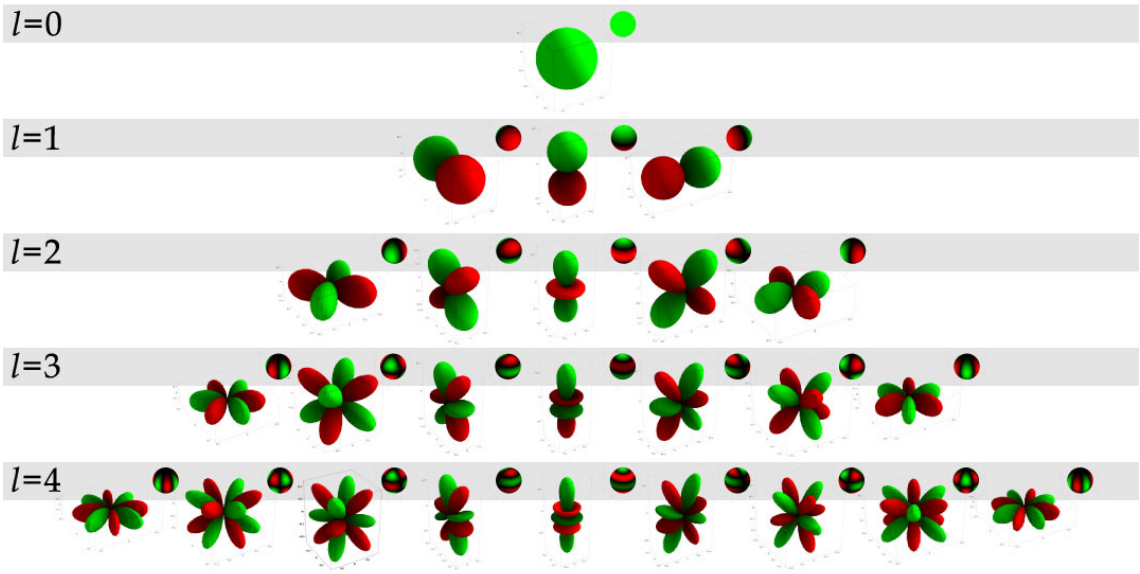
Podobně je tomu i v případě buněk mřížky *GV*, které lze nazývat *sufrely*, jež obsahují *SH koeficienty* pro vyjádření míry zastínění v daném směru – *blokující potenciál*.

Zbývá doplnit, že celá mřížka geometrie *GV* neleží na téměř stejné pozici jako *LPV*, ale je posunuta v každém směru o polovinu velikosti buňky tak, aby středy buněk byly umístěny v rozích buněk *LPV*.

4.2 Sférické harmonické funkce

Tyto funkce lze popsat jako množinu ortonormálních bázových funkcí, které tvoří polynom, pomocí kterých je možné aproximovat průběh funkce závislé na úhlu v prostoru jednotkové koule [20]. Výsledkem *SH* je vektor koeficientů, které určují míru zastoupení dané bázové funkce. Tyto koeficienty značíme: $y_{l,m}(\omega)$, $y_l^m(\omega)$ nebo $y_i(\omega)$, kde l je stupeň polynomu, m je řád polynomu a ω je úhel v polárních souřadnicích (θ, ϕ). Platí že: $l \geq 0$, $m \in < -l, l >$ a alternativní označení i vyjadřuje: $i = l(l+1) + m$. Počet *SH koeficientů* pak odpovídá l^2. Obrázek 4.1 zobrazuje vizualizaci *SH* pro prvních pět stupňů. *Sférické harmonické funkce* nachází uplatnění v mnoha oblastech fyziky a počítačové grafiky.

Mezi vlastnosti *SH* patří například uzavřenost vůči rotaci, operace projekce a reprojekce a podmnožina tzv. *zonálních sférických harmonických funkcí* (*ZH*). *ZH* jsou speciálním případem *SH*, které jsou rotačně symetrické okolo osy *Z*. Pro použití těchto vlastností v *LPV* je zaměřena další část.



Obrázek 4.1: Vizualizace prvních pěti stupňů polynomu *SH* v řádcích, sloupce odpovídají řádům funkce pro konkrétní stupně polynomu. Zelená barva ilustruje kladnou hodnotu funkce a červená zápornou. Pro lepší názornost je zobrazen průběh funkce také na normalizované kouli, kde intenzita barvy odpovídá vzdálenosti od středu soustavy. Převzato z [6].

4.3 Inicializace mřížek

Prvním krokem výpočtu je inicializace *LPV* a *GV* mřížek. Inicializace *LPV* probíhá v každém vykreslovaném snímku a principem je umístit do mřížky mnoho sekundárních bodových zdrojů světla – *Virtual Point Light* (*VPL*). Všechna *VPL* jsou umístěna na takové pozice ve scéně, kde dochází ke světelným odrazům přímého osvětlení na objektech scény. Pro tento proces je využita metoda *Reflective Shadow Maps*, která byla popsána v sekci 3.2. *RSM* pro každý světelný zdroj vytvoří trojici *textur*, tvořenou *texely* obsahující pozici, normálu a záři. *Texel* pak reprezentuje nové *VPL*, jehož intenzitu v daném směru je potřeba převést na sadu *SH koeficientů*, které lze vložit do konkrétní buňky *LPV*.

Pro eliminaci vzniku artefaktů osvětlení nebo zastínění sebe samého je pozice všech *VPL* posunuta o vzdálenost poloviny velikosti buňky *LPV* ve směru normály *VPL*. Na rozdíl od původní metody *RSM*, kde je aplikován prioritizovaný výběr *VPL* z textury, v tomto případě jsou do *LPV* jsou přidány všechny *VPL*. Nicméně pokud je *RSM* ve vyšším rozlišení je vhodné ji podvzorkovat.

Při injekci je nejprve potřeba vyjádřit směrovou distribuci světla $I_p(\vec{\omega})$ z pozice *VPL*, s normálou $\vec{n_p}$ a odraženou září Φ_p (dle opravy [13]):

$$I_p(\vec{\omega}) = \frac{\Phi_p}{\pi}\langle \vec{n_p}|\vec{\omega}\rangle_+ \tag{4.1}$$

SH koeficienty představují tok světla a pro jejich vyjádření jsou aplikovány polynomy stupně $l = 2$, což odpovídá čtyřem *SH koeficientům* $y_l^m(\omega)$ pro bázové funkce. V tom případě lze řešení nazvat jako *SH* nízkého řádu. Tento stupeň byl zvolen z důvodu možnosti efektivně uložit sadu koeficientů pro každou barvu do jedné proměnné typu `vec4`. Bázové funkce v polynomiální formě lze zapsat tvarem [20]:

$$y_0^0(\vec{\omega}) = \frac{1}{2\sqrt{\pi}} \qquad y_1^{-1}(\vec{\omega}) = -\frac{\sqrt{3}}{2\sqrt{\pi}}y \qquad y_1^0(\vec{\omega}) = \frac{\sqrt{3}}{2\sqrt{\pi}}z \qquad y_1^1(\vec{\omega}) = -\frac{\sqrt{3}}{2\sqrt{\pi}}x \tag{4.2}$$

Pro zjednodušení, koeficienty $(y_{0,0},\ y_{1,-1},\ y_{1,0},\ y_{1,1})$ mohou být také značeny čtveřicí (c_0, c_1, c_2, c_3). Dále je potřeba znát způsob získání *SH koeficientů* pro *sevřený kosínový lalok*, který je otočený ve směru normály *VPL*. Pro tento účel jsou vhodné *zonální harmonické funkce* (*ZH*), jejichž koeficienty lze získat pomocí rovnic [5]:

$$c_0 = \frac{\sqrt{\pi}}{2} \qquad c_1 = -\sqrt{\frac{\pi}{3}}y \qquad c_2 = \sqrt{\frac{\pi}{3}}z \qquad c_3 = -\sqrt{\frac{\pi}{3}}x \tag{4.3}$$

Výsledné koeficienty je nutné vynásobit s vyzařovaným světlem *VPL* čímž lze získat směrovou distribuci světla ve formě koeficientů, které je nakonec možné vložit do buňky *LPV*.

Dalším krokem je vytvoření mřížky zastínění *GV*. Postup je obdobný jako v případě inicializace *LPV*, kdy je vzorkována *RSM* textura a jednotlivé vzorky jsou vkládány do *GV* v podobě *SH*. Kromě textur z pozice světel je potřeba také vytvořit *RSM* z pozice kamery. Výhodné je pro tuto situaci stínovat scénu pomocí *odloženého stínování*, protože je možné použít výsledky z *G – bufferu*. Nicméně pro získání podrobnějších vzorků lze například použít techniku *depth peeling*. U statických scén není potřeba generovat zcela nové *GV* pro každý snímek, ale je možné využít *GV* z předchozích snímků a do něj přidávat další informace scény z nových *RSM*.

Jak již bylo zmíněno buňky $GV-surfely$ obsahují tzv. *blokující potenciál* ve smyslu schopnoti zablokovat světlo proudící buňkou daným směrem. Síla útlumu $B(\vec{\omega})$ může záviset na velikosti s celé mřížky GV, ploše *surfelu* A_s a úhlem mezi jeho normálou $\vec{n_s}$ a směrem světla $\vec{\omega}$. Formálně lze vztah zapsat jako:

$$B(\vec{\omega}) = A_s s^{-2} \langle \vec{n_s} | \vec{\omega} \rangle_+ \tag{4.4}$$

Samotné vložení do GV je velmi obdobné vložení do LPV. V tomto případě jsou do GV vkládány SH blokujícího potenciálu ve formě vektoru čtyř hodnot *SH koeficientů*. Aby však bylo zabráněno případu vložení vícero koeficientů do stejné buňky GV, pak vkládání poteciálu nejprve probíhá na lokální úrovni, kdy každá RSM obsahuje vlastní GV jejichž výsledky jsou nakonec sloučeny do výsledné GV metodou výběru největších koeficientů SH.

4.4 Propagace světla mřížkou

Po inicializaci mřížek následuje část propagace instenzity světla, kdy je iterativně pro každou buňku LPV distribuováno světlo do okolních šesti sousedních buněk (obr. 4.2 vlevo). Přesněji řečeno je potřeba určit přenos světla pro každou stěnu přilehlých buněk (obr. 4.2 uprostřed). Pro jednu stěnu buňky f lze přenos Φ_f určit pomocí:

$$\Phi_f = \int_\Omega I(\vec{\omega_c}) V(\vec{\omega}) \mathrm{d}\vec{\omega} \tag{4.5}$$

Kde $I(\vec{\omega})$ je přibližná intenzita světla ze zdrojové buňky, dána vzorcem:

$$I(\vec{\omega}) \approx \sum_i c_l^m y_l^m(\vec{\omega}) \tag{4.6}$$

Pro připonemutí: i je souhrnné označení pro: $i = l(l+1)+m$. $V(\vec{\omega})$ reprezentuje operátor viditelnosti stěny f, který je roven jedné, pokud paprsek vržený se středu buňky ve směru $\vec{\omega}$ protíná stěnu f. V opačném případě je roven nule. Funkci viditelnosti lze také získat projekcí do SH:

$$V(\vec{\omega}) \approx \sum_i v_l^m y_l^m(\vec{\omega}) \tag{4.7}$$

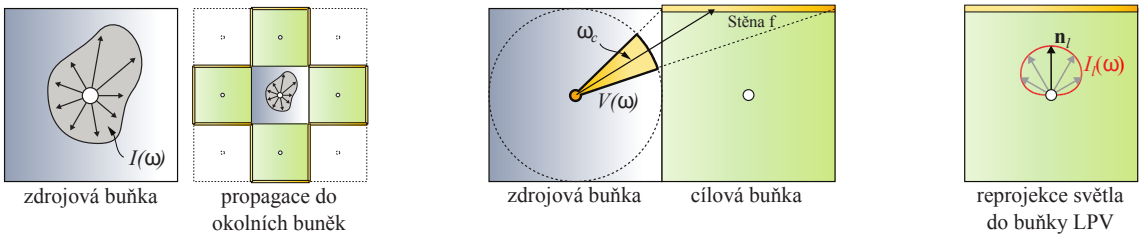
Ovšem výpočet je možné výrazně zjednodušit pomocí skalárního součinu koeficientů SH představující světelnou intenzitu c_l^m s koeficienty viditelnosti v_l^m.

Nicméně kvůli nízké přesnosti výsledků výpočtu funkce viditelnosti $V(\vec{\omega})$ při použití SH nízkého řádu, je místo toho určena hodnota *pevného úhlu* $\Delta_{\vec{\omega}f}$ pro každou stěnu f cílové buňky. Čímž lze získat hlavní vektor $\vec{\omega}$ určující směr kuželu viditelnosti. Pro *pevný úhel* platí:

$$\Delta_{\vec{\omega}f} = \int_\Omega V(\vec{\omega}) \mathrm{d}\vec{\omega} \tag{4.8}$$

Dle poznámek autora [13] lze rozlišit dva *pevné úhly*: pro stěnu otočenou přímo a pro stěnu natočenou bokem ke zdrojové buňce. V prvním případě se jedná o úhel 0.4_{rad} a ve druhém případě o hodnotu 0.4234_{rad}.

Světelná intenzita $\Phi_{\vec{\omega}f}$ dopadající ve směru $\vec{\omega}$ na stěnu f potom odpovídá průměrné intenzitě v oblasti *pevného úhlu*:

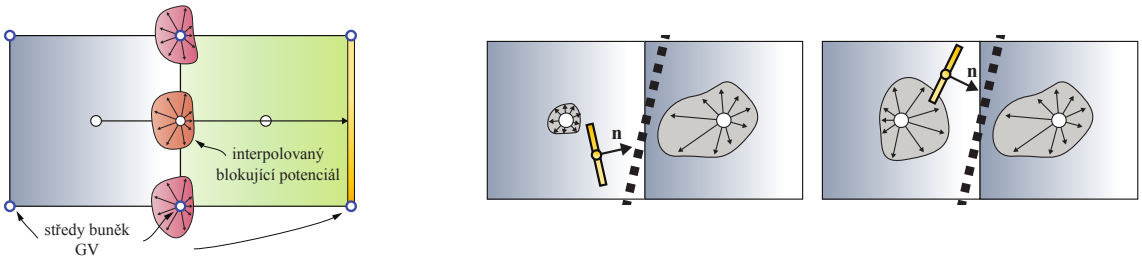


Obrázek 4.2: Postup výpočtu při propagavi světla v *LPV*. Vlevo: zdrojová buňka (modrá) s *SH koeficienty* reprezentující distribuci světla. Kolem ní sousedí buňky (zelené), do jejichž stěn (žluté) bude světlo propagováno. Uprostřed: propagace intenzity do stěny sousední buňky. Vpravo: reprojekce nového světla v sousední buňce vyjádřené pomocí *sevřeného kosínového laloku*. Převzato z [10].

$$\Phi_{\vec{\omega}f} = \frac{\Delta_{\vec{\omega}f}}{4\pi} I(\vec{\omega}_c) \tag{4.9}$$

Následně je pro každou stěnu vytvořen nový světelný zdroj, jehož intenzita Φ_l odpovídá intenzitě Φ_f stěny f (obr. 4.2 vpravo). Určit *SH koeficienty* tohoto *VPL* lze obdobně jako v případě inicializace *LPV*, a to vynásobením *sevřeného kosínového laloku* otočeného ve směru *VPL* s pevným úhlem $\Delta_{\omega f}$ a intenzitou Φ_l. Tu lze získat pomocí skalárního součinu *SH koeficientů* normály ve směru na střed cílové stěny (rovnice 4.2) s *SH koeficienty* zdrojové buňky. Výsledná intenzita cílové buňky je dána součtem všech *SH koeficientů* dílčích reprojekcí.

Kromě propagace světelné intenzity je také potřeba vyhodnocovat mřížku zastínění (*GV*). Pro každý výpočet distribuce intenzity buňky v *LPV* jsou interpolovány (díky posunutí *GV*) *SH koeficienty* v *GV*, které představují *blokující potenciál* (obr. 4.3 vlevo) na pozici průniku hlavního vektoru mířícího na sousední stěnu se sdílenou stěnou mezi buňkami. Získané koeficienty pak určují míru útlumu pro danou intenzitu v *LPV*. Po použití útlumu lze získat opět sadu čtyř *SH koeficientů* buňky pro každou barvu, připravenou pro další iteraci propagace světla.



Obrázek 4.3: Vlevo: interpolace rohových *SH koeficientů* mřížky zastínění *GV* pro určení výsledného útlumu při propagaci světla do sousední buňky *LPV* na pozici průniku vektoru se sdílenou stěnou. Vpravo: ukázka z procesu vykreslení scény: rozdílné intenzity sousedících buněk v *LPV* způsobené zastiňujícím objektem (přerušovaná čára). Pro správný výpočet stínování vzorku (žlutá plocha) s normálou `n` je potřeba určit změnu ve směru normály. Převzato z [10].

4.5 Použití výsledků pro stínování scény

Po dokončení určitého počtu iterací propagace světla, je připraveno LPV s nepřímým osvětlením scény. Při vykreslování geometrie objektů, je pak pro každý vykreslovaný vzorek zjištěna jeho pozice ve scéně a normála. Pro získání osvětlení z LPV je nejprve nutné promítnout obrácenou normálu vzorku do SH a následně provést skalární součin s interpolovanými koeficienty z LPV na pozici vzorku [5].

Při vykreslování však mohou nastat případy tzv. *světelného průsaku*, kdy se objeví světlé plochy objektů na místech, kde by správně měly být plochy tmavé. Tento problém je obvykle způsoben velkým rozměrem buněk LPV a tenkou zástěnou mezi buňkami. Autoři [10] přišli s řešením pomocí tzv. *faktoru útlumu*, který využívá směrovou změnu distribuce světelné intenzity. Pro každý vzorek s pozicí $\mathtt{x}$ a normálou $\vec{n}$ je potřeba určit interpolované *SH koeficienty* c_l^m a změnu směru ve směru normály vzorku $\nabla_{\vec{n}} c_l^m$. Pokud je změna mezi c_l^m a $\nabla_{\vec{n}} c_l^m$ větší, potom by měly být koeficienty c_l^m utlumeny (obr. 4.3 vpravo).

Kapitola 5

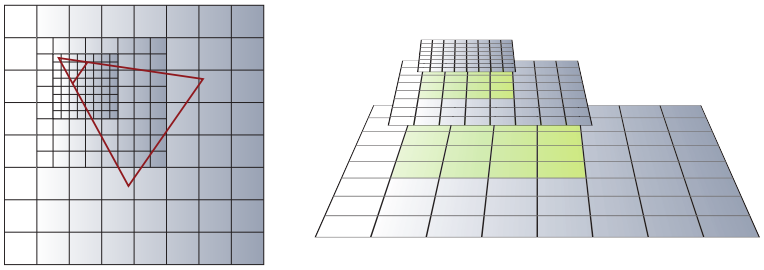
Rozšíření pro Light Propagation Volumes

V této kapitole jsou popsány různé rozšíření metody *Light Propagation Volumes*, díky kterým lze dosáhnout lepších grafických výsledků, při zachování nebo mírném zvýšení časové náročnosti metody na výpočet.

5.1 Cascaded Light Propagation Volumes

Klasické řešení *LPV* přináší několik omezení a problémů. Jedním z nich je nutnost pojmout celou nebo část scény do jedné uniformní mřížky. Což je nevhodné v případě rozsáhlých scén, protože při projektivním zobrazení je potřeba vyhodnocovat mnoho malých buněk vzdálených od kamery, které nakládají se zbytečně velkým počtem detailů. Na rozdíl u blízkých ploch, kde je potřeba zobrazit nejvíce detailů, tyto detaily chybí.

Řešením může být právě metoda *Cascaded Light Propagation Volumes* [10], která využívá hierarchickou strukturu několika *LPV* a *GV* s rozdílnými rozměry buněk (obr. 5.1). Díky tomu je možné při vykreslování blízkých ploch využít podrobnější mřížku s menšími buňkami a naopak pro vzdálené plochy použít mřížku s buňkami mnohonásobně většími. Další výhodou je možnost použít mřížku pouze pro část scény a ne scénu celou. Z toho ovšem vyplývá nutnost udržovat mřížku v blízkosti kamery a při jejím pohybu pohybovat také s kaskádou *LPV* a *GV* mřížek.



Obrázek 5.1: Ukázka kaskády *LPV*: pro vykreslované oblasti s menší vzdáleností od kamery jsou vzorkována data z úrovní s menším rozměrem buněk a obráceně. Převzato z [10].

Nicméně při navýšení počtu mřížek je potřeba také drobně modifikovat algoritmy z minulé kapitoly. Při procesu vkládání *VPL* jsou *SH koeficienty* světelné intenzity resp.

blokujícího potenciálu vkládány do všech úrovní *LPV* resp. *GV* zároveň. V další části propagace jsou iterace prováděny pro všechny úrovně nezávisle, avšak intenzity z úrovní s menšími buňkami mohou být vkládány do úrovní s buňkami většími. V poslední části při vykreslování jsou pro daný vzorek intenzity osvětlení interpolovány mezi nejbližšími dvěmi úrovněmi *LPV*.

5.2 Usměrnění propagace

Další z problémů této metody je grafický artefakt způsobující zpětné nasvícení podložky, která je zdrojem *VPL*, při větším počtu propagací. Tento problém způsobuje využití *sférických harmonických funkcí* o nízkém řádu, které nedokáží postihnout velký počet detailů funkce distribuce světelné intenzity do okolí. K vyřešení problému se naskýtá několik možností: například využít *SH* o vyšším řádu, nicméně takové řešení by řádově zpomalilo výpočet metody *LPV* z důvodu potřeby pracovat s vysokým počtem *SH koeficientů*.

Proto bylo navrženo řešení postavené na principu sekundární reprojekce *SH koeficientů* v původních směrech šíření *VPL*. Toho je docíleno pomocí vložení normál z *VPL* při inicializaci do další mřížky a dilatací normál do okolí před každou iterací propagace *SH koeficientů*. Při výpočtu reprojekce *SH koeficientů* na stěny okolních buněk je provedena druhá reprojekce pomocí kladného skalárního součinu s *SH koeficienty* ve směru dilatované normály získanými pomocí vzorce 4.2. Výsledek je opět vynásoben s *SH koeficienty sevřeného kosínového laloku* (vzorec 4.3) ve směru této normály.

5.3 Injekce světla oblohy

Dalším problémem metody je omezení výpočtu pouze na *VPL* získaná z přímého osvětlení scény, přičemž metoda nedokáže vypočítat nasvícení scény z osvětlení získaného z pozadí scény. Pro řešení bylo navrženo jednoduché rozšíření, které vytvoří *RSM* při pohledu shora na scénu. Nyní každý vzorek *RSM* nepředstavuje nové *VPL* vkládané do *LPV*, ale představuje oblast kolize ve scéně, podobně jako *blokující potenciál*. *VPL* jsou vkládány do *LPV* ve sloupcích shora dolů, dokud nenarazí na kolizi v podobě vzorku z *RSM*. Vložení je provedeno do stejné *LPV* mřížky jakou využívá původní metoda.

5.4 Rozptyl průchodem povrchu

V minulých sekcích metoda *LPV* předpokládala pouze neprostupný povrch světlem. Nicméně tuto techniku lze rozšířit o výpočet propagace světla skrz průsvitné objekty – *Subsurface Scattering LPV*. Z hlediska prostředků se jedná o duplikaci původní metody *CLPV* a z hlediska výpočtu se rozšíření liší v následujících bodech:

1. do *RSM* jsou vykresleny pouze objekty, které jsou průsvitné
2. *VPL* do *SSLPV* jsou vloženy ve směru obrázených normál a *VPL* není třeba posouvat o násobky velikosti buněk *LPV*
3. propagace probíhá téměř identicky s původní metodou *LPV* s rozdílem vynásobení *SH koeficientů* konstantou útlumu materiálu

Při stínování průsvitných materiálů může být pro rychlost výpočtu vyčtena hodnota z *SSLPV* pouze na povrchu objektu, nicméně pro zpřesnění je vhodné provést ray – marching ve směru vektoru pohledu skrz objekt.

5.5 Matné odrazy

Poslední rozšíření navržené autory [10] je simulace odlesků povrchu pomocí *LPV*. Jedná se o rozšíření poslední části metody – vykreslení, kdy je proveden ray – marching v mřížce *LPV* ve směru vektoru odrazu od vzorku povrchu ve scéně. Při průchodu paprsku mřížkou *LPV* je možné buď číst hodnotu z jedné úrovně kaskády *LPV* nebo dle délky vektoru číst z několika úrovní kaskády s nutností interpolovat hodnoty mezi úrovněmi.

Kapitola 6

Návrh aplikace

Tato kapitola popisuje návrh aplikace a obsažených komponent nutných pro řížení a vykreslování scény, zpracování modelů, stínů a shaderů. Dále je popsán návrh součástí zajišťující výpočet stínování pomocí metody *Light Propagation Volumes* a navržených rozšíření.

6.1 Zobrazovací engine

Zobrazovací engine představuje komplexní program s řadou funkcí pro načítání obrázků, modelů, shaderů a v neposlední řadě také vykreslování trojrozměrné scény. Herní enginy jsou komplexní knihovny, které obsahují podporu například pro fyziku, herní logiku, efekty a další prvky.

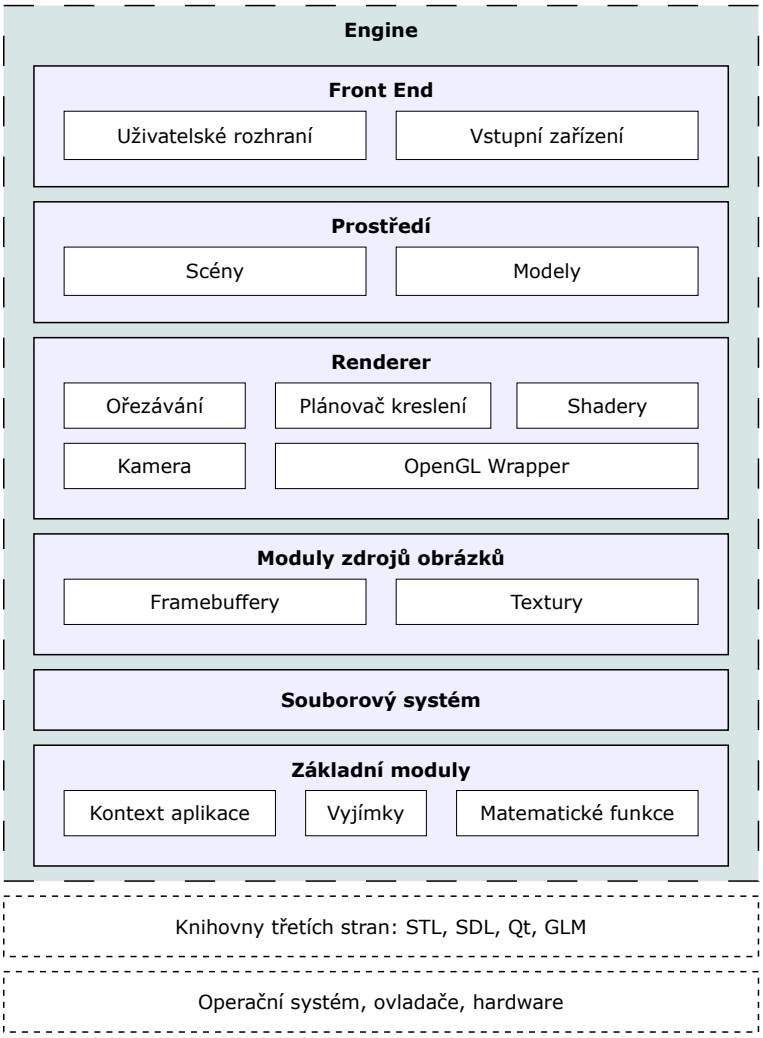
Nicméně při implentaci této aplikace byl využit vlastní jednoduchý engine, který v základu podporoval kromě načítání a zobrazování modelů také funkce pro ovládání kamery, použití *framebufferů*, kompilaci *shaderů*, správu souborů a další. Logické uspořádání modulů zachycuje obrázek 6.1.

Na nejvyšší vrstvě enginu se nachází tzv. *front end* obsahující třídy pro uživatelské rozhraní, widget pro kreslení grafického výstupu, zpracování argumentů příkazové řádky a vstupu uživatele. Z této úrovně je spouštěna vykreslovací smyčka, při které jsou volány instance objektů z nižších úrovní. Do nižší úrovně patří prvky reprezentující scény, které uschovávají různé objekty, například světla a modely. Při zobrazování scény je naplánováno vykreslení každé viditelné geometrie modelu, která může využívat různé materiály a osvětlení. Tyto úkony zajišťují moduly ze sekce *Renderer*.

Všechny instance tříd enginu propojuje objekt *kontextu*, který disponuje odkazy na všechny důležité moduly. Použitím takového řešení může každý objekt z dané vrstvy získat libovolnou instanci z vrstvy nižší. Například prvek scény může získat odkaz na libovolný model, shader, texturu, apod.

Souborový systém v enginu umožňuje načítat libovolné soubory z mnoha definovatelných cílů (*cest*). Pro každý cíl jsou důležité tyto adresáře:

- *data/maps* – textury modelů scény
- *data/models* – modely scény ve formátu `4ds`
- *data/shaders* – veškeré shadery enginu



Obrázek 6.1: Uspořádání hlavních modulů enginu.

6.2 Reprezentace scény

Engine byl navržen tak, aby bylo možné v jeden okamžik (např. při startu programu) načíst a připravit několik různých scén, přičemž zobrazována může být pouze jedna z nich. Z časových důvodů nejsou scény uloženy v samostatných souborech, ale jejich vzhled je určen přímo v kódu *front endu*. Objekty scény mohou nabývat dvou typů: *světla* a *modelu*. U světla lze definovat translaci, zář a dosah. U modelu translaci, konkrétní soubor modelu a určení, zda je model umístěn ve scéně nebo v pozadí. Taková možnost je vhodná při kreslení oblohy a slunce.

Při vytvoření nebo pohybu objektu modelu je určeno několik doplňujících parametrů, mezi které patří aktuální osvětlení ovlivňující tento model a obalová tělesa geometrie využívaná před jejím vykreslováním. V případě změny parametrů světla jsou aktualizovány všechny objekty modelů přijímající zář změněného světla.

6.3 Reprezentace modelů

Modely scény jsou uloženy ve formátu `4ds`, který využívaly hry od tehdejší brněnské společnosti *Illusion Softworks* (nyní *2K Czech*). Verzí tohoto formátu je více, nicméně engine podporuje pouze modely ve formátu kompatibilním se hrou *Mafia* na PC. Formát specifikuje[1] v souboru dvě hlavní části: seznamy materiálů a *meshů*.

Materiály obsahují například barvu, průhlednost a textury pro *difuzi*, *průhlednost* a *odlesk*. Pro zaručení kompatibility formátu s původní specifikací vs. přidáním podpory dalších parametrů jsou například *spekulární* a *normálové* textury automaticky vyhledávány v adresáři s texturami. Zde je zachována jmenná konvence souborů, které jsou využívány herním enginem *CryEngine*. Pro každý typ musí názvy souborů končit řetězcem (před příponou) `_spec` pro *spekulární* texturu resp. `_ddn` pro *normálovou* texturu.

Mesh popisuje objekt v modelu, který je složen z množiny hraničních reprezentací geometrie. Formát definuje mnoho typů geometrie, například pro podporu *skeletální* nebo *morfované* animace, avšak pro tuto aplikaci je využit pouze typ *standardního meshe*, který představuje statickou geometrii. Tento typ *meshe* se skládá z několika *úrovní detailu*, které obsahují konkrétní geometrii, rozdělenou na několik částí, kde každá část využívá jiný druh materiálu.

6.4 Inicializace enginu a zobrazovací smyčka

Po spuštění programu jsou vytvořeny a propojeny všechny moduly enginu pomocí objektu kontextu. Následně je vytvořeno gui, jsou nastaveny výchozí cesty pro systém souborů a také jsou načteny a zkompilovány shadery. Poté jsou vytvořeny tři scény a do nich jsou vloženy testovací modely *Sponzy*, *katedrály sv. Jakuba v Šibeniku* a *Klarkova motelu* ze hry *Mafia*. A nakonec jsou do každé scény vloženy světla pro slunce a ambientní osvětlení, několik menších testovacích modelů a navigační body pro pohyblivou kameru.

Při běhu aplikace není zvolená scéna vykreslována neustále, ale je vykreslována pouze v případě reakce na vstup uživatele, například při otočení a pohybu kamery nebo při změně parametrů vykreslování. Opakované vykreslování při pohybu kamery zajišťuje běžící časovač a správnou rychlost pohybu kamery určuje hodnota *simulačního kroku* vyjádřená jako rozdíl systémového času mezi snímky. Avšak pokud by byl tento krok počítán pouze v případě překreslení scény, mohlo by docházet k velkým skokům pohybu kamery způsobené většími časovými intervaly mezi snímky. Z toho důvodu je simulační krok počítán každých $16ms$ i v případě statické kamery.

Celkové vykreslení scény s dodatečnými efekty probíhá v následujících průchodech:

1. vykreslení pozadí scény
2. vykreslení stínů a RSM
3. výpočet metody *Light Propagation Volumes*
4. vykreslení scény pro výpočet slunečních paprsků
5. standardní vykreslení s využitím výsledů metody *LPV*
6. vykreslení slunečních paprsků

[1] Kompletní specifikaci formátu `4ds` lze nalézt na adrese autora této práce: http://www.djbozkosz.wz.cz/index.php?id=17

Průchody, při kterých je zobrazována scéna z pohledu kamery jsou složeny se dvou kroků:

1. průchod scénou a výběr viditelných geometrií z objektů
2. postupné vykreslení vybraných geometrií

Vzhledem v malému počtu objektů ve scéně je první krok procesu proveden sekvenčně a nebyly tedy implementovány hierarchické struktury dělení prostoru, například *octree*, *quadtree*, apod. Pro výběr viditelných objektů scény je využita metoda *frustum culling*, pomocí které lze určit, zda obalové těleso daného objektu pronikne do komolého jehlanu pohledu kamery. V případě této aplikace obalová tělesa představují *Axis Aligned Bounding Box*, což jsou kvádry zarovnané s osami scény. Pokud se daný objekt scény protíná s frustem, poté jsou testovány na průnik všechny *meshe* vyšetřovaného modelu. Nicméně kontrétní *mesh* se nekreslí přímo, ale jeho vykreslení je naplánováno v modulu *rendereru*. Díky tomu je možné kreslení seřadit, docílit tak menší procesorový „overhead“ ke grafické kartě a urychlit vykreslování. V druhém kroku, při vykreslování jsou postupně v *rendereru* přepínány *shadery* pro skupiny *meshů*. Pro každý *mesh* je nastavena množina *uniformních proměnných* patřící ke zvolenému *shaderu* a geometrie je vykreslena. Celý proces vykreslení shrnuje algoritmus 6.1.

Algoritmus 6.1: Vykresli scénu z pohledu kamery.

```
1:  for každý objekt aktivní scény do
2:      if objekt je typu model a je viditelný v pohledovém frustu then
3:          for každý mesh v modelu do
4:              if mesh je standardní typ a je viditelný v pohledovém frustu then
5:                  for každou část geometrie do
6:                      urči transformaci geometrie na obrazovku;
7:                      naplánuj vykreslení geometrie vložením do pole rendereru;
8:                  end for
9:              end if
10:         end for
11:     end if
12: end for
13: for každý shader do
14:     zvol shader jako aktuální;
15:     for každá naplánovaná geometrie do
16:         nastav parametry shaderu;
17:         vykresli geometrii;
18:     end for
19: end for
20: vyčisti pole s naplánovanými geometriemi;
```

6.5 Shadery

Shadery jsou programy běžící na grafické kartě a ovlivňují části grafického řetězce. Tyto *shadery* jsou sdružovány do *shader programů* pro naprogramování několika bloků *GPU* ve stejný okamžik. Cílem *shaderů* může být například výpočet stínování geometrie scény nebo obecný výpočet na *GPU*.

Každý *shader* a *shader program* spadá do modulu *shaders*. Tento modul se stará o načtení dílčích *shaderů* ze souboru, jejich přeložení, *slinkování* do *shader programu* a následné nazelení *atributů* a *uniformních proměnných*. Jelikož je třída *shader programu* definována obecně pro všechny druhy *shaderů* dohromady, daný *shader program* je konkretizován pomocí řady přepínačů určující například využité *atributy*, *uniformní proměnné*, *texturovací jednotky* nebo nastavení *vykreslovacího řetězce*. Při psaní *shaderů* se velmi často stává, že jednotlivé soubory *shaderů* využívají identické části kódu. Proto byly všechny *shadery*, řešící podobný problém, slouženy do jednoho souboru, přičemž jejich rozdílné úseky kódu byly obaleny do *podmíněných bloků* a pojmenovány pomocí příkazů *preprocesoru*. Libovolné bloky pak bylo možné při překladu vybrat pomocí přiloženého seznamu jmen bloků.

Během vykreslování geometrie scény jsou před každým vykreslením nejprve povoleny *atributy* pro zpracování vrcholů geometrie, následovány nastavením zvolených *uniformních proměnných* a *texturovacích jednotek*. Po vykreslení geometrie jsou povolené *atributy* vypnuty.

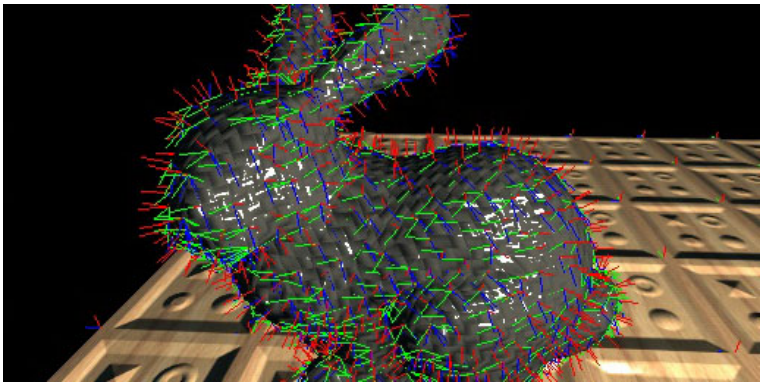
6.6 Stínování

V úvodní části bylo zmíněno, že se metoda *Light Propagation Volumes* při výpočtu stínování používá jako doplnění ke standardním jednoduchým metodám stínování. Mezi takové metoly lze určitě řadit lokální osvětlovací modely. Pro demonstrační aplikaci byl využit *Blinn – Phongův osvětlovací model* [1], který vychází z *Phongova osvětlovacího modelu*, avšak na rozdíl od něj definuje spekulární odraz vzorcem:

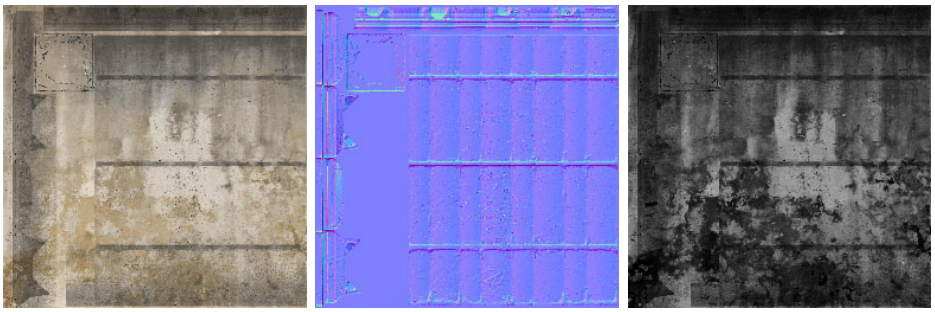
$$L_S = \left\langle \frac{\vec{l} + \vec{v}}{\left|\vec{l} + \vec{v}\right|}, \vec{n} \right\rangle_+ \tag{6.1}$$

Pro zvýšení úrovně vizuální kvality zobrazení byla ke stínování přidána technika *Normal Mapping*, která využívá další texturu pro uložení normál (obr. 6.3 uprostřed). Výsledkem je podrobnější informace o nerovnostech povrchu. Kromě *normál* vrcholů geometrie je třeba znát také *tangenty* a *bitangenty*. Všechny tři vektory jsou v prostoru mezi sebou kolmé a orientované podle změny texturovacích souřadnich v okolí bodu (obr. 6.2). Pro správné určení normály vzorku je nejprve nutné vytvořit pro vrcholy geometrie tzv. *TBN* matici sestavenou z *normály*, *tangenty* a *bitangenty* vynásobené *normálovou inverzní transponovanou* maticí. Poté je možné buď osvětlení počítat v *tangent space*, kdy jsou vektory světla a pohledu vynásobeny touto maticí nebo je možné *TBN* matici invertovat a osvětlení počítat v tzv. *world space*. Normála je poté určena jejím vyčtením z *normálové textury*, posunutím do prostoru $< -1, 1 >$ a vynásobením interpolovanou *TBN* maticí mezi vrcholy primitiva. I když je druhý přístup výpočetně náročnější, lze později normálový vektor jednoduše využít v poslední části metody *Light Propagation Volumes*.

Pro omezení odlesku je využita *spekulární mapa* (obr. 6.3 vpravo), pomocí které je vynásobena intenzita *spekulární odrazu*.



Obrázek 6.2: Ukázka zobrazení *normál* (červené), *tangent* (zelené) a *bitangent* (modré) na modelu králíka.



Obrázek 6.3: Ukázka textur pro *difuzi*, *normály*, a omezení *spekulárního sodlesku*.

6.7 Stíny

Problémem lokálních osvětlovacích modelů v real – time počítačové grafice je neschopnost řešit zobrazení stínů. Tento nedostatek je odstraněn použitím metody *stínových map*, která je založena na principu vykreslení scény z pozice světla do textury. Pro získání textury stínu není potřeba vykreslovat scénu se všemi efekty a texturami, ale je nutná pouze informace o hloubce. Během kreslení scény do textury je potřeba uchovat transformaci kamery vůči geometrii scény (*mvp* matice). Tyto matice jsou později vynásobeny tzv. *bias* maticí pro transformaci z *NDC* prostoru do rozsahu *sampleru* textury $< 0, 1 >$.

Po vytvoření hloubkové textury následuje standardní vykreslení scény, při které je použita získaná textura hloubky a daná „stínová“ *mvp* matice. Každý bod je kromě své *mvp* matice vynásoben také druhou *mvp* maticí pro jeho transformaci do oblasti stínové mapy:

$$shadowPos = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0.5 & 0.5 & 0.5 & 1 \end{bmatrix} \cdot mvp \cdot vertexPos \tag{6.2}$$

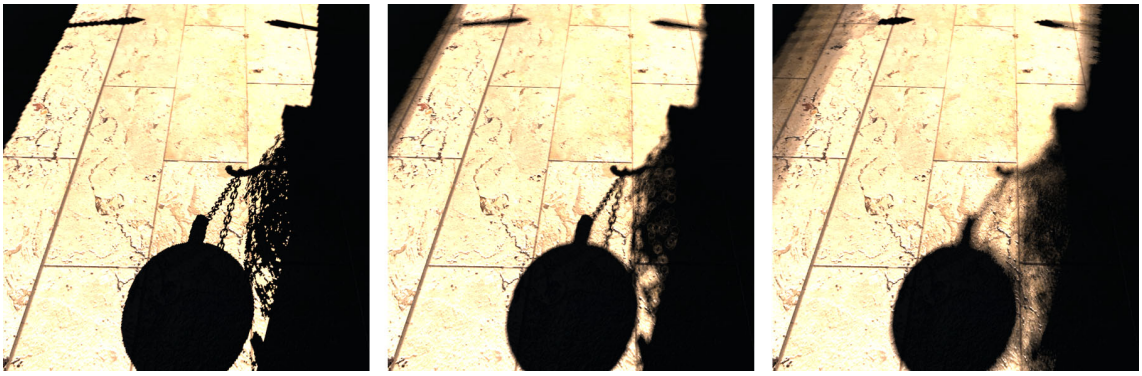
Při rasterizaci pak lze získat dle pozice vzorku danou hloubku z textury a tu porovnat s hloubkou aktuálního vzorku. Pokud je hloubka z textury menší, poté je vzorek scény zastíněn.

V případě tvorby stínu pro směrové světlo, které má neuřitou pozici, je použita paralelní projekce pro *„kameru stínu“* a oblast scény je vykreslena s ohledem na pozici a pohled hlavní kamery. Při pohybu kamery lze ovšem spatřit problikávání a pohyb stínů na úrovni vzorků. V takovém případě je vhodné *kameru stínu* zarovnat na násobky velikosti vzorků textury včetně respektování rotace *kamery stínu* kolem hlavní kamery. Postup nastavení kamery může být následující:

1. zpětná rotace kamery pro zarovnání se souřadnou osou
2. zaokrouhlení pozice na nejbližší násobek velikosti vzorku textury
3. dopředná rotace o stejný úhel

Popsanou základní metodu stínových map lze rozšířit o aproximaci proměnlivého rozmazání stínu závislého na vzdálenosti zdroje stínu a místa dopadu stínu (obr. 6.4). Princip metody vychází ze zdrojů [11] a [21] a výpočet se skládá ze dvou kroků:

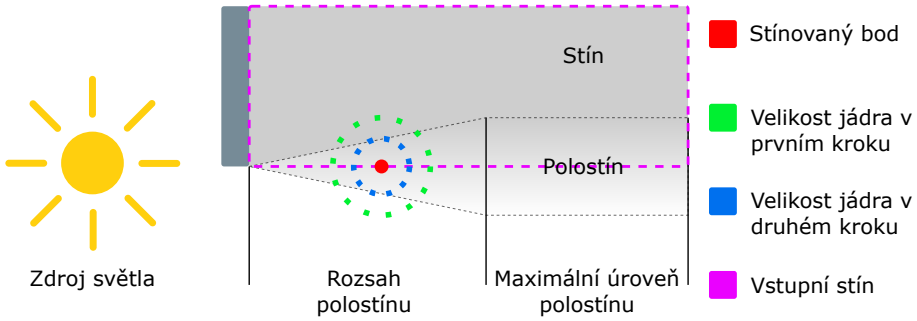
1. odhad průměrné délky stínu
2. výpočet intenzity zastínění



Obrázek 6.4: Ukázka různé síly rozmazání stínů odlišně vzdálených od stínících předmětů.

V obou krocích se využívá druhové konvoluční jádro, které na rozdíl od metody [21], z důvodu urychlení výpočtu, využívá pouze několik bodů po obvodu kruhu namísto většího počtu bodů seřazených od okraje do středu. Při určení zastínění daného vzorku scény je v prvním kroku výpočtu, s využitím kruhového jádra, zjištěna průměrná hloubka okolí, přičemž velikost jádra odpovídá maximálnímu průměru polostínu. Po zjištění hloubky je vykonán druhý krok, při kterém je zvoleno měřítko jádra takové, aby respektovalo vzdálenost mezi odhadnutou hloubkou a hloubkou zdroje stínu vůči shora omezené velikost jádra. Poté je jádro využito ke ztištění (ne)zastítění okolních bodů počatečního vzorku (obr. 6.5). Zprůměrováním získaných hodnot lze částečně aproximovat efekt polostínů.

Technika *stínových map* není omezena pouze na jednu hloubkovou texturu, nicméně při zobrazování prostorově rozsáhlých scén je téměř nutné využít kaskádu několika textur, do kterých je kreslena různě rozsáhlá část scény. Během zobrazování vzdálených míst scény jsou použity právě textury hloubky obsahující vzdálené oblasti, často také se sníženou úrovní detailu.



Obrázek 6.5: Ilustrace principu výpočtu polostínů.

6.8 Násobné vykreslení scény

Využití kaskád textur u stínů a *RSM* při injekci u metody *Light Propagation Volumes* je žádoucí, nicméně jejich tvorba v základním provedení vyžaduje další zobrazovací průchody scénou, což se velmi negativně projevuje na výkonu. Proto bylo navrženo řešení násobného kreslení geometrie do textury pomocí tzv. *instancingu*. Jedná je funkčnost grafických *API*, díky kterému lze jedním voláním kreslení vytvořit několik kopií stejné geometrie bez účasti procesoru. Výstupní textura je virtuálně rozdělena do několika stejně velkých dlaždic, kde každá dlaždice představuje jednu texturu kaskády. Při násobném vykreslování je daná *instance* identifikována a její vrcholy jsou transformovány do oblasti odpovídající dlaždice:

$$vertexTilePos = \begin{bmatrix} tileWidth & 0 & 0 & 0 \\ 0 & tileHeight & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tileX & tileY & 0 & 1 \end{bmatrix} \cdot mvp \cdot vertexPos \tag{6.3}$$

Po transformaci je geometrie ořezána určením hodnoty vzdálenosti vrcholu od roviny tvořící hranici dlaždice pomocí skalárního součinu rovnice roviny s vrcholem.

6.9 Light Propagation Volumes

Řešení metody *Cascaded Light Propagation Volumes*, která byla popsána v kapitolách 4 a 5, bylo v aplikaci navrženo s ohledem na kompatibilitu starších grafických karet a komplexnost dnešních grafických karet. Proto byl výpočet metody navržen ve dvou variantách, využívající:

- zavedený *grafický řetězec* – GS varianta
- *compute shadery* – CS varianta

První varianta (GS) využívá dvojice *framebufferů* s 3D texturami, které jsou přepínány mezi propagacemi. Druhá varianta (CS) využívá *image jednotky* pro náhodné čtení a zápis dat do 3D textury. Textury vždy jsou zvoleny tak, aby byly k dispozici dvě přepínané textury pro čtení a uložení výsledků propagace, akumulační textura propagace a textura pro blokující potenciál. Vzhledem k tomu, že kaskáda mřížek textur LPV a GV byla navřena tak, aby se pohybovala s kamerou, je nutné, z důvodu odstranění blikání LPV při pohybu, pozice každé úrovně mřížky zarovnat nezávisle na sobě na násobky velikosti buněk.

6.9.1 Inicializace

Již bylo zmíněno, že výpočet metody je rozdělen do tří částí: inicializaci, propagaci a vykreslení. V první části inicializaci jsou všechny textury vymazány a poté je vytvořena *Reflective Shadow Mapa* obsahující pozice, normály a difuzní odrazy scény z pozice hlavního světla a v případě využití mřížky zastínění, také z pozice kamery. Jelikož *LPV* využívá kaskádu mřížek, jsou úrovně kaskády *RSM* vytvořeny pomocí stejného systému dlaždic popsaného v minulé sekci.

Po získání *RSM* textury je pro každý její vzorek známá pozice, pomocí které je určena cílová buňka pro jednu ze dvou přepínaných textur *LPV* a akumulační textury *LPV*. K určení intenzity *VPL* je použit vzorec 4.1 s využitím normály a difuzního odrazu vzorku. Následně jsou pomocí vzorce 4.3 určeny *SH koeficienty sevřeného kosínového laloku* ve směru normály vzorku a každá barevná složka intenzity je vynásobena těmito koeficienty. Proces inicializace je v případě *GS* varianty proveden pomocí kreslení bodů (*point renderingu*) a v případě *CS* varianty je proveden pomocí *atomických operací*. Při injekci je důležité kažké *VPL* posunout o určitou vzdálenost ve směru normály vzorku z důvodu osvětlení geometrie sebe samé. Posun o půlku velikosti buňky *LPV* se v rámci testování stal jako nedostatečný, a proto jsou všechny *VPL* posunuty o celou délku buňky.

V případě vložení blokujícího potenciálu do *GV* se postupuje obdobně: vkládány jsou blokující poteciály ze vzorků *RSM* tvořené z pozic okolních světel a z pozice kamery. Síla zablokování odpovídá vzorci 4.4, přičemž zde není žádoucí posouvat nový poteciál ve směru normály.

Z důvodu urychlení inicializace textur mřížek každá úroveň kaskády *RSM* vkládá hodnoty pouze do jedné odpovídající úrovně kaskády *LPV* nebo *GV*.

6.9.2 Propagace

Dalším krokem výpočtu je propagace, která může probíhat v několika iteracích. Postup výpočtu jedné iterace je také navržen ve dvou variantách: *GS* a *CS*. Na počátku každé iterace je vymazána jedna ze dvou *LPV* textur, která bude určena pro zápis nových *SH koeficientů*. Druhá *LPV* textura slouží ke čtení *SH koeficientů* z předchozí iterace. Zároveň je výsledek také zapisován do akumulační *LPV* textury.

Vzhledem k respektování původního teoretického algoritmu propagace *SH koeficientů* v *LPV* je každá z obou variant počítáná metodou sběru nebo rozptylu *SH* koeficientů. Metoda rozptylu se nijak neliší od původní metody a naopak metoda sběru je počítána zpětně, kdy okolní buňky jsou považovány za zdrojové a středová buňka jako cílová. Princip propagace metodou rozptylu shrnuje algoritmus 6.2.

GS varianta využívá pro metodu sběru jednoduché kreslení čtverců do *framebufferu* pro spuštění *fragment shaderu* nad každým vzorkem a pro metodu rozptylu využívá *point rendering* obdobným způsobem jako v případě inicializace. *CS* varianta v obou případech využívá *image jednotky* pro náhodný přístup a zápis dat do textur. V případě metody rozptylu využívá *atomické operace*.

6.9.3 Použití výsledků

Při finálním vykreslování scény je využita pouze *akumulační textura*, ze které jsou čteny hodnoty dle pozice pozice, normály a vzálenosti vzorku ve scéně pro zvolení úrovně kaskády *LPV*. Vyhodnocení probíhá s využitím vzorce 4.2 ve směru obrácené normály vzorku pro každou barevnou složku.

Algoritmus 6.2: Propagace *SH koeficientů* jedné buňky.

```
1:  načti SH koeficienty a aktuální (zdrojové) buňky;
2:  for pro každou ze 6 okolních buněk do
3:      vypočti pozici sousední buňky;
4:      vynuluj lokální proměnné pro SH koeficienty;
5:      for pro každou z 5 nesdílených stěn v sousední buňce do
6:          urči vektor v ze zdrojové buňky na střed stěny sousední buňky;
7:          urči SH koeficienty b v tomto směru v pomocí vzorce 4.2;
8:          urči SH koeficienty c sevřeného kosínového laloku ve směru ze středu
            sousední buňky na střed stěny pomocí vzorce 4.3;
9:          vyhodnoť SH koeficienty a s b pomocí kladného skalárního součinu;
10:         vynásob výsledek s SH koeficienty c a s hodnotou pevného úhlu;
11:         if je povoleno použít mřížku zastínění then
12:             vyčti SH koeficienty d z mřížky zastínění na pozici průniku v se
                sdílenou stěnou mezi buňkami;
13:             vyhodnoť SH koeficienty b s d a výsledek odečti od 1.0;
14:             výnásob SH koeficienty s výsledkem;
15:         end if
16:         přičti výsledek k výsledkům reprojekcí na ostatní stěny;
17:     end for
18:     přičti výsledek k aktuálním hodnotám sousední buňky;
19: end for
```

6.9.4 Rozšíření

K prvnímu z navržených rozšíření řešící usměrnění propagace *SH koeficientů* je potřeba modifikovat původní algoritmus metody *LPV*. Pro výpočet je nutné přidat dvě přepínané 3D textury pro uložení dilatovaných normál. Na počátku výpočtu jsou tyto textury, jako ostatní textury *LPV*, vymazány a během injekce *VPL* do *LPV* jsou do jedné z textur uloženy normály *VPL* na stejných pozicích. Před každou propagaci je druhá z nepoužitých textur vymazána pro uložení výstupu a poté je pro každou buňku vypočtena dilatace metodou sběru – tzn. každá buňka hledá ve svém okolí normály, které by použila. Pokud existuje normál více, určí se jejich průměr. Je důležité, aby tento krok proběhl před samotným výpočtem propagace *LPV*, jelikož buňky *LPV* využívají tyto data pro správně usměrnění. Úpravu původního algoritmu propagace *SH koeficientů* v *LPV* zachycuje pseudokód 6.3.

Další z navržených rozšíření pro vložení *VPL* z pozadí scény vyžaduje vytvoření *RSM* při pohledu shora dolů na scénu a při vložení jsou vyhodnocovány všechny buňky *LPV* jako v případě propagace. Daná buňka poté zjišťuje, zda přidělený vzorek z *RSM* je ve scéně níž než pozice buňky. V kladném případě je v buňce vytvořeno nové *VPL* směrem dolů. V opačném případě se předpokládá, že daná buňka je vzorkem z *RSM* zastíněna.

Rozšíření *SSLPV* je z hlediska návrhu pouze kopií původní metody *LPV*. Při injekci je potřeba rozeznat průsvitné objekty, které jsou kresleny do *RSM*. Injekce a propagace probíhá zcela nezávisle na původní metodě a při stínování během vykreslování jsou data z *SSLPV* čtena pouze pro průsvitné objekty, pro které byla metoda počítána.

Algoritmus 6.3: Úprava původního algoritmu propagace *SH koeficientů* pomocí usměrnění.

1: *...po vyhodnocení SH koeficientů* `a`, `b` *a* `c`*...*
2: urči SH koeficienty `e` ve směru dilatované normály pomocí vzorce 4.2;
3: urči SH koeficienty `f` sevřeného kosínového laloku ve směru dilatované normály pomocí vzorce 4.3;
4: vyhodnoť SH koeficienty výsledku původní metody s `e` pomocí kladného skalárního součinu;
5: vynásob výsledek s SH koeficienty `f`;

Poslední rozšíření autorů metody *LPV* nepotřebuje žádné dodatečné textury pro uložení, ale pouze doplňuje krok vykreslení scény o metodu *ray – marchingu* v *LPV*.

6.10 Sluneční paprsky

Cílem této techniky se imitace paprsků zdroje světla v mlze, například od slunce (obr. 6.6). Výpočet spočívá ve vytvoření dvou *framebufferů*, kdy do prvního je vykreslena specificky obarvená scéna: geometrie slunce je vybarvena světlou až bílou barvou, pozadí středně tmavou barvou a vše ostatní černou barvou. Poté je určena 2D pozice zdroje světla na projekčním plátně vzhledem k transformaci kamery a následně je vykreslena textura z předchozího kroku do druhého *framebufferu*. Během druhého vykreslování je pro každý vzorek vypočten vektor ve směru 2D pozice zdroje světla, pomocí kterého je cyklicky vzorkována první textura s pevně daným krokem. Výsledkem je rozmazaná textura směrem ven (od středu 2D pozice zdroje světla). Tato textura je vykreslena přes celou obrazovku na závěr vykreslovací smyčky.

6.11 Animace kamery

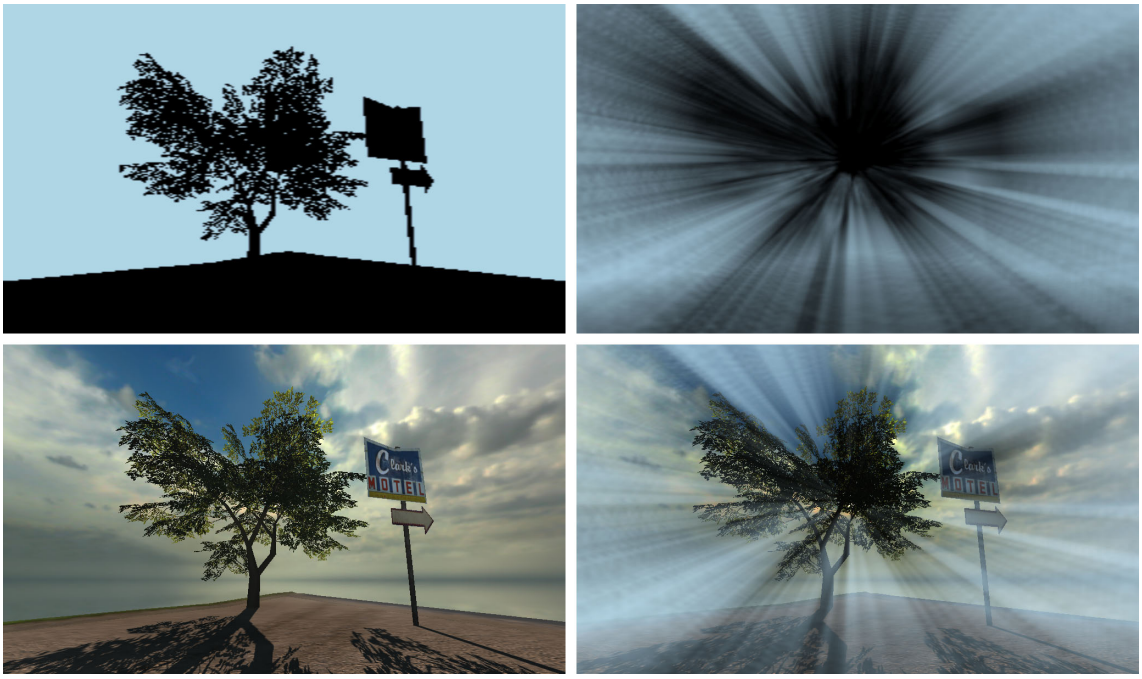
Součásti výsledké aplikace byla možnost spustit animaci kamery, která proletí scénou a demonstruje vizuální výsledky metody *LPV*. Proto byly ve scéně definovány body pro trajektorii kamery (pozici a rotaci), které představovaly řídící body pro *Catmull – Rom Spline* [2]. Tento typ splajnu byl vybrán z důvodu procházení křivky řídícími body.

Po spuštění animace je vynulován posun na splajnu, který definuje interpolační koeficient a výběr čtyř bodů splajnu. V každém snímku je vypočtena pozice a rotace kamery pomocí interpolace mezi řídícími body splajnu (rovnice 6.4) a poté je k posunu na splajnu přičten simulační krok, který odpovídá době vykreslení snímku vynásobenou konstantní rychlostí kamery.

$$x = \frac{1}{2} \cdot \begin{bmatrix} 1 & n & n^2 & n^3 \end{bmatrix} \cdot \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{6.4}$$

kde:

n	je koeficient z intervalu $< 0, 1 >$
p_i	je řídící bod



Obrázek 6.6: Ukázka metody aproximace slunečních paprsků. Vlevo dole: standardní kreslení scény, vlevo nahoře: vykreslení scény ve specifických barvách, vpravo nahoře: rozmazání předchozího výsledku směrem od pozice zdroje světla, vpravo dole: přímíchání výsledku do standardního kreslení scény.

Kapitola 7

Implementace aplikace

V této kapitole jsou popsány implementační detaily aplikace, využité knihovny při implementaci, systém zobrazení scény, mechanizmy použité při výpočtu metody *Light Propagation Volumes*, jejím rozšířením a další prvky aplikace.

Pro implementaci byl využit jazyk *C++* spolu s rozšiřující knihovnou *STL*. Pro vytvoření okna a zachytávání zpráv o vstupu uživatele byly použity knihovny *SDL* a *Qt*. Knihovna *Qt* doplňuje aplikaci o *GUI* a je alternativou k první zmiňované. Samotné vykreslování grafiky bylo prováděno pomocí knihovny *OpenGL*. Pro matematické výpočty byla využita rozšižující knihovna *GLM* a podporu pro načítání obrázků ve speciálních formátech zajisťovala knihovna *SDL Image*.

7.1 Jádro aplikace

Aplikace je reprezentována třídou `CEngine`, která zapouzdřuje veškeré moduly včetně okna, ovládání, scén a vykreslování. Každý modul je také reprezentován svou třídou, například scény třídou `CScenes` nebo textury třídou `CMaps`, apod. Veškeré třídy, které chtějí přistupovat k ostatním třídám v enginu dědí z bázové třídy `CEngineBase`. Tato třída obsahuje kontext `CContext`, který obsahuje odkazy na všechny moduly.

Po spuštění a inializaci enginu, kterou představuje proces vytvoření instancí modulů a jejich propojení kontextem, je v případě využití knihovny *Qt* ve třídě `CEngine` vytvořeno okno aplikace a *GUI*. V případě použití knihovny *SDL* je okno aplikace definováno ve třídě `CWindow`, ve které je také vytvořen *OpenGL* kontext, *shadery*, *framebuffery* a další věci nutné k běhu aplikace. Tato třída v případě *Qt* varianty dědí z vestavěné třídy `QGLWindow` (resp. `QOpenGLWindow` pro novější verzi *Qt*), ve které je *OpenGL* kontext vytvořen automaticky. Aplikace pro načtení *OpenGL* funkcí nepoužívá knihovny třetích stran, ale načítání řeší třída `COpenGL`, která inicializuje pouze *OpenGL* funkce jádra. Nutno podotknout, že třída `CWindow` řídí veškeré spouštění kreslení scény, stínů, *RSM* a výpočet *LPV*.

7.2 Zobrazovací průchod

V sekci 6.4 návrhu aplikace bylo zmíněno, že scéna není překreslována neustále, ale pouze v případě vstupu uživatele. Pro správné určování simulačního kroku a pro cyklické překreslování v případě držení kláves pohybu, je v aplikaci spuštěn neustále běžící časovač, který je určen buď identifikátorem `SDL_TimerID` nebo třídou `QTimer`. Při vypršení časovače nesmí být scéna vykreslena přímo, protože *OpenGL* kontext není validní ve vláknu, ve kterém běží

tento časovač. Proto je překreslení naplánováno, v případě *SDL* aplikace, novou událostí nebo v případě *Qt* aplikace pomocí *signálů* a *slotů*.

V této sekci byla také popsána zorazovací smyčka, která začíná ve třídě `CScene`, kde jsou zpracovávány instance objektů scény `CSceneObject`. Pokud je objekt představuje model, jsou zpracovány jeho dílčí *meshe* ve třídě `CModel`. Následně je seřazeno a naplánováno vykreslení geometrií v *rendereru* `CRenderer` s využitím nastavení zvoleného *shader programu* `CShaderProgram`.

Ke standardnímu vykreslení scény se využívá několik *shaderů* iluminace, ve kterých je počítán *Blinn – Phong* osvětlovací model, stíny, a je používán výsledek metod *LPV* a *SSLPV*. Nicméně v mnoha již zmíněných případech je potřeba vykreslit celou scénu s určitými vlastnosti ke specifickému účelu, například k získání hloubky, *RSM*, apod. Z toho důvodu třída `CRenderer` disponuje řadou přepínačů, kterými lze zvolit určitý režim kreslení, který je spjat s jedním nebo několika *shadery*. Příkladem může být přepínač `NRenderer::MODE_DEPTH_CASCADE`, který několikrát vykreslí hlouku různě transformované scény do předem definovaných dlaždic.

7.3 Kaskády stínů a RSM

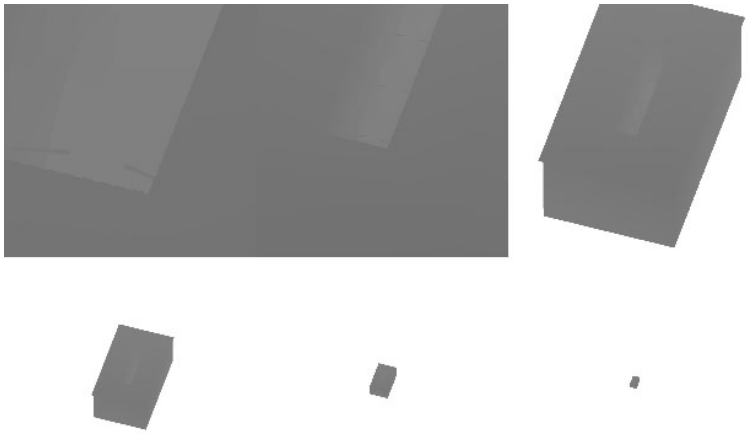
Princip tvorby stínů byl popsán v sekci 6.7 a rozšíření o kaskády v sekci 6.8. Základní metoda stínů potřebuje ke své funkci *framebuffer* s jednou přiloženou texturou typu `DEPTH_COMPONENT16|24|32`. Tato textura je při finálním vykreslování scény připojena do *shaderu* na *texturovací jednotku* typu `sampler2DShadow` včetně spravného nastavení režimu komparace textury `GL_COMPARE_REF_TO_TEXTURE`. Nicméně metoda hladkých stínů potřebuje, pro správné zjišťování vzdáleností fragmentů, tuto texturu použít ve standardním režimu bez komparace. *OpenGL* nedovoluje připojit v jeden okamžik stejnou texturu k různým *texturovacím jednotkám* s rozdílnými parametry, a proto byla textura hloubky rozkopírována ve výstupu *shaderu* hloubky s využitím připojení druhé textury do *framebufferu*.

V sekci návrhu byla popsána metoda využití dlaždic ve 2D textuře pro úrovně kaskád. Tato technika nahradila původní metodu, která využívala 2D pole textur. Vrstva textury byla určována stejným způsobem jako v aktuální implementaci a byla zvolena v *geometry shaderu* pomocí vestavěné proměnné `gl_Layer`. Nicméně z důvodu kombinace složitých shaderů a nutností použít *geometry shader* dosahovala tato implementace velmi drastických snímkových propadů. Proto byla nahrazena systémem dlaždic, který transformuje všechny úrovně kaskád do dlaždic v jedné 2D textuře s využitím ořezových rovin `gl_ClipDistance` (obr. 7.1 a 7.2).

Systém kaskád stínů a *RSM*, který využívá funkci *instancingu*, před vykreslením zjišťuje náležitost geometrie do různých úrovní kaskád. Dle výskytu pak sestaví pole o maximální délce počtu úrovní, do kterého je uloženo propojení *instance* s úrovní kaskády. *Vertex shader* poté může s využitím vestavěné proměnné `gl_InstanceID` a pole propojení zjistit číslo instance a úroveň, do které daná *instance* patří. Vrcholy různých *instancí*, poté může transformovat pomocí odlišných *mvp* matic.

7.4 Light Propagation Volumes

Sekce 6.9 naznačila, že metoda *LPV* je implementována ve dvou hlavních variantách využívající *grafický řetězec* (*GS*) a *compute shadery* (*CS*). V prvním případě se předpokládá,



Obrázek 7.1: Ukázka vykreslení dlaždic hloubky scén pro využití při tvorbě stínů s několika úrovněmi kaskády.

že uživatel bude disponovat grafickou kartou zvládající *OpenGL* ve verzi `3.2` a ve druhém případě *OpenGL* alespoň verzi `4.3`.

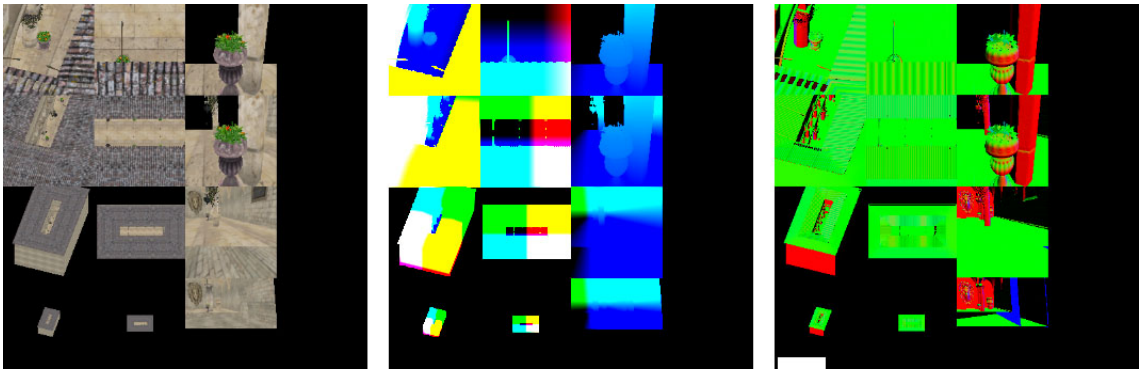
GS varianta využívá k výpočtu základní metody *LPV* tři *framebuffery*. První dva jsou přepínané mezi propagacemi a je k nim připojeno celkem šest 3D textur pro uložení RGB kanálů *SH koeficientů LPV* pro poslední výsledek a akumulaci koeficientů. Třetí *framebuffer* využívá pouze jednu 3D texturu pro mřížku zastínění. Všechny textury jsou ve formátu `GL_RGBA32F`. Z důvodu snížení počtu textur, jsou jednotlivé úrovně kaskád *LPV* a *GV* uloženy vedle sebe v ose X textury.

CS varianta je velmi omezena na využití počtu *image jednotek* – maximální počet je osm, a proto jsou použity pouze čtyři 3D textury s významem dvou přepínaných *LPV mřížek*, akumulační *LPV* mřížkou a *GV* mřížkou. Jelikož *compute shadery* nad těmito texturami využívají *atomické operace*, které jsou definovány pouze nad jednokanálovým celočíselným typem textury, jsou tyto textury formátu dat `GL_R32I` a jednotlivé barevné kanály jsou prokládány také v ose X.

7.4.1 Vymazání

Před vložením *SH koeficientů* do mřížek nutné všechny textury vymazat. *Framebuffery* u *GS* varianty využívající 3D textury lze vymazat voláním funkce `glClear()`, nicméně tento způsob nefunguje u starších grafických karet, kde je vymazána pouze nultá vrstva textury. Proto jsou textury vymazány vykreslením černé geometrie přes celý rozměr textury v každé vrstvě. K tomuto procesu je nutné povolit *míchání* barev s nastavenými oběma faktory na `GL_ZERO`. Cílem je co nejjednoduššeji spustit *fragment shader* pro všechny *body* ve 3D textuře, který zapíše nuly do všech připojených textur. Toho lze docílit vykreslením bodů, jejichž počet odpovídá hloubce *LPV* textur. V *geometry shaderu* jsou tyto body rozvinuty na čtverce překrývající celý *NDC* prostor v XY osách a Z osa je určena vrstvou pomocí `gl_Layer`. Tím lze vytvořit takovou geometrii, která pokryje všechny body v připojených 3D texturách.

V novějších verzích *OpenGL* pro *CS* variantu existuje funkce `glClearTexImage` pro



Obrázek 7.2: Ukázka vytvoření *RSM* textury obsahující difuzní barvu, pozici a normálu. Řádky odpovídají jednotlivým úrovním kaskády a sloupce daným pohledům kamery pro *RSM*: ze směru slunce, z pohledu shora a z aktuálního pohledu pro injekci do *GV*.

nastavení obsahu textur. Nicméně tato funkce v implemtaci nefungovala správně, a proto jsou textury, pro *CS* variantu, vymazány *compute shaderem*.

7.4.2 Inicializace

Vložení *VPL* z *RSM* v případě *GS* varianty je provedeno pomocí *point renderingu*. To znamená, že je vykreslen počet bodů odpovídající počtu fragmentů v *RSM*. Každý bod je v *geometry shaderu* transformován na pozici cílové buňky *LPV* v *NDC* prostoru pro osy X a Y. Pro osu Z je vybrána vrstva pomocí `gl_Layer`. Nakonec je bod potvrzen s využitím funkce `EmitVertex()`. Ve *fragment shaderu* je intenzita převedena na *SH koeficienty* a uložena do RGB textur posledního výsledku *LPV* a akumulační textury. Mechanizmus vložení *blokujících potenciálů* do *GV* probíhá stejně, nicméně proces vložení je oddělen z důvodu jiného počtu vzorků v *RSM* a jiného posunu v *LPV* mřížce. Pro alespoň částečnou emulaci *atomického součtu* je vhodné povolit *míchání barev* s nastavenými oběma faktory na `GL_ONE`.

CS varianta s *compute shadery* je spuštěna s počtem *kernelů* odpovídající počtu fragmentů v *RSM*. *Compute shader* může injekci provádět v jednom kroku bez, nyní zbytečného, přepočítání do *NDC* prostoru a výsledky může akumulovat na zvolená místa v texturách pomocí atomického součtu `imageAtomicAdd`. Vložení *blokujícího potenciálu* do *GV* i v tomto případě probíhá odděleně.

7.4.3 Propagace

V sekci 6.9 byl navržen výpočet propagace *LPV* pomocí čtyř variant: s využitím standardního *grafického řetězce* (*GS*) nebo s využitím *compute shaderů* (*CS*) a každá z těchto variant počítá propagaci buď metodou sběru *SH koeficientů* nebo metodou jejich rozptylu. Implementace každé metody se liší, a proto jsou v následujících odstavcích popsány všechny čtyři varianty.

Výpočet jedné iterace propagace variantou *GS* metodou sběru využívá k výpočtu především *geometry shadery* a *fragment shadery*. Na počátku každé interace je vymazána sada textur, která je spjata s jedním ze dvou přepínaných *framebufferů*, postupem shrnutým v sekci 7.4.1. Poté je zvolen druhý *framebuffer* a stejným principem, jako v případě vymazání,

je spuštěn *fragment shader* pro výpočet propagace každého fragmentu 3D textur. Během propagace jsou čteny připojené textury druhého *framebufferu* obsahující data z inicializace nebo z předchozí propagace.

GS varianta počítající *LPV* metodou rozptylu je liší v požadavku zápisu hodnot na jiná místa, než je aktuálně zpracovávaná buňka *LPV*. Proto se postup výpočtu inspiruje principem injekce *VPL*. Místo rozvinutí bodů je rovnou vykreslen počet bodů, který odpovídá počtu buněk v *LPV* (včetně kaskád). V *geometry shaderu* jsou body šestinásobně zkopírovány pro spuštění *fragment shaderu* v šesti okolních bodech textury. Zbylá část propagace je opět počítána ve *fragment shaderu* a výsledky jsou spolu sečteny pomocí *míchání* na straně HW.

Obě *CS* varianty již nejsou tak komplikované na řízení, nicméně z důvodu prokládání dat textur způsobené malým počtem *image jednotek* je potřeba složitěji počítat pozice v textuře. Dalším problémem je omezení typu textur na celočíselný typ a proto jsou všechny hodnoty v texturách *CS* varianty vynásobeny, při zápisu, hodnotou `1000` a při čtení stejnou hodnotou vyděleny. Tento způsob se jeví z hlediska využití číselného rozsahu jako zcela dostatečný, nicméně z důvodu čtení čtyř koeficientů a tří kanálů sekvenčně je tento přístup pomalejší. Po dokončení propagace následuje dodatečný krok, který převede naakumulované *SH koeficienty* z celočíselné podoby do textur s desetinným typem hodnot pro zjednodušení *shaderů* při finálním stínování scény.

7.4.4 Rozšíření

První z rozšíření, řešící usměrnění, přidává pouze jednu texturu ke každému *framebufferu* a jednu celočíselnou texturu pro *CS* variantu. V případě *CS* verze je textura rozdělena na dvě poloviny pro „přepínání“ čtení a zápisu. Vložení normál do textur probíhá v okamžiku injekce *VPL* do *LPV* textur a styl propagace odpovídá implementacím *GS* a *CS* sběrům.

Druhé rozšíření vkládající *VPL* ze svislého směru pouze obohacuje proces injekce o vyhodnocení zastínění každé buňky v *LPV* a z technického hlediska se implementace neliší od stylu spuštění *fragment shaderů* od předchozího rozšíření.

Rozšížení *SSLPV* duplikuje prostředky a výpočet od metody *LPV* se všemi předchozími rozšířeními. Nicméně se ukázalo jako zbytečné využívat mřížku zastínění, proto ji tato implementace nepoužívá.

Poslední rozšíření simulující neostré odlesky pouze přidává funkčnost do shaderu *iluminace* scény a předchozí výpočty nijak neovlivňuje.

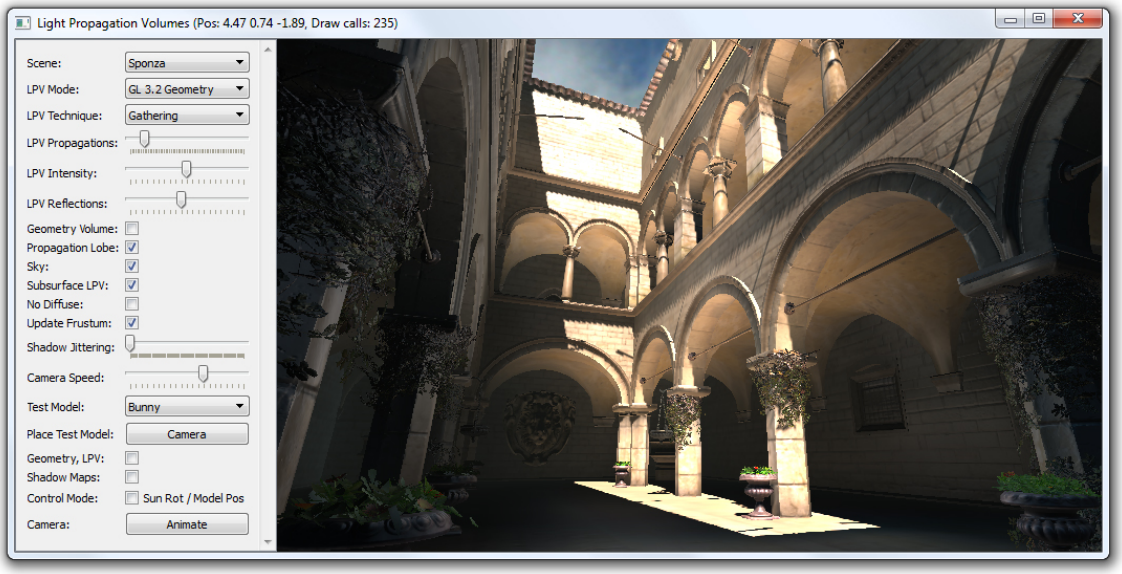
Tabulka 7.1 nakonec shruje všechny využité *framebuffery* a textury nutné pro výpočet metody *LPV* a její rozšíření.

7.5 Uživatelské rozhraní

Uživatelské rozhraní, které jistým způsobem zjednodušuje úpravu parametrů aplikace, bylo vytvořeno pomocí knihovny *Qt* a nahrazuje klávesové ovládání implementované v případě použití *SDL* knihovny. Byly použity standardní prvky *GUI* jako jsou tlačítka, posuvníky, roletové menu a zaškrtávací tlačítka (obr. 7.3).

Varianta	Označení	Typ	Uspořádání dat v ose X
GS	FBO 0: SH R	RGBA32F	kaskády
GS	FBO 0: SH G	RGBA32F	kaskády
GS	FBO 0: SH B	RGBA32F	kaskády
GS	FBO 0: SH akum. R	RGBA32F	kaskády
GS	FBO 0: SH akum. G	RGBA32F	kaskády
GS	FBO 0: SH akum. B	RGBA32F	kaskády
GS	FBO 0: normály	RGBA32F	kaskády
GS	FBO 1: SH R	RGBA32F	kaskády
GS	FBO 1: SH G	RGBA32F	kaskády
GS	FBO 1: SH B	RGBA32F	kaskády
GS	FBO 1: SH akum. R	RGBA32F	kaskády
GS	FBO 1: SH akum. G	RGBA32F	kaskády
GS	FBO 1: SH akum. B	RGBA32F	kaskády
GS	FBO 1: normály	RGBA32F	kaskády
GS	FBO GV: SH	RGBA32F	kaskády
CS	SH LPV 0	R32I	SH koef. * kanály * kaskády
CS	SH LPV 1	R32I	SH koef. * kanály * kaskády
CS	SH akum.	R32I	SH koef. * kanály * kaskády
CS	normály	R32I	xyzw * kaskády * swap
CS	SH GV	R32I	SH koef. * kaskády

Tabulka 7.1: Využití textur pro výpočet *LPV*, jejich typ a uspořádání dat. Rozšíření *SSLPV* využívá stejný počet textur bez textury pro *GV*.



Obrázek 7.3: Ukázka výsledné aplikace s uživatelským rozhraním.

Kapitola 8

Měření a výsledky

Tato kapitola shrnuje naměřené časy aplikace na dvou různých sestavách a zachycuje vizuální výsledky metody pro různé parametry nastavení.

8.1 Měření času

Aplikace byla testována na sestavách s grafickými kartami *NVIDIA GeForce* a *AMD Radeon*. Úplnou specifikaci uvádí tabulka 8.1.

Parametr	Sestava Intel	Sestava AMD
CPU	Intel Core i7 4790K 4.0 GHz	AMD Athlon 64 X2 3800+ 2.0 GHz
RAM	16 GB DDR3	2 GB DDR2
GPU	AMD Radeon R9 Fury Series 4 GB	NVIDIA GeForce GTX 960 2 GB
OS	Windows 7 Pro x64	Windows 7 Pro x64

Tabulka 8.1: Testovací sestavy pro měření.

Před uvedením výsledků měření je nutno podotknout, že vzhledem k nepoměru výkonu sestav nelze mezi těmito sestavami porovnávat naměřené hodnoty a lze je porovnat pouze na stejných sestavách v rámci vlivu nastavených parametrů aplikace.

Měření bylo provedeno na sestavách s operačním systémem *Windows 7* v *64* bitové verzi a ke zjištění času bylo nejprve počkáno na dokončení všech *OpenGL* příkazů ve frontě pomocí funkce `glFinish()` a poté byla zjištěna hodnota uběhlých cyklů systémového časovače, která byla vydělena frekvencí systémového časovače. Hodnoty lze získat pomocí *WinAPI* funkcí `QueryPerformanceCounter` a `QueryPerformanceFrequency`. Časy strávené výpočtem na *GPU* mohou být odlišné, nicméně měřené časy zachycují také procesorovou režii nutnou k přípravě scény a její vykreslení.

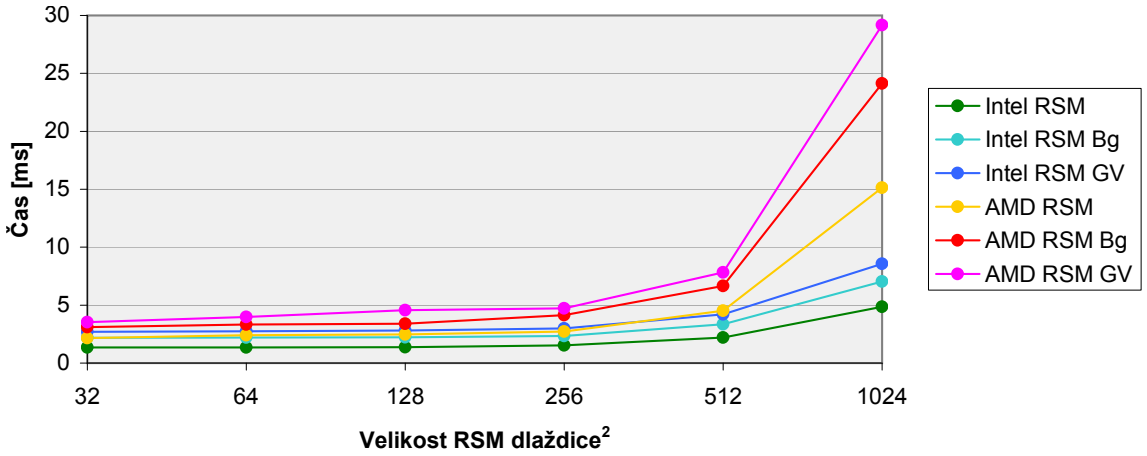
8.1.1 RSM a inicializace LPV

První s testů se zaměřuje na vliv velikosti *RSM* na dobu jeho vytvoření (tabulka 8.2 a graf 8.1) a také dobu potřebnou k vložení jednotlivých *VPL* z *RSM* do textur *LPV* (tabulky 8.3, 8.4 a graf 8.2). Měření je provedeno na obou sestavách a jsou měřeny obě implementace *GS* a *CS*. U každé z implementací je *RSM* tvořeno: základním pohledem ze směru slunce, s přidaným pohledem shora pro vložení *VPL* z pozadí (Bg sloupec) a s pohledem z pozice

aktuální kamery pro vložení *blokujících potenciálů* (GV sloupec). Ve všech následujících měřeních jsou zvoleny čtyři úrovně kaskády *RSM* a *LPV*.

Rozměr RSM dlaždice	**Sestava Intel**			**Sestava AMD**		
	RSM	RSM Bg	RSM GV	RSM	RSM Bg	RSM GV
32^2	1.34	2.18	2.69	2.17	3.10	3.54
64^2	1.36	2.20	2.74	2.40	3.32	3.98
128^2	1.38	2.23	2.82	2.47	3.39	4.57
256^2	1.54	2.34	2.98	2.73	4.13	4.72
512^2	2.21	3.35	4.21	4.52	6.66	7.83
1024^2	4.85	7.04	8.57	15.13	24.14	29.17

Tabulka 8.2: Vliv velikosti RSM na čas (v ms) vykreslení všech dlaždic scény na sestavách Intel a AMD.



Obrázek 8.1: Vliv velikosti RSM na čas (v ms) vykreslení všech dlaždic scény na sestavách Intel a AMD.

Z prvního měření lze pozorovat na obou sestavách logaritmickou časovou závislost při zvětšování *RSM* textury do velikosti 256^2. Poté se objevuje lineární časová závislost. Z hodnot lze také odvodit *CPU* režii nutnou k přípravě vykreslení scény, která odpovídá přibližně 1 ms pro první sestavu, resp. 2 ms pro druhou.

Z druhého měření vyplývá, že i když je grafická karta z první sestavy přibližně třikrát výkonnější, v tomto případě je výpočet na druhé kartě téměř dvakrát rychlejší při využití *vykreslovacího řetězce* pro vložení *VPL* do *LPV*. V případě využití *compute shaderů* jsou časy srovnané, nicméně vzhledem k výkonu karet lze usuzovat mnohem lepší využití prostředků u druhé karty.

8.1.2 Propagace LPV

V této části je měřena doba trvání různého počtu propagací s využitím čtyř implementovaných metod výpočtu. Opět byly zvoleny čtyři úrovně kaskády *LPV*. První měření zachycené tabulkou 8.5 a grafem 8.3 měří základní způsob propagace bez navržených rozšíření. Druhé

Rozměr RSM dlaždice	**Sestava Intel**					
	GS	GS Bg	GS GV	CS	CS Bg	CS GV
32^2	0.14	0.42	0.51	0.07	0.11	0.13
64^2	0.15	0.45	0.52	0.07	0.12	0.13
128^2	0.22	0.47	0.63	0.10	0.15	0.23
256^2	0.66	1.06	1.66	0.32	0.41	0.65
512^2	2.56	2.83	5.28	1.36	1.39	2.51
1024^2	13.22	14.13	23.87	6.08	6.18	10.39

Tabulka 8.3: Vliv velikosti RSM na čas (v ms) injekce do LPV na sestavě Intel.

Rozměr RSM dlaždice	**Sestava AMD**					
	GS	GS Bg	GS GV	CS	CS Bg	CS GV
32^2	0.30	0.42	0.54	0.21	0.25	0.27
64^2	0.41	0.76	0.53	0.24	0.29	0.47
128^2	0.48	0.71	0.83	0.37	0.37	0.57
256^2	0.71	0.98	1.32	0.43	0.69	0.76
512^2	1.81	2.28	3.95	1.31	1.67	2.87
1024^2	6.67	7.18	15.98	5.22	5.54	11.68

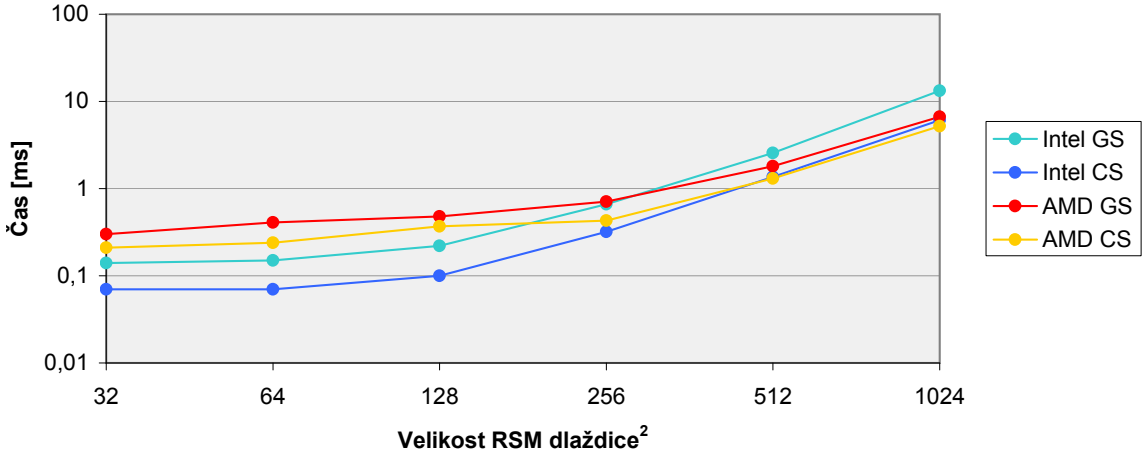
Tabulka 8.4: Vliv velikosti RSM na čas (v ms) injekce do LPV na sestavě AMD.

měření (tabulka 8.6) využívá při propagaci mřížku zastínění a poslední měření (tabulka 8.7) využívá rozšíření pro usměrnění propagace.

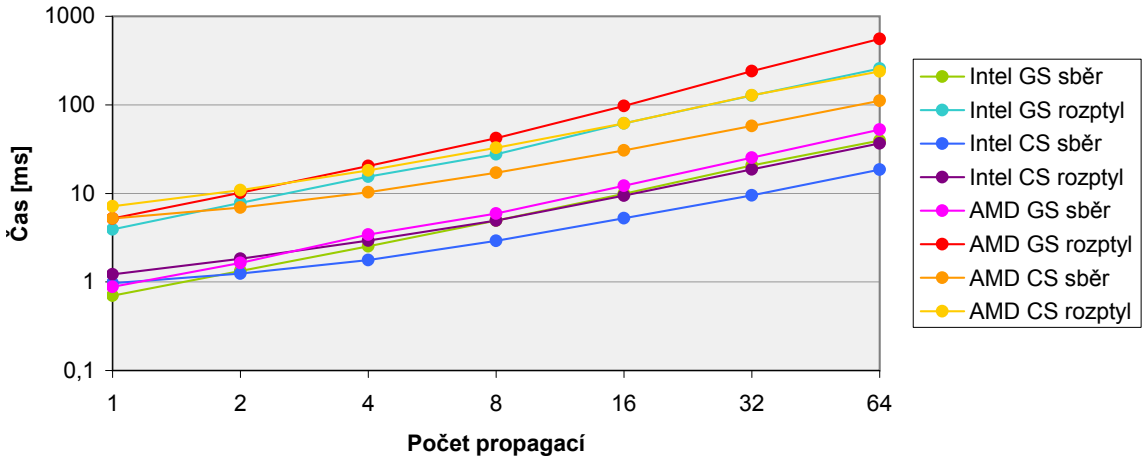
Počet propagací LPV	**Sestava Intel**				**Sestava AMD**			
	GS		CS		GS		CS	
	sběr	rozptyl	sběr	rozptyl	sběr	rozptyl	sběr	rozptyl
1	0.70	3.92	0.97	1.22	0.88	5.21	5.20	7.17
2	1.33	7.83	1.24	1.83	1.64	10.16	6.91	10.89
4	2.53	15.46	1.77	2.94	3.42	20.43	10.31	18.24
8	4.93	27.57	2.92	4.95	5.93	41.98	17.10	32.69
16	9.91	61.94	5.23	9.47	12.24	97.27	30.61	61.99
32	20.71	127.47	9.51	18.71	25.35	239.57	57.64	128.41
64	39.55	257.73	18.53	36.69	52.67	554.78	111.43	237.97

Tabulka 8.5: Vliv počtu propagací na čas (v ms) metody LPV bez rozšíření na sestavách Intel a AMD.

Ze všech naměřených hodnot je na první pohled vidět lineární závislost mezi počtem propagací a časem potřebným pro výpočet. *GS* varianta rozptylu je u všech implementací a sestav nejpomalejší z důvodu generování velkého počtu primitiv v *geometry shaderu* a použití *additivního míchání*. Implementace pomocí *compute shaderů* se zdá být na první sestavě nejrychlejší, na rozdíl od druhé sestavy, kde je daleko pomalejší od *GS* varianty sběru. Toto zpomalení lze nejspíš přisuzovat menšímu počtu paralelních jednotek na čipu karty. Využití *mřížky zastínění* a *usměrnění propagace* se zdá být podobně výpočetně náročné, přičemž



Obrázek 8.2: Vliv velikosti RSM na čas (v ms) injekce do LPV na sestavách Intel a AMD.



Obrázek 8.3: Vliv počtu propagací na čas (v ms) metody LPV bez rozšíření na sestavách Intel a AMD.

zpomalení oproti původní metodě propagace je 50 až 80 procentní pro všechny implementace kromě metody *GS* sběru, kde je zaznamenáno pouze mírné zpomalení.

Na zavěr měření je vhodné poznamenat, že výpočet rozšíření *SSLPV* všechny časy inicializace a propagace zdvojnásobí.

8.2 Grafický výstup

Následující sada obrázků zachycuje vliv různého nastavení metody *LPV* projevující se rozdílnými grafickými výsledky. Pokud nejsou explicitně určeny jiné parametry výpočtu, používají se následující:

- počet *VPL*: 128^2
- velikost *LPV*: 32^3

Počet propagací LPV (s GV)	Sestava Intel				Sestava AMD			
	GS		CS		GS		CS	
	sběr	rozptyl	sběr	rozptyl	sběr	rozptyl	sběr	rozptyl
1	1.38	9.25	1.69	2.04	1.07	5.24	6.25	7.42
2	2.56	18.84	2.62	3.24	1.83	10.73	9.19	11.32
4	2.67	18.90	2.69	3.39	3.35	20.69	14.84	19.31
8	5.12	37.68	4.36	5.89	6.49	42.11	25.96	34.54
16	10.73	74.59	9.38	12.00	13.17	96.95	40.10	65.45
32	21.41	150.34	16.38	22.62	26.97	238.77	92.74	127.58
64	41.92	310.02	31.08	45.40	53.54	556.41	181.61	251.09

Tabulka 8.6: Vliv počtu propagací na čas (v ms) metody LPV s použitím mřížky zastínění na sestavách Intel a AMD.

Počet propagací LPV (s usměr.)	Sestava Intel				Sestava AMD			
	GS		CS		GS		CS	
	sběr	rozptyl	sběr	rozptyl	sběr	rozptyl	sběr	rozptyl
1	0.81	5.16	1.19	1.26	1.05	5.24	5.33	7.34
2	1.53	8.77	1.24	2.47	1.86	10.44	7.06	11.14
4	2.83	18.73	1.82	2.96	3.45	20.63	10.67	18.86
8	6.23	49.35	3.23	5.19	6.91	42.14	17.46	34.41
16	11.39	71.55	5.17	9.91	13.84	94.53	31.51	64.69
32	22.47	143.27	9.84	18.71	29.02	233.86	59.46	126.02
64	43.68	287.82	19.38	36.22	58.63	542.81	114.69	247.45

Tabulka 8.7: Vliv počtu propagací na čas (v ms) metody LPV s použitím usměrnění propagace na sestavách Intel a AMD.

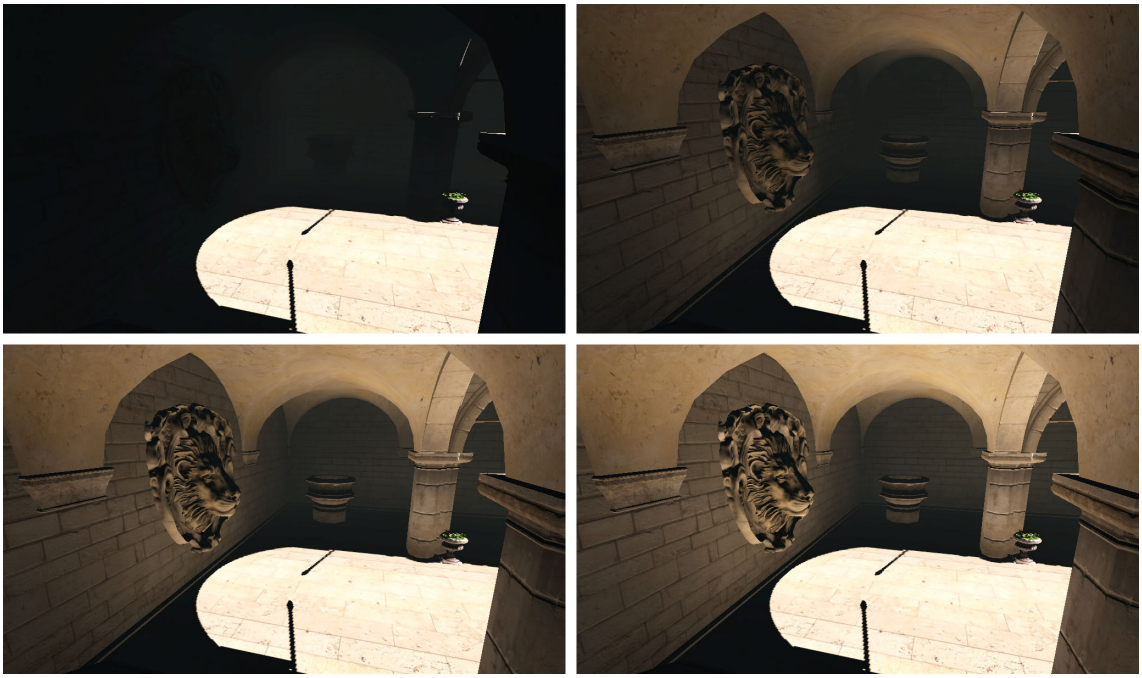
- počet propagací: 8
- počet úrovní *LPV* kaskády: 4

První ze sady obrázků (obr. 8.4) porovnává pouze lokální osvětlení scény s metodou *LPV* s různým počtem propagací. U propagací bylo aktivní rozšíření usměrnění propagace a pro každý fragment bylo vyčteno také několik hodnot z vyšších úrovní kaskády *LPV*, než do které fragment spadal. Výsledkem je, že i při nízkém nebo žádném počtu propagací lze vidět více globálního osvětlení scény.

Obrázek 8.5 porovnává jednotlivé implementace *LPV*. V případě inicializace *LPV* s počtem *VPL* 128^2 se výsledky jeví podobné. V případě vyššího počtu *VPL* jsou obě *CS* varianty přesvícené z důvodu použití *atomických operací*.

Obrázek 8.6 zobrazuje scénu, kdy byla nebo nebyla použita mřížka zastínění při propagaci. Bohužel tato mřížka ve výsledné implementaci nefunguje zcela správně, nedokáže ve většině případů zabránit průchodu světla skrz stěny a kvůli nízkému počtu vzorků v oblasti kamery dochází k nepříjemnému blikání při jejím otáčení.

Další obrázek 8.7 porovnává různé metody vložení *VPL* do *LPV*. Buď jsou vloženy pouze *VPL* z *RSM* vykreslené ze směru slunce nebo jsou vloženy *VPL* do všech vrstev *LPV*, kde dopadá světlo při pohledu shora na scénu. Pokud je spuštěno rozšíření usměrnění, může



Obrázek 8.4: Ukázka metody LPV. Vlevo nahoře: pouze lokální osvětlení se stíny, ostatní: globální osvětlení počítané metodou GS sběru, počet propagací: 0, 2, 8. Počet VPL: 128^2, velikost LPV: 32^3, počet LPV kaskád: 4.

se stát, že intenzita je ze svislého směru otočena do směru šíření původních *VPL*. Tento problém je způsoben využitím dilatací normál, které byly vytvořeny pouze z původních *VPL*.

Obrázek 8.8 porovnává právě rozšíření usměrnění toku světla při propagaci. Při použití tohoto rozšíření není osvětlena podložka, která se stala zdrojem *VPL*. Usměrnění lze také vidět na textuře cihel, kde více vystupují hrany. Doplňující obrázky 8.9 a 8.10 zachycují řezy jedné z *LPV* textur a texturu s dilatovanými normálami. V případě využití usměrnění lze z řezů *LPV* textury napravo vidět intenzitu propagovanou pouze směrem nahoru od podložky.

Rozšíření *SSLPV* zobrazuje obrázek 8.11.

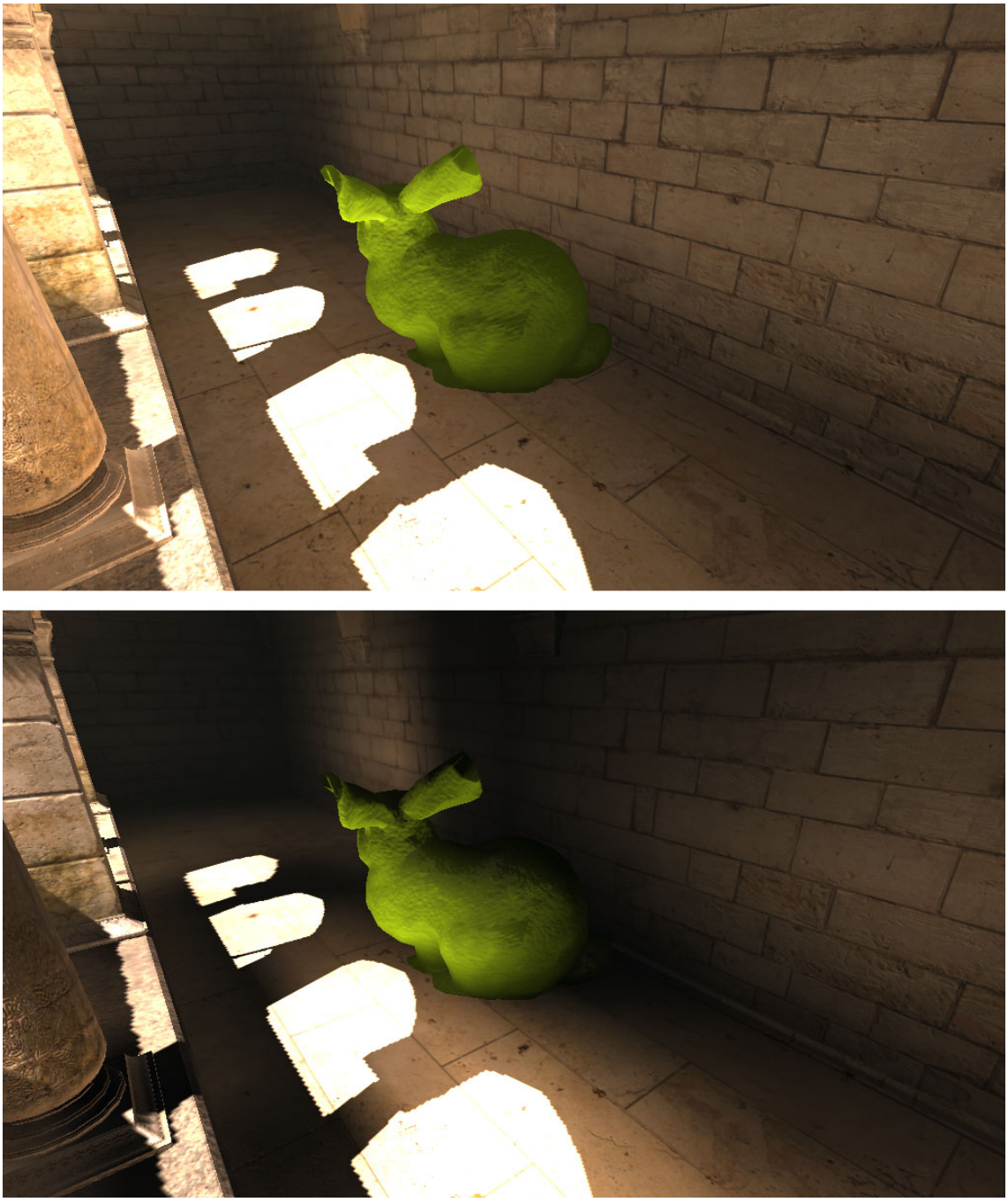
Drobné vylepšení posledního kroku metody *LPV* zachycuje obrázek 8.12, kdy nejsou čteny *SH koeficienty* pouze z jedné úrovně *LPV*, ale jsou čteny koeficienty i z vyšších úrovní *LPV*. Výsledkem je pak uvěřitelnější stínování a snížení dopadu omezení použití stejného počtu propagací pro všechy úrovně kaskády, které způsobuje problém projevující se nemožností „dopropagovat“ světlo do větších vzdáleností u nízkých úrovní v kaskádě.

Obrázek 8.13 zobrazuje jednoduché rozšíření, které využívá ray – marching v mřízce *LPV* pro zachycení odlesku. V obrázku nahoře je odlesk geometricky nejpřesnější, protože nebyla vykonána žádna propagace. Obrázky uprostřed a dole zobrazují spíše odlesk toku světla v prostoru.

Poslední obrázky 8.14 a 8.15 zachycují zbylé dvě testovací scény: Šibenik a Klarkův model ze hry *Mafia*. V případě scény kostela metoda *LPV* nefunguje velmi dobře, pokud světlo svítí shora, protože dochází k poměrně viditelným světelným průsakům. U otevřené scény s motelem lze spatřit nestabilitu metody *LPV*, při otáčení světlem, projevující se

Obrázek 8.5: Porovnání jednotlivých implementací LPV. Nahoře: GS varianta, dole: CS varianta, vlevo: sběr, vpravo rozptyl. Počet VPL: 128^2, velikost LPV: 32^3, počet propagací: 8, počet LPV kaskád: 4.

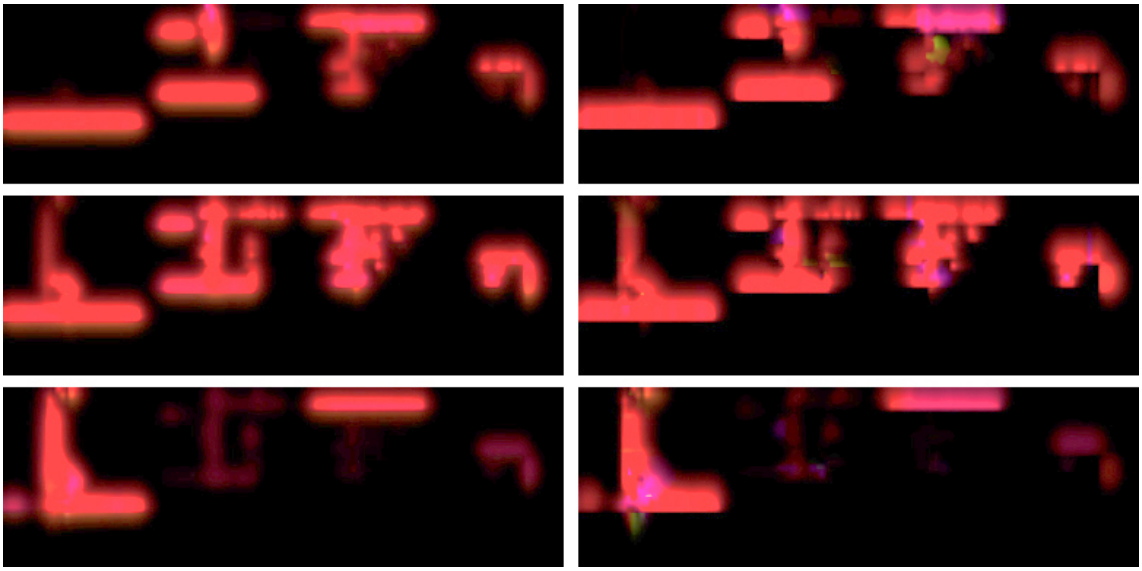
blikáním. Tento problém je způsoben nízkým počtem *VPL* a také aliasem vzniklým při vkládání *VPL* do *LPV*.



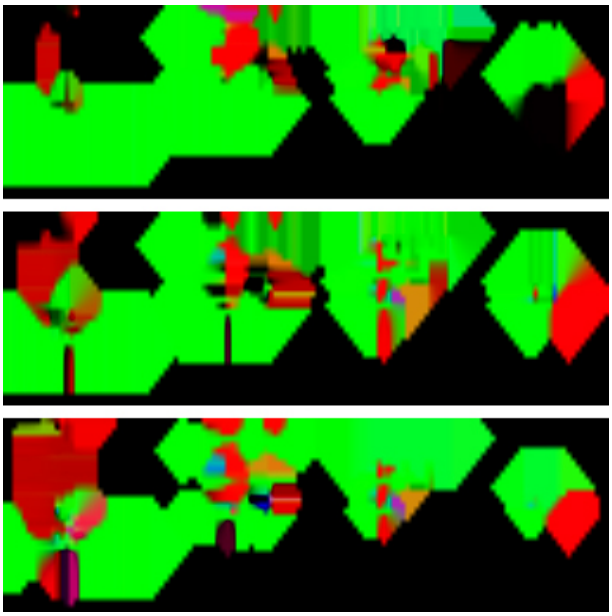
Obrázek 8.6: Porovnání propagace bez využití (nahoře) a s využitím (dole) mřížky zastínění.

Obrázek 8.7: Porovnání lokálního osvětlení (nahoře) s metodou LPV. Uprostřed: původní metoda inicializace, dole: vložení VPL z pohledu shora s barvou pozadí scény.

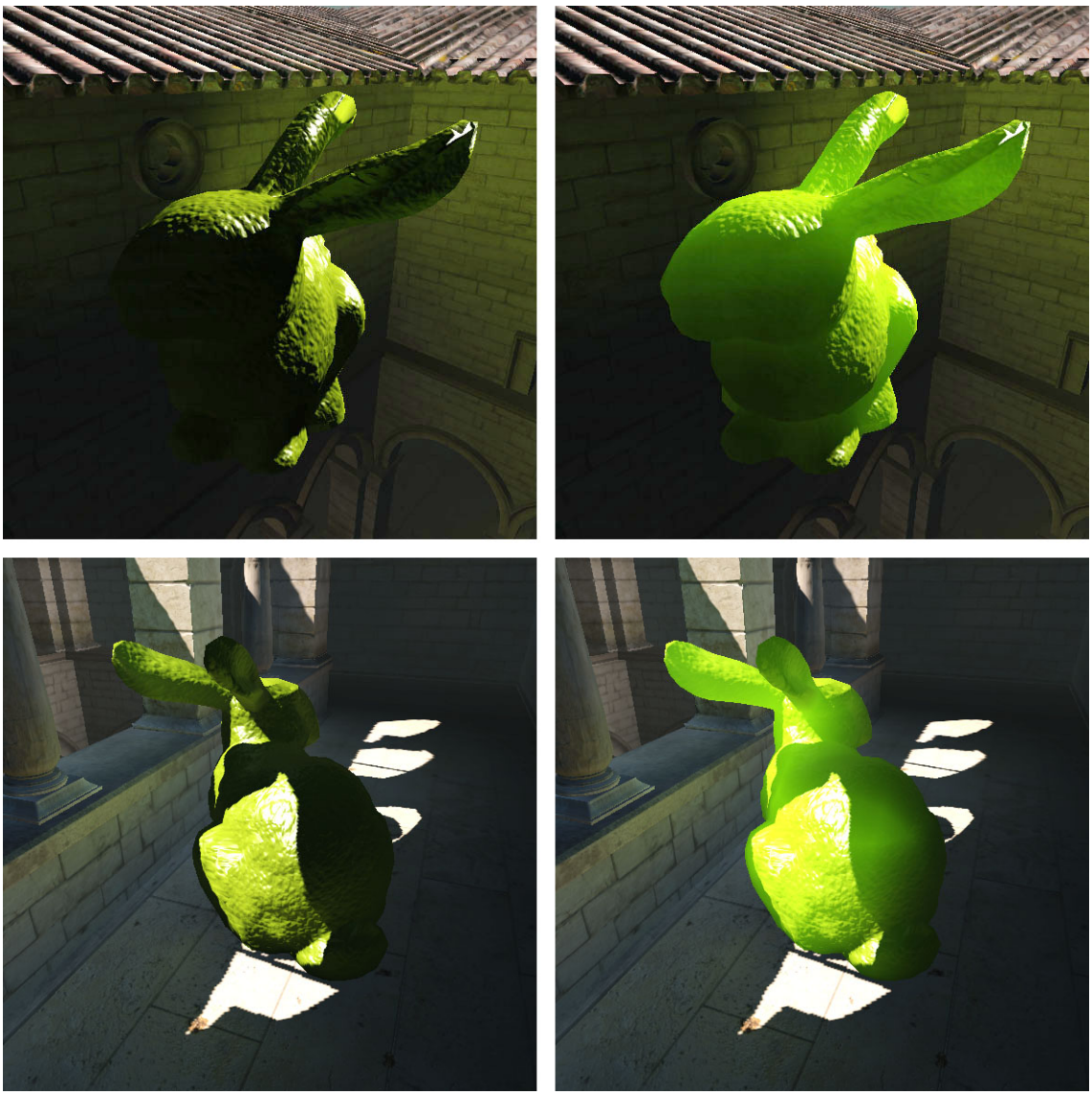
Obrázek 8.8: Porovnání původní metody (nahoře) s rozšířením, které řeší usměrnění toku světla při propagaci (dole).



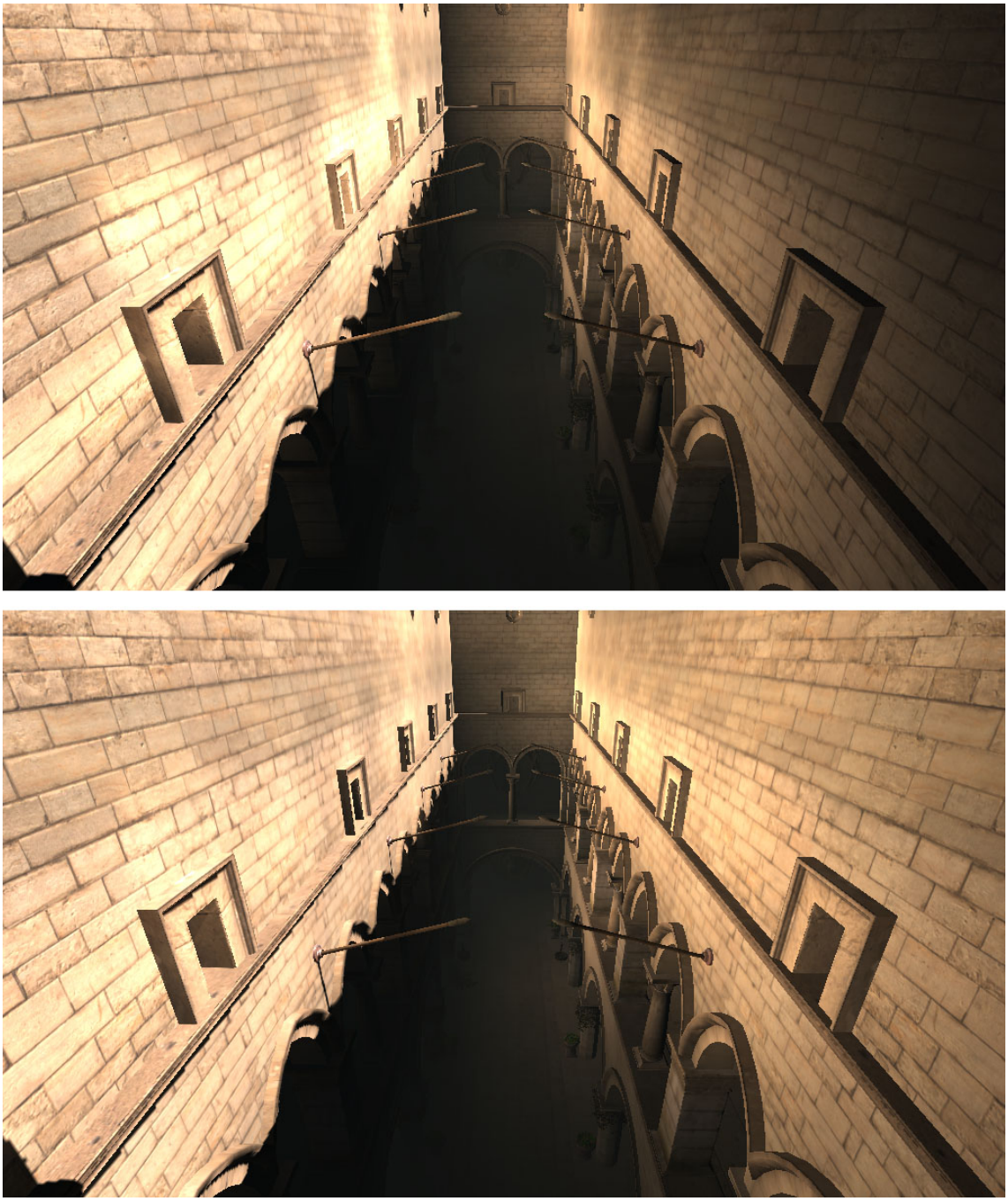
Obrázek 8.9: Ukázka tří řezů LPV textury při pohledu z boku s využitím metody GS sběru bez usměrnění (vlevo) a s usměrněním (vpravo). Obrázek byl zesvětlen.



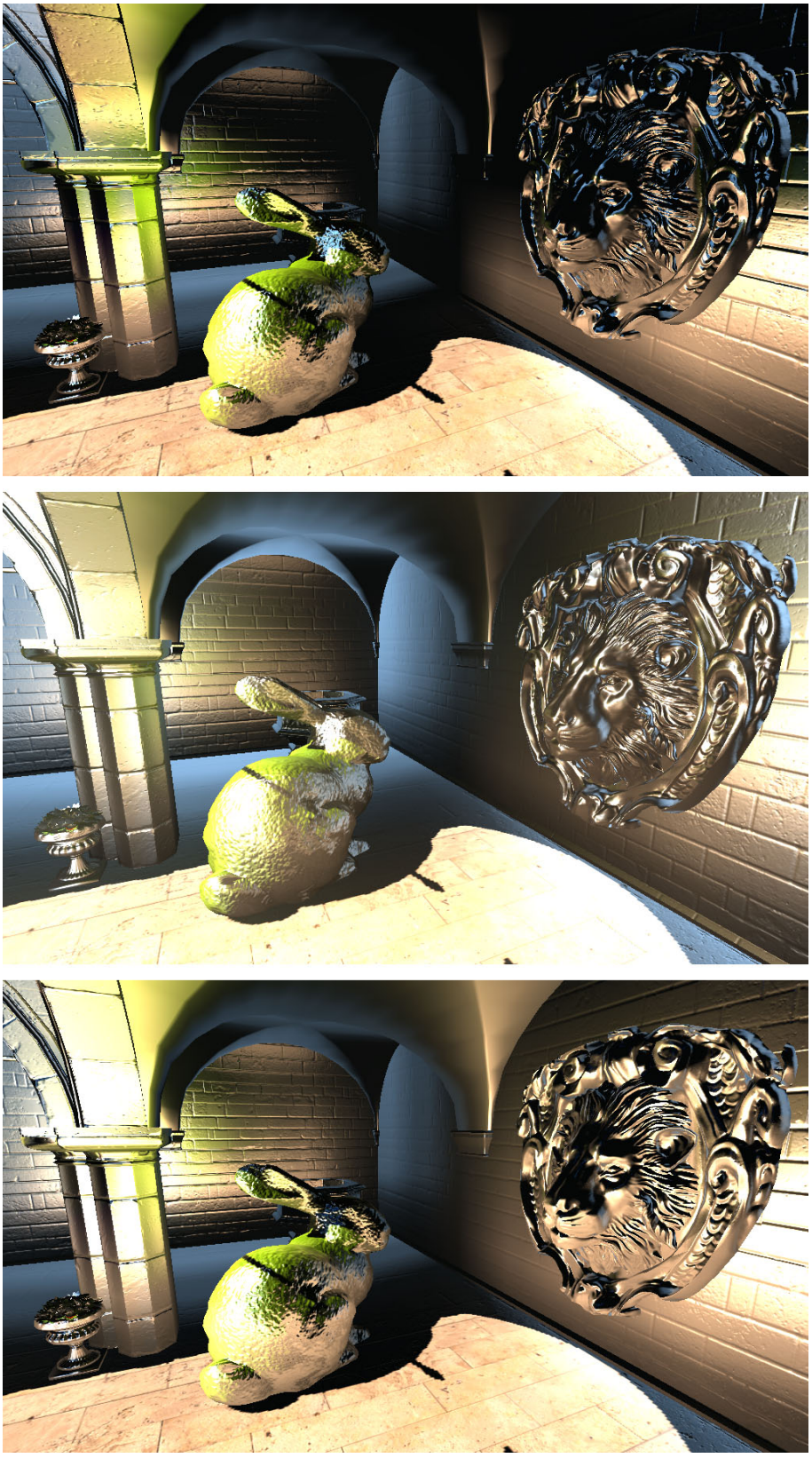
Obrázek 8.10: Ukázka tří řezů textury dilatovaných normál při pohledu z boku.



Obrázek 8.11: Porovnání metody LPV (vlevo) s kombinací s metodou SSLPV (vpravo) pro imitaci průsvitných materiálů.



Obrázek 8.12: Porovnání stínování scény, kdy jsou pro každý fragment vyčteny koeficienty pouze z jedné úrovně LPV (nahoře) a ze tří úrovní (dole).



Obrázek 8.13: Porovnání odlesků získaných z LPV. Nahoře: bez propagace, uprostřed: 8 propagací, dole: 8 propagací s usměrněním.

Obrázek 8.14: Ukázka metody LPV v modelu Šibeniku.

Obrázek 8.15: Porovnání lokálního osvětlení (nahoře) s metodou LPV (dole) na modelu Klarkův motel.

Kapitola 9

Závěr

Cílem teoretické části práce bylo v kapitole 2 stručné představení pojmů osvětlovacích modelů a jejich využití v počítačové grafice. Dále bylo cílem podrobněji se zabývat v kapitole 3 skupinou globálních osvětlovacích metod se zaměřením na výpočet v reálném čase. Hlavní metodě *Light Propagation Volumes*, tvořící jádro práce, byla věnována celá kapitola 4 obsahující podrobný popis třech kroků výpočtu metody: inicializace, propagace a vykreslení. V kapitole 5 byla zmíněna některá existující rozšíření této metody a především zde byl uveden teoretický základ vlastních tří rozšíření, které zlepšují vizuální kvalitu metody. Samotný návrh aplikace byl popsán v kapitole 6 a její implementace v kapitole 7. V kapitole 8 bylo provedeno důkladné měření metody na dvou sestavách a také byly porovnány grafické výsledky při různých parametrech metody.

Při implementaci byl použit vlastní zobrazovací engine, který byl rozšířen v mnoha směrech, především o výpočet metody *Cascaded Light Propagation Volumes*. Implementace metody byla provedena s ohledem na kompatibilitu starého HW a výkon nového HW. Proto výsledná aplikace může metodu počítat pomocí zavedeného *grafického řetězce* a také pomocí *compute shaderů*. Každá z těchto implementací může k výpočtu přistupovat buď zpětně nebo dopředně.

Tři navržená rozšíření řeší poblémy: usměrnění propagace pro odstínění zpětné propagace na zdrojovou podložku, dále rozšíření pro zlepšení globálního osvětlení scény pomocí vložení světelné intenzity z pozadí scény a rozšíření výpočtu metody o průchod světla skrz průsvitné materiály.

Dle měření, při výpočtu propagací *LPV* na grafické kartě *R9 Fury* je nejrychlejší implementace pomocí *compute shaderů* metodou sběru *SH koeficientů*, která dokáže výsledek 8 propagací spočítat za necelé 3 ms celkový výsledek za 5 ms. Zatímco u karty *GTX 960* se zdá jako nejrychlejší, implementace pomocí *grafického řetězce*, také metodou sběru. Ta dokáže celkový výsledek spočítat za 8 ms s využitím 4 úrovní kaskády *LPV*.

Součástí práce bylo zveřejnění zdrojových kódů implementace[1] pod svobodnou licencí a vytvoření demonstračního videa[2].

Možností pro další rozšíření funkčnosti nebo optimalizaci stávající metody je několik. Především by šla implementace pomocí *compute shaderů* nezanedbatelně zrychlit pomocí *lokální paměti*. Dále by šla optimalizovat využitelnost textur pro *compute shadery* slučováním několika datových položek dohromady a proces propagace lze obohatit o podporu více světelných odrazů.

[1] Zdrojové kódy práce jsou dostupné na adrese: https://github.com/djbozkosz/Light-Propagation-Volumes

[2] Video lze dohledat na kanálu autora: https://www.youtube.com/user/djbozkosz

Literatura

[1] Blinn, J.: Models of Light Reflection for Computer Synthesized Pictures. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '77, New York, NY, USA: ACM, 1977, s. 192–198.

[2] Catmull, E.; Rom, R.: A class of local interpolating splines. *Computer Aided Geometric Design*, 1974: s. 317–326.

[3] Crassin, C.; Neyret, F.; Sainz, M.; aj.: Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum*, 2011, ISSN 1467–8659.

[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.

[5] Elvek, J.: *Extending OGRE with Light Propagation Volumes*. Diplomová práce, University of Gothenburg, Göteborg, Sweden, 2012.

[6] Green, R.: Spherical Harmonic Lighting: The Gritty Details. In *Archives of the Game Developers Conference*, GDC '03, 2003.

[7] Green, S.: *GPU Gems*, kapitola Real – time approximations to subsurface scattering. Addison – Wesley Boston, MA, první vydání, 2004, ISBN 978-0321228321, s. 263–278. URL http://http.developer.nvidia.com/GPUGems/gpugems_ch16.html

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

[9] Kaplanyan, A.: Light Propagation Volumes in CryEngine 3. In *Advances in Real – Time Rendering in 3D Graphics and Games Course*, SIGGRAPH '09, USA, 2009.

[10] 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.

[11] Kasyan, N.: Playing with Real – Time Shadows. [online], 2013. URL http://www.crytek.com/download/PlayingwithReal-TimeShadows.pdf

[12] Keller, A.: Instant Radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, New York, NY, USA: ACM Press/Addison – Wesley Publishing Co., 1997, ISBN 0-89791-896-7, s. 49–56.

[13] Kirsch, A.: Light Propagation Volumes – Annotations. [online], 2010.
URL
http://blog.blackhc.net/wp-content/uploads/2010/07/lpv-annotations.pdf

[14] Laine, S.; Saransaari, H.; Kontkanen, J.; aj.: Incremental Instant Radiosity for Real – time Indirect Illumination. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, Aire-la-Ville, Switzerland: Eurographics Association, 2007, ISBN 978-3-905673-52-4, s. 277–286.

[15] 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.

[16] Mittring, M.: Finding Next Gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, New York, NY, USA: ACM, 2007, ISBN 978-1-4503-1823-5, s. 97–121.

[17] Nicodemus, F.: Directional reflectance and emissivity of an opaque surface. *Applied Optics*, ročník 4, č. 7, 1965: s. 767–775.

[18] Růžička, T.: *Nástroj pro výpočet světelných map*. Bakalářská práce, Vysoké učení technické v Brně, Brno, 2014.

[19] Ritschel, T.; Grosch, T.; Hans P. S., Approximating Dynamic Global Illumination in Image Space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, New York, NY, USA: ACM, 2009, ISBN 978-1-60558-429-4, s. 75–82.

[20] Sloan, P.: Stupid Spherical Harmonics (SH) Tricks. In *Archives of the Game Developers Conference*, GDC '08, 2008.

[21] Uralsky, Y.: *GPU Gems 2*, kapitola Efficient Soft – Edged Shadows Using Pixel Shader Branching. Addison – Wesley Boston, MA, 2005, ISBN 0321335597.
URL http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter17.html

[22] Zhou, K.; Hou, Q.; Wang, R.; aj.: Real – time KD – tree Construction on Graphics Hardware. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, New York, NY, USA: ACM, 2008, ISBN 978-1-4503-1831-0, s. 126:1–126:11.

[23] Zhukov, S.; Inoes, A.; Kronin, G.: An ambient light illumination model. In *Rendering Techniques '98*, Eurographics, Springer Vienna, 1998, ISBN 978-3-211-83213-4, s. 45–55.

Přílohy

Seznam příloh

Příloha A

Obsah CD

V zadní části práce je možné nalézt datový nosič s následujícím obsahem:

- *application*: složka obsahující zdrojové soubory implementace a spustitelnou aplikaci v prostředí *Microsoft Windows* a datové soubory testovacích scén
- *thesis*: složka obsahující zdrojové soubory textové části práce
- *tools*: složka obsahující dodatečné nástroje
- *readme.txt*: textový soubor s popisem obsahu nosiče, návodem pro překlad a ovládání aplikace
- *thesis.pdf*: zpráva v elekronické podobě
- *video.mp4*: demonstrační video