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

VÝPOČET OSVĚTLENÍ VE SCÉNĚ V REÁLNÉM ČASE

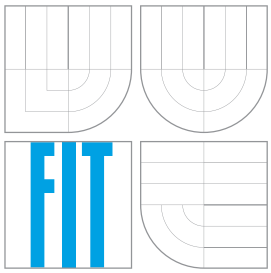
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ MARTANOVIČ

BRNO 2013

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

VÝPOČET OSVĚTLENÍ VE SCÉNĚ V REÁLNÉM ČASE

REAL-TIME ILLUMINATION OF A SCENE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUKÁŠ MARTANOVIČ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN NAVRÁTIL

BRNO 2013

Abstrakt

Práce se zaměřuje na popis metod výpočtu globálního osvětlení ve 3D grafických scénách v reálném čase. První kapitola obsahuje stručný úvod do problematiky výpočtu globálního osvětlení a zároveň připomíná principy nejběžnějších k tomu sloužících postupů. V další je představena metoda výpočtu viditelnosti při nepřímém osvětlení s využitím Imperfect Shadow Mappingu. Po bližším rozboru metody a v ní využívaných algoritmů následuje podrobnější popis praktické implementace a struktury demonstrační aplikace realizující tuto metodu. Obsahem předposlední kapitoly je následné testování a krátké zhodnocení vlastností výstupního programu. V samotném závěru práce se nachází náčrt možného rozšíření algoritmu zahrnutím zhlukování virtuálních světelných zdrojů.

Abstract

This thesis is focused on describing methods of computation of global illumination of 3D graphics scenes in real-time. First chapter contains brief introduction to the issue of global illumination (GI) computation, as well as quick summarisation of principles of most commonly used GI computation approaches. A method of visibility computing for indirect illumination, taking advantage of Imperfect Shadow Mapping, is introduced next. After closer examination of this method and prerequisite algorithms follows a description of its practical implementation, as well as of the structure of simple demonstrative application. Next chapter then contains testing and brief examination and evaluation of resulting program's behaviour. Finally, a possible method extension by means of virtual point light clustering is proposed.

Klíčová slova

Globální osvětlení, viditelnost, stínové mapy, interaktivní 3D grafika

Keywords

Global illumination, visibility, shadow maps, shadow mapping, interactive 3D graphics

Citace

Lukáš Martanovič: Výpočet osvětlení ve scéně v reálném čase, diplomová práce, Brno, FIT VUT v Brně, 2013

Výpočet osvětlení ve scéně v reálném čase

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Navrátila a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Martanovič
21. května 2013

Poděkování

Rád by som poďakoval pánu Ing. Navrátilovi za cenné rady a pripomineky, vedenie a smerovanie, ako aj poskytnutú voľnosť pri vypracúvaní tejto práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Účelom práce je oboznámiť sa s technikami výpočtu globálneho osvetlenia v 3D grafických scénach so zameraním sa na techniky umožňujúce zobrazovanie dynamických scén a interaktivitu v reálnom čase.

Bližšie bude popísaná metóda výpočtu viditeľnosti pri tvorbe nepriameho osvetlenia scény prostredníctvom tzv. Imperfect Shadow Mappingu a algoritmy, z ktorých vychádza, alebo ktoré využíva. Táto metóda je zaujímavá zahrnutím značného množstva pokročilých postupov modernej počítačovej grafiky, od offscreen renderingu a využívania viacerých cieľov renderovania súčasne, rôznych variánt a rozšírení shadow mappingu a s nimi spojených projekčných metód, ako aj „odbočky" ku generovaniu pseudonáhodných postupností, bodových reprezentácií 3D modelov (tzv. *point cloudov*), filtrovania 2D obrazových dát, či vypĺňaniu „dier" v množinách dát (*pull-push* algoritmus).

Ďalej bude popísaná podoba zamýšľanej vzorovej aplikácie, demonštrujúcej funkčnosť popisovaných postupov a priblížené rôzne aspekty jej praktickej implementácie.

Pokusy a testovanie, vykonané na rôznych modeloch a 3D grafických scénach pomocou mnou vyhotovenej implementácie algoritmu, ako aj zhodnotenie vlastností výstupného programu, by mali byť predmetom záverečnej časti tejto práce. V poslednej kapitole sa na záver navyše pokúsim o rozšírenie a zefektívnenie algorimtu návrhom potenciálneho postupu pre clustering virtuálnych svetelných zdrojov.

Kapitola 2

Techniky globálneho osvetlenia

Moderná počítačová grafika je v súčasnosti schopná takmer fotorealistického zobrazovania, resp. generovania obrazov netrénovaným okom takmer neodlíšiteľných od reality. Fyzikálne správna simulácia šírenia svetla v scéne, zohľadňujúca nielen priame osvetlenie jednotlivých nezávislých objektov, ale aj ich vzájomnú interakciu, viditeľnosť, difúzne vyžarovanie, či odrazivosť a refrakciu v materiáloch je však aj s využitím súčasného moderného hardwareu stále výpočetne veľmi náročná. V rámci snahy o priblíženie sa k realistickým scénam zároveň umožňujúcim interaktivitu v reálnom čase, teda pri vysokých rýchlostiach zobrazovania, je stále nutné pristupovať k rôznym metódam aproximácie vlastností ako osvetlenia, tak aj samotnej geometrie a materiálových vlastností zobrazovanej scény a vzájomných závislostí objektov v nej obsiahnutých a rôznym kompromisom medzi kvalitou a rýchlosťou zobrazovania.

Pod pojmom globálne osvetlenie scény môžeme zjednodušene rozumieť také spôsoby (výpočtu) osvetlenia scény, kde svetlo (resp. farba) dopadajúca do oka pozorovateľa – teda kamery – nepochádza len z priameho osvetlenia primárnymi svetelnými zdrojmi, ale je aj produktom vzájomnej interakcie svetla a objektov v scéne. Keďže v praxi je takto osvetlená väčšina viditeľných objektov v reálnom svete, je existencia globálneho osvetlenia kľúčová pre vytvorenie prirodzeného, realistického dojmu zo scény, ktorú zobrazujeme/pozorujeme.

Z hľadiska ich použiteľnosti pre zobrazovanie v reálnom čase, teda s umožnením interakcie zo strany pozorovateľa (užívateľa), môžeme existujúce techniky výpočtu globálneho osvetlenia (Global Illumination – *GI*) rámcovo rozdeliť do dvoch kategórií.

2.1 Offline metódy

Významnými medzi metódami fotorealistického zobrazovania sú dve hlavné skupiny metód. Prvou sú najmä metódy vychádzajúce zo sledovania lúčov (tzv. *Ray-Tracing*), ako sú Path-Tracing, Beam-Tracing, či Metropolis light transport. Druhou potom metódy pracujúce so simuláciou svetelného vyžarovania plošných elementov scény a ich vzájomného iteratívneho ovplyvňovania sa (radiozita a jej modifikácie). V praxi sú často využívané tzv. hybridné metódy spájajúce výpočet distribúcie vyžiareného svetla s tvorbou obrazu založenou na ray-tracingu, ako je napr. Photon Mapping, snažiace sa o spájanie výhod a „schopností" z oboch strán.

Tieto postupy síce umožňujú veľmi realistické zobrazovanie zahŕňajúce takmer fyzikálne správne zachytenie šírenia svetla v scéne, jeho interakciu s objektami a mnoho pokročilých optických javov, ako sú odrazy a lom svetla, kaustiky, interferenčné javy, subsurface

scattering a mäkké tiene, rýchlosťou zobrazovania sú však stále relatívne ďaleko od interaktivity. Ich uplatnenie je aj v súčasnosti s všadeprítomným programovateľným grafickým hardwareom obmedzené na predspracovanie a tvorbu zdrojov využiteľných pri zobrazení statických scén (napr. lightmapy, zachytávajúce predpočítané osvetlenie zo statických, nemenných svetelných zdrojov), prípadne v neinteraktívnych aplikáciách, ako napr. tvorba realistických animácií, videosekvencií či filmových špeciálnych efektov. Určite je však vhodné aspoň stručne pripomenúť princípy a myšlienky stojace za spomenutými skupinami zobrazovacích techník, keďže tvoria základ pre postupy, z ktorých vychádzajú metódy snažiace sa o realistické zobrazovanie v reálnom čase.

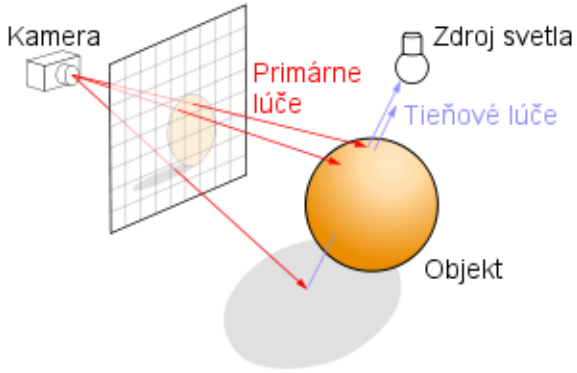
Ray-Tracing a odvodené metódy

Základnou – dnes už klasickou – metódou realistického zobrazovania, z ktorej vychádza celá plejáda ďalších, vylepšujúcich a pridávajúcich pokročilejšie grafické efekty, je Ray-Tracing. Princípom Ray-Tracingu je „vrhanie" lúčov z kamery cez jednotlivé pixely projekčnej plochy do scény. V záujme zvýšenia obrazovej kvality, resp. boja s aliasingom a inými nežiaducimi javmi, je samozrejme možné každým pixelom vrhať viac ako jeden lúč s offsetom (*supersampling*). Posunutím východzej pozície, alebo úpravou iných vlastností vysielaných lúčov, alebo ich časti, je možné simulovať rôzne optické javy závislé na vlastnostiach šošovky kamery, ako napr. hĺbka poľa. Lúče vychádzajúce priamo z kamery nazývame *primárne*. Pre každý vrhnutý lúč je následne zistený priesečník s geometriou scény. Ak tento existuje a nieje za kamerou, objekt, na ktorého ploche sa nachádza, je vyhodnotený ako priamo viditeľný z pohľadu pozorovateľa a v danom bode je spočítané osvetlenie. V tejto triviálnej podobe sa jedná o tzv. *Ray casting*.

Farebný príspevok v jednotlivých bodoch obrazu však nemusí byť len dôsledkom priameho osvetlenia objektu primárnym svetelným zdrojom. V rámci globálnych vzťahov v scéne môže byť daný bod zatienený (voči jednému, alebo viacerým svetelným zdrojom), alebo (v závislosti na materiálových vlastnostiach) môže dochádzať k odrazom, prípadne lomu lúčov na povrchu objektu v danom bode. Jednoduché primárne (tvrdé) tiene sú zisťované rekurzívnym vysielaním tzv. tieňových lúčov z priesečníku k svetelným zdrojom umiestneným v scéne. Bod je osvetlený v prípade, keď takýto lúč nenarazí na svojej ceste k svetlu na žiadnu prekážku. Mäkké tiene je možné aproximovať vrhaním väčšieho množstva tieňových lúčov, pričom pre každý je pozícia svetla mierne posunutá, prípadne (ak sa jedná o plošný svetelný zdroj) náhodným výberom bodu na tomto zdroji ako pozície svetla. Odrazy a lom svetla sú simulované rekurzívnym vysielaním sekundárnych lúčov z bodu priesečníku primárneho lúča s objektom v smere danom geometrickými (napr. normála v danom bode), resp. materiálovými (napr. index lomu) vlastnosťami objektu a vzťahmi, ktoré sú predmetom skúmania fyzikálnej optiky. Pre samotný výpočet tieňovania je následne možné použiť ľubovoľný vhodný osvetľovací model (napr. Blinn-Phongov) so zohľadnením všetkých získaných farebných (svetelných) príspevkov.

Zložením farebných príspevkov primárnych a prípadne zanorených sekundárnych lúčov na osvetlených častiach viditeľných objektov v scéne je následne daný finálny obraz.

Táto metóda je, najmä pre veľmi veľké množstvo zisťovaných priesečníkov lúčov s geometriou scény, výpočetne pomerne náročná. S pomocou existujúcich akceleračných metód (obalové telesá, priestorové delenie scény, adaptívny subsampling, ...), redukujúcich počet vysielaných lúčov, alebo počet objektov, skúmaných pri hľadaní priesečníkov s vrhnutými lúčmi a vďaka možnostiam súčasného grafického HW sú už však dnes dosiahnuteľné takmer interaktívne rýchlosti.



Obr. 2.1: Ray Tracing (zdroj: Wikipedia)

Radiozita a vyžarovacie metódy

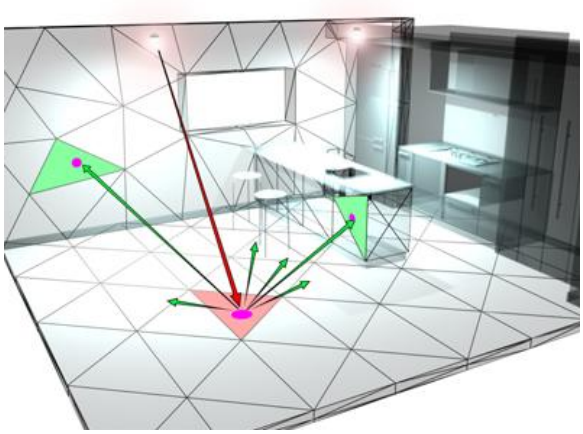
Radiozita vychádza z metód pre aproximáciu šírenia tepelného žiarenia, jedná sa teda v podstate o aplikáciu metódy konečných prvkov (FEM – *Finite Element Method*). Metóda zohľadňuje nepriame osvetlenie vzniknuté difúznym odrazom svetla, resp. difúznym vyžarovaním materiálov, z ktorých sú vytvorené objekty nachádzajúce sa v scéne. Celá geometria scény je tvorená konečnou množinou menších plôšok, predstavujúcich jednotlivé spojité úseky povrchov objektov v scéne. Pre každú dvojicu z týchto plôšok je spočítaný takzvaný *form factor* (tiež view factor), ktorý na základe materiálových (napr. odrazivosť) a geometrických (napr. vzájomné natočenie, vzdialenosť, ...) závislostí určuje, ako dobre na seba jednotlivé plôšky „vidia", a tým zároveň to, ako bude prebiehať pohlcovanie, alebo naopak odraz, či vyžiarenie svetla plôškou A voči tej-ktorej plôške B. V prípade existencie prekážok medzi plôškami môže byť form factor nulovaný, čím je indikované ich vzájomné zatienenie.

Na začiatku výpočtu je zvolená množina plôšok určená ako primárne svetelné zdroje (lampy, okná, atp.). Je tiež možné výpočet „naštartovať" aj so svetelnými zdrojmi, ktoré niesú priamou súčasťou geometire scény (slnko). Následne je iteratívne pre každú plôšku podľa form factoru voči zdrojom žiarenia spočítaná jej „osvetlenosť", resp. „žiarivosť" z jednotlivých svetelných zdrojov. Plôšky, ktoré prekročia zvolený prah, sa sami v nasledujúcej iterácii stávajú svetelnými zdrojmi. Výpočet je ukončený dosiahnutím rovnovážneho stavu, alebo – častejšie – po vopred zvolenom počte iterácií, prípadne vopred zvolenej dobe, v závislosti na želanej kvalite výsledného zobrazenia.

Radiozita podáva výborné výsledky pri výpočte mäkkých tieňov, difúznych odrazov a pod., optické javy, ako je odraz či lom svetla, zrkadlenie a pod., už však nezvláda. V kombinácii s ray-tracingom a/alebo s od neho odvodenými metódami je možné dosiahnuť takmer fotorealistické zobrazenie.

2.2 Online metódy

Na druhej strane stoja metódy umožňujúce zobrazovanie v reálnom čase a určitú mieru interaktivity a dynamiky zobrazovanej scény, alebo sa k tomuto cieľu snažiace priblížiť (tzv. *online* metódy). Svoje uplatnenie nachádzajú v interaktívnych vizualizáciach, pokročilých virtuálnych realitách, či zábavných aplikáciách vo forme počítačových hier. Medzi najvýznamnejšie môžeme zaradiť napr. Instant Radiosity [5], Instant Global Illumination [14] alebo Reflective Shadow Mapping [2] (*RSM* – väčší priestor je jeho implementácii venovaný



Obr. 2.2: Radiozita (zdroj: 3dmax-tutorials)

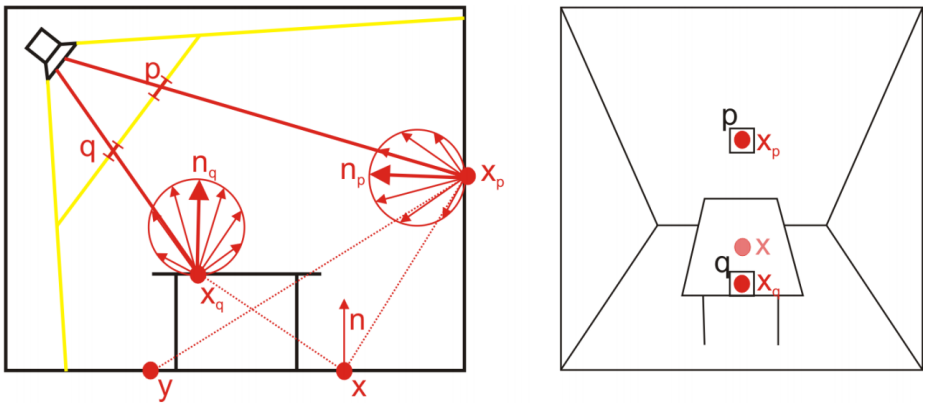
v kapitole 5). Tieto techniky vo všeobecnosti pristupujú k obetovaniu fyzikálnej správnosti práve v prospech efektivity zobrazovania, obyčajne formou jej nahradenia vhodnou aproximáciou, zohľadňujúcou vlastnosti ľudského zraku a zobrazovacích zariadení. Spomínané metódy umožňujú interaktivitu do značnej miery len pre statické, alebo geometricky veľmi jednoduché dynamické scény. Spoločným prvkom býva približný výpočet nepriameho osvetlenia v scéne prostredníctvom využitia dynamicky vytváraných sekundárnych (tzv. virtuálnych) svetelných zdrojov, čo je postup principiálne podobný zjednodušenej myšlienke radiozity. Výpočetne najnáročnejším a časovo najzložitejším býva výpočet viditeľnosti z jednotlivých virtuálnych svetelných zdrojov pomocou shadow mappingu, alebo shadow volumes (preto napr. základný algoritmus RSM viditeľnosť pri výpočte nepriameho osvetlenia zanedbáva úplne). Jeho uplatnenie je však nevyhnutné pre dosiahnutie presvedčivej kvality výsledného zobrazenia.

Jedným z možných riešení tohto problematického aspektu GI v reálnom čase je technika *Imperfect Shadow Mapping*, ktorej spracovaním, implementáciou a nasadením pre efektívny výpočet nepriameho osvetlenia v scéne sa zaoberajú neskoršie kapitoly tejto práce.

Reflective Shadow Mapping

Ako Reflective Shadow Mapping (RSM), tak Instant Radiosity vychádzajú z poznatku, že vďaka uplenteniu techník tzv. odloženého (*deferred*) shadingu je možné pomerne „lacno" do scén pridávať veľké množstvo svetelných zdrojov. RSM, ako napovedá názov, vychádza z – opäť dnes už klasickej – metódy shadow mappingu, navyše však okrem hĺbkovej inforácie ukladá aj inforácie o pozíciách, normálach a materiálových vlastnostiach (najmä difúznu farbu) objektov z pohľadu svetelného zdroja (teda vlastnosti osvetlených povrchov). Tieto informácie sú následne pravidelne, alebo (pseudo)náhodne vzorkované a použité pre generovanie tzv. virtuálnych svetelných zdrojov, simulujúcich difúzny odraz svetla od povrchov v scéne (podobne, ako sa niektoré ožiarené plôšky stávajú svetelnými zdrojmi v prípade radiozity). Uchované pozície a normály určujú pozície a smer žiarenia nepriamych svetiel, uchované materiálové vlastnosti potom farbu a prípadne útlm svetla. Pri deferred shadingu scény v screen space sa pri výpočte farby jednotlivých pixelov výsledného obrazu potom do osvetľovacieho modelu započítavajú ako príspevky od primárnych, tak od sekundárnych svetelných zdrojov.

Ako bolo spomenuté, RSM nerieši viditeľnosť voči jednotlivým virtuálnym svetelným

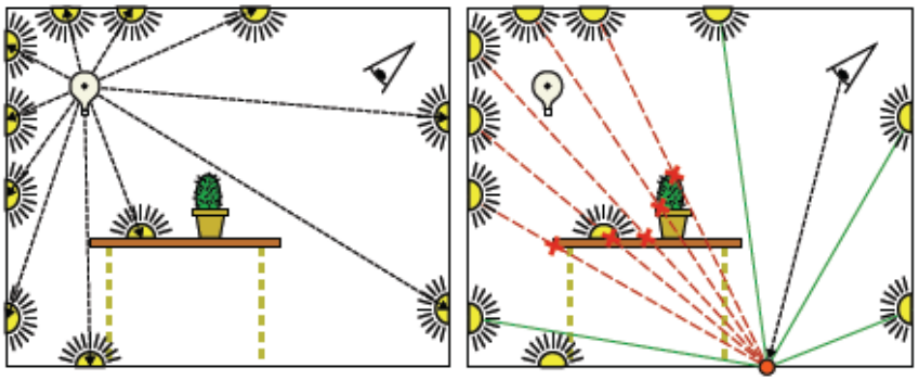


Obr. 2.3: Reflective Shadow Mapping – virtuálne svetelné zdroje x_p a x_q zodpovedajú pixelom p a q v shadow mape (zdroj: [2]).

zdrojom, nepriame svetlo sa teda vo výslednom obraze dotýka aj miest, ktoré by voči nemu mali byť zatienené, čo môže na diváka (pozorovateľa) pôsobiť rušivo.

Instant Radiosity

Algoritmus Instant Radiosity pristupuje k problému výpočtu nepriameho osvetlenia veľmi podobne, na rozdiel od RSM však pre generovanie virtuálnych svetelných zdrojov nevyužíva obraz získaný z pohľadu kamery, ale vrhanie lúčov do scény z pozície jednotlivých priamych svetelných zdrojov. Na základe vlastností priameho svetla je určená ich hustota a smer a lúče sú do scény následne vrhané s počiatkom na mieste primárneho svetelného zdroja. Priesečník lúču s geometriou scény potom určuje pozíciu sekundárneho svetelného zdroja, jeho orientácia, farba a ďalšie vlastnosti sú následne získané z materiálových a geometrických vlastností zasiahnutého objektu na mieste kontaktu s lúčom. Samotný svetelný (farebný) príspevok je započítaný, rovnako ako pri RSM, pri deferred výpočte osvetlenia v screen space nad obrazom, resp. popisom scény získaným z pohľadu kamery.



Obr. 2.4: Instant Radiosity (zdroj: [5]).

Podobne ako RSM, ani Instant Radiosity nijakým spôsobom priamo nerieši viditeľnosť voči sekundárnym svetelným zdrojom. Táto je riešená (ak riešená je) štandardnými metódami k tomu určenými, teda väčšinou pomocou Shadow Mappingu, alebo Shadow Volumes.

Kapitola 3

Imperfect Shadow Mapping

Pre výpočet a zobrazenie presvedčivého nepriameho osvetlenia scény – jadro všetkých metód globálneho osvetlenia – je kritickým riešenie viditeľnosti vzhľadom k zdrojom odrazeného svetla, resp. výpočet nimi vrhaných tieňov. Väčšina súčasných algoritmov túto problematiku rieši buďto využitím presných shadow máp vo vysokom rozlíšení, resp. tieňových telies pre všetky virtuálne svetelné zdroje (napr. Instant Radiosity), prípadne v princípe nerieši vôbec (Reflective Shadow Mapping). Shadow mapy síce je možné pre statické scény predpočítať, prípadne za behu čiastočne prepočítať pri zmenách osvetlenia, kompletný prepočet tieňových máp alebo telies (shadow volumes) pre dynamické scény je však veľmi problematický pri snahe o zachovanie interaktivity.

Princíp využitia Imperfect Shadow Maps (ISM) pri výpočte nepriameho osvetlenia vychádza z predpokladu, že keďže sa na osvetlení každého jednotlivého bodu povrchu v scéne podieľa mnoho virtuálnych svetelných zdrojov, výpočet viditeľnosťi vzhľadom ku každému jednému z nich nieje nutné riešiť presne (napr. – ako bolo spomenuté – pomocou shadow volumes alebo presných shadow máp), ale je postačujúce ho hrubo aproximovať a vzniknuté nepresnosti sa „spriemerujú" pri finálnom započítaní jednotlivých príspevkov, pričom kvalita výsledného obrazu výrazne neutrpí.

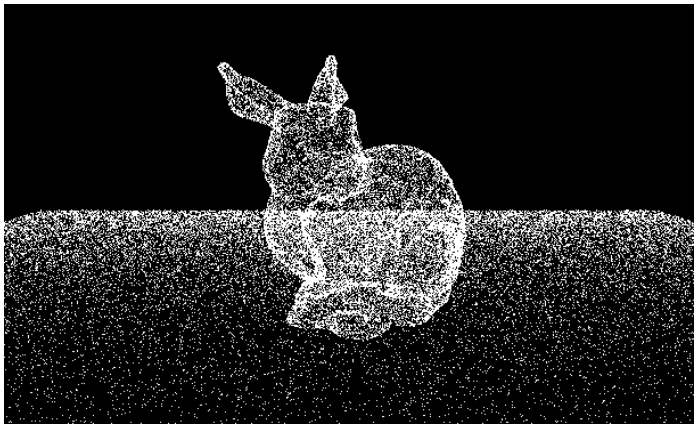
Základom pre popis procesu tvorby ISM v tejto kapitole je [11].

3.1 Predspracovanie scény

Keďže pre každý virtuálny svetelný zdroj postačí vytvoriť len približnú shadow mapu vo veľmi nízkom rozlíšení, je zbytočné na jej získanie spracúvať celú komplexnú geometriu scény. Pred samotnou tvorbou ISM preto scéna prejde predspracovacím krokom, v rámci ktorého jej polygonálnu reprezentáciu aproximujeme približne uniformne rozloženou množinou bodov.

Jednotlivé body generujeme náhodným výberom trojuholníka na ľubovoľnom objekte v scéne a následným náhodným výberom bodu na ňom. Pravdepodobnosť výberu jednotlivých trojuholníkov je váhovaná ich rozmermi, čím je zabezpečená približná rovnomernosť pokrytia geometrie takto tvorenými bodmi (väčšie, jednoduchšie plochy budú pokryté bodmi s porovnateľnou hustototu ako malé, komplexné objekty). Počet generovaných bodov je daný želaným počtom virtuálnych svetelných zdrojom (teda počtom ISM) a vhodne zvoleným počtom bodov priradených pre tvorbu ISM jednému VPL.

Pre každý zvolený bod je – okrem jeho (world-space) súradníc – uchovaný index trojuholníka, na povrchu ktorého sa nachádza a relatívne barycentrické súradnice. Týmto je



Obr. 3.1: Aplikácia predspracujúceho kroku na jednoduchú scénu.

dosiahnutá podpora dynamických scén, v ktorých môže dochádzať k deformácii či pohybu geometrie bez nutnosti prepočítavania jej bodovej reprezentácie. Vygenerovaná množina bodov je následne rozdelená do rovnomerne početných skupín prislúchajúcich jednotlivým VPL, pre tvorbu každej ISM je teda použitá len určitá podmnožina bodovej reprezentácie scény, čím je ďalej urýchlený výpočet. Pri zmenách v scéne sa potom dynamicky mení len táto distribúcia. V dôsledku toho je možné scénu predpripraviť offline a minimalizovať tak vplyv tohto kroku na rýchlosť a dynamiku jej zobrazovania. Uchovanie barycentrických súradníc tiež zabezpečuje jednoduchý prístup k normále objektu a odrazivosti materiálu v jednotlivých bodoch, nutné pri výpočte sekundárnych odrazov nepriameho osvetlenia (bližšie viď 4.1 – 14).

3.2 Tvorba ISMs

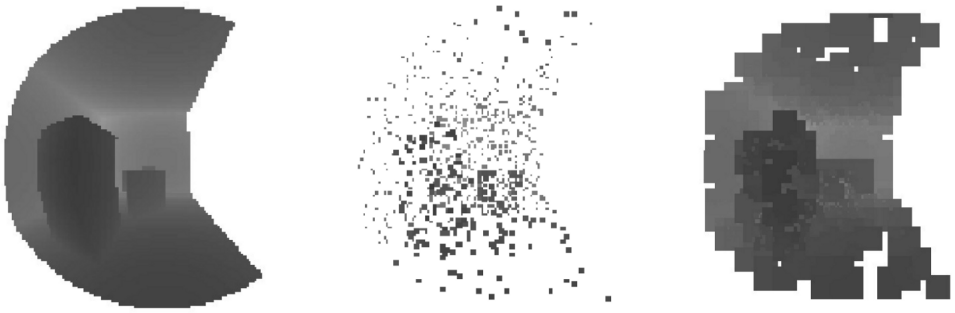
Samotná tvorba Imperfect Shadow Map prebieha „splattingom" jednotlivých bodov do depth bufferu, pričom veľkosť „splatu" je nepriamo úmerná druhej mocnine jeho vzdialenosti od zodpovedajúceho VPL. Keďže VPL predstavujú v princípe hemisferické svetelné zdroje, využívame pre zachytenie pohľadovej „pologule" paraboloidné shadow mapy [1].

V rámci jedného renderovacieho prechodu je takto vytvorených mnoho ISM uložených v rámci jednej veľkej textúry. Typicky sa jedná napr. o ISM s rozlíšením 128×128 uložených v textúre 4096×4096. Rovnomerná distribúcia vstupnej postupnosti bodov aproximujúcich geometriu scény medzi jednotlivé VPL je zabezpečená dynamicky priamo pri výpočte na GPU v rámci vertex shaderu.

Keďže pre výpočet každej ISM je použitá len obmedzená podmnožina už zjednodušenej reprezentácie scény, je veľmi pravdepodobné, že v takto vytvorenej depth mape sa bude nachádzať pomerne veľké množstvo oblastí bez priradenej zmysluplnej hodnoty aj na mieste skutočného výskytu objektov v scéne, tzv. dier. Tieto vypĺňame pull-push metódou.

3.3 Pull-Push algoritmus

Pre vyplenenie dier v ISM vzniknutých v dôsledku jej tvorby z obmedzenej podmnožiny pomerne malých bodov zjednodušene reprezentujúcich scénu je využitý dvojkrokový takzvaný Pull-Push algoritmus [7].



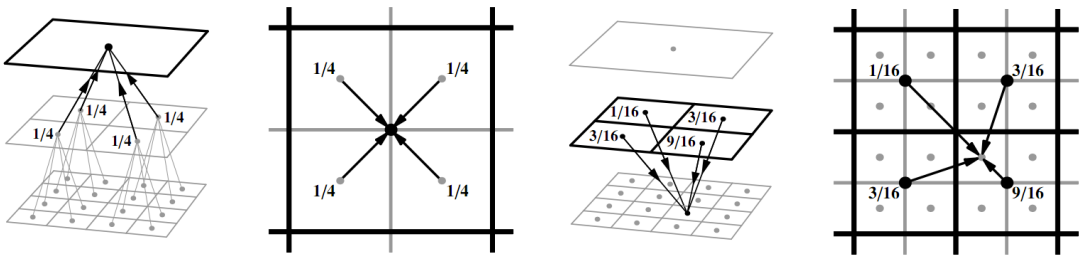
(a) Klasická Shadow Mapa (b) Imperfect Shadow Map (c) ISM po aplikácii pull-push

Obr. 3.2: Imperfect Shadow Maps (zdroj: [11])

V prvom – *Pull* – kroku je vytvorená pyramidálne usporiadaná štruktúra, obsahujúca na každej úrovni spracúvanú depth mapu v polovičnom rozlíšení oproti predošlej, podobne ako pri mip-mappingu. Hodnoty na hrubšej úrovni určíme priemerovaním hodnôt štyroch pixelov presnejšej úrovne, do úvahy však samozrejme berieme len príspevky tých pixelov, ktoré sú označené ako valídne, teda už na jemnejšej úrovni obsahujú použiteľnú hĺbkovú informáciu. V prípade, že sú všetky štyri príspevky nevalídne, je aj výsledný pixel na hrubšej úrovni nevalídny.

Následne v druhej – *Push* – fáze postupujeme zhora (teda od hrubších úrovní pyramidálnej štruktúry vytvorenej v predchádzajúcom kroku) nadol. Interpolujeme v tejto fáze iba chýbajúce nevalídne hodnoty na jemnejších úrovniach. Hodnota na jemnejšej úrovni je vypočítaná na základe hodnôt štyroch pixelov vyššej úrovne. Príspevky jednotlivých pixelov sú váhované korešpondujúc s bikvadratickým B-spline delením rovnomernej štvorhrannej mriežky (viď obr. 3.3). Do výpočtu sú analogicky k pull fáze zahrnuté len valídne pixely hrubšej úrovne, ich váhy teda musia byť normalizované (váha každého príspevku je predelená súčtom váh valídnych pixelov hrubšej úrovne).

V oboch fázach je možné podľa [7] uplatniť tzv. outlier rejection - na základe vopred stanoveného prahu (určeného separátne pre pull a push fázu), škálovaného podľa aktuálne spracúvanej úrovne „mip-mapy" sú v pull fáze kombinované len blízke hodnoty a naopak v push fáze sú nahrádzané len hodnoty dostatočne vzdialené od relevantných hodnôt hrubšej úrovne.



(a) Pull fáza (b) Push fáza

Obr. 3.3: Pull-Push algoritmus [7]

Pre vyplnenie prakticky všetkých „dier" v textúre, resp. dvojdimenzionálnom poli sa ukazuje ako postačujúce postúpiť z najjemnejšej – teda vypĺňanej – úrovne len o dve úrovne vyššie.

Kapitola 4

Efektívny výpočet nepriameho osvetlenia s využitím ISM

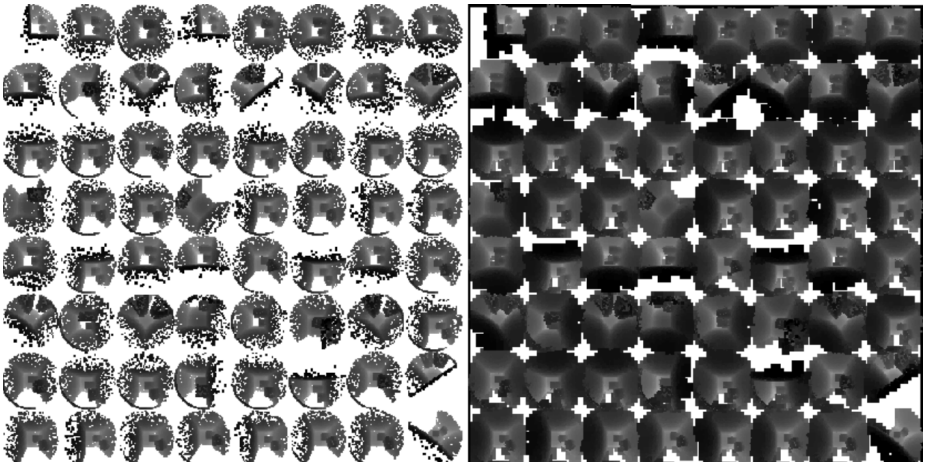
Výpočet nepriameho osvetlenia popisovanými metódami v princípe vychádza z myšlienky Instant Radiosity [5], teda stochastickej distribúcie virtuálnych svetelných zdrojoch na priamo osvetlených povrchoch, ktoré následne fungujú ako zdroje nepriameho osvetlenia. Následne pri zohľadnení viditeľnosti a zatienenia dôjde k spočítaniu príspevkov jednotlivých VPL a tým k určeniu nepriameho osvetlenia povrchov v scéne. Pre efektívne riešenie viditeľnosti a nepriamych tieňov zabezpečujúce inetraktivitu scény je využitá práve metóda Imperfect Shadow Mappingu

4.1 Algoritmus

Pre generovanie VPL v scéne na základe jej osvetlenia primárnym svetelným zdrojom využívame Reflective shadow mapping [2]. Pozíciu jednotlivých VPL určíme samplovaním takto vytvorenej mapy, buďto zohľadňujúc jas jednotlivých osvetlených povrchov (*importance sampling*), alebo volíme (pseudo)náhodne, napríklad s využitím Haltonovej sekvencie [4].

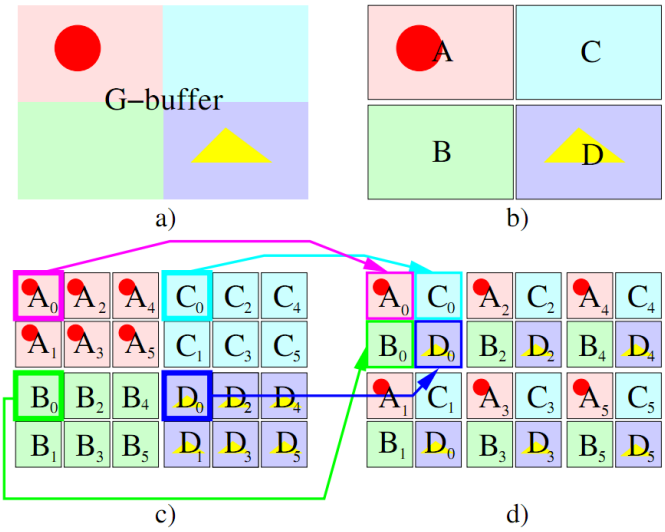
V ďalšom kroku potom na základe predpripravenej bodovej aproximácie scény (viď 3.1 – 9) vytvoríme ISM pre každý virtuálny svetelný zdroj a tieto uchovávame v jednej veľkej hĺbkovej mape. Pre určenie pozície jednotlivých bodov v sub-textúre pritom používame paraboloidnú projekciu. Vytvorenú shadow textúru následne spracujeme, zameriavajúc sa najmä na odstránenie dier v ISM vzniknutých v dôsledku zjednodušenia a rozdelenia geometrie použitej pri ich tvorbe, pull-push metódou. Metóda tvorby ISM, ako aj ich následná úprava technikou pull-push je bližšie popísaná v 3.2 – 10.

Osvetlenie je získané v screen space, nad obrazom získaným z pohľadu kamery, pomocou tzv. deferred shadingu, umožňujúc efektívne zahrnúť príspevky veľkého počtu svetelných zdrojov, ako primárnych, tak nagenerovaných virtuálnych. Pre výpočet nepriameho osvetlenia jednotlivých zobrazovaných pixelov nepoužívame celú množinu virtuálnych svetelných zdrojov, ale využívame techniku tzv. *Interleaved G-bufferu* [12]. G-buffer, do ktorého renderujeme z pohľadu kamery, rozdelíme na „dlaždice" pevnej veľkosti (napr. 4×4 pixely). Následne jednotlivé pixely preskladáme do súvislých úsekov (v tomto prípade 16 úsekov), z ktorých každý obsahuje jeden pixel z každej dlaždice. Nad týmito úsekmi potom spočítame nepriame osvetlenie z určitej podmnožiny virtuálnych svetelných zdrojov a buffer preskladáme do pôvodnej podoby. Týmto v oblasti jednej dlaždice získame príspevok všetkých



Obr. 4.1: Aplikácia Pull-Push algoritmu pri generovaní Imperfect Shadow Maps [11].

VPL, zatiaľčo každý pixel túto dlaždicu tvoriaci je počítaný len z určitej ich podmnožiny (pre lepšiu predstavu viď obr.4.2 str.13). Osvetlenie susediacich pixelov sa takto môže viditeľne líšiť, preto je pre odstránenie vzniknutého šumu vo výslednom obraze nutné aplikovať ešte jeden krok, a to gaussovské filtrovanie s prihliadaním na geometriu scény (edge-aware, resp. geometry-aware blur).



Obr. 4.2: *Interleaved G-Buffer.* (a) reprezentuje G-buffer pred delením. (b) G-buffer je rozdelený na 4 (2×2) bloky. Rozmer prekladu je 3×2. (c) zobrazuje G-buffer po rozdlení. Každý blok bol rozdelený na 6 (3×2) sub-blokov. (d) Výsledné sub-buffere sú rozložené v celom pôvodnom bufferi [12].

Finálny obraz získame vykreslením štvoruholníka prekrývajúceho viewport a zarovnaného voči kamere, s textúrou vyfiltrovaného obrazu kombinujúceho zložky priameho aj nepriameho osvetlenia.

Multi-bounce

V prípade, že chceme scénu renderovať s viac ako jedným odrazom nepriameho osvetlenia, je nutné nahradiť ISM Imperfect Reflective Shadow Mapami (*IRSM*), podobne, ako by sme postupovali pri nahrádzaní klasickej shadow mapy jej reflective verziou. VPL pre druhý odraz generujeme renderovaním bodovej reprezentácie do IRSM pre každý z počiatočných VPL. Takto vytvorenú veľkú RSM, podobne ako pri ISM sa skladajúcu z jednotlivých máp v nízkom rozlíšení, následne samplujeme. Pozícia a normála generovaných svetelných zdrojov pre druhý odraz je potom získaná z bodovej reprezentácie vďaka uchovaným barycentrickým súradniciam jednotlivých bodov. Pri výpočte samotného nepriameho osvetlenia postupujeme potom prakticky zhodne s postupom pre jediný odraz. Rovnaký princíp uplatňujeme pre každý prípadný ďalší odraz.

4.2 Ďalšie možnosti využitia ISM

Výpočet nepriameho osvetlenia plne dynamických scén nieje jediným možným využitím techniky Imperfect Shadow Mappingu. Ako ďalšie možné využitia uvádzajú autori priame osvetlenie scény plošnými svetelnými zdrojmi – teda zdrojmi produkujúcimi mäkké tiene – alebo environment mapou (farebne nehomogénne svetelné zdroje)[11].

Plošné svetelné zdroje

Metódou ISM je možné tiež využiť pri zobrazovaní mäkkých tieňov vzniknutých osvetlením scény plošným svetlom (*Area Light*). Vyžarujúcu plochu môžeme aproximovať pomocou množiny VPL a tieň vrhaný objektmi v scéne vzhľadom na každé z nich potom prostredníctvom ISM. Napriek nepresnosti jednotlivých ISM sa tu, podobne ako pri nepriamom osvetlení, môžeme spoľahnúť na „spriemerovanie" chyby pri skladaní príspevkov od jednotlivých virtuálnych svetelných zdrojov. Podobne ako pri výpočte nepriameho osvetlenia, aj tu je možné pre ďalšie zefektívnenie výpočtu využiť prekladaný G-buffer. Naproti metódam špecificky sa zameriavajúcim na prácu s obdĺžnikovými svetelnými zdrojmi je metódou ISM možné aproximovať aj plošné osvetlenie dané zložitejšou geometriou.

Environment maps

Podobne ako v predchádzajúcich príkladoch možno postupovať aj pri priamom osvetlení scény environment mapou. Zložitú distribúciu farby a intenzity environment mapy ako svetelného zdroja aproximujeme množinou VPL a ISM sú opäť využiteľné pre výpočet tieňov a viditeľnosti.

Kapitola 5

Popis implementácie techniky ISM

Funkčnosť a vlastnosti spracúvaných postupov sú demonštrované jednoduchou aplikáciou. Ako implementačný jazyk som zvolil C++ s využitím knižnice OpenGL (vo verzii 3.3) a jazyka GLSL (OpenGL Shading Language – verzia 330) pre realizáciu shaderov, umožňujúce využiť vlastnosti moderného grafického HW nutné pre implementáciu techniky ISM, ako je existencia programovateľnej grafickej pipeline (vertex a fragment shader processing), využitie Multiple Render Targets a vôbec offscreen renderingu a mnohé ďalšie.

Pre vytvorenie a nastavenie grafického kontextu, okna aplikácie, spracovanie užívateľských vstupov z myši a klávesnice a ovládanie demonštračnej aplikácie s dôrazom na jej prenositeľnosť a jednoduchosť, ako aj meranie časov jednotlivých krokov algoritmu a ďalšiu pomocnú funkcionalitu, som sa rozhodol využiť knižnicu SFML vo verzii 2.0 (Simple and Fast Multimedia Library – domovská stránka `http://www.sfml-dev.org`).

Pre načítanie modelov a ich súčastí, ktoré boli využité pri testovaní a ďalej slúžia ako ukážky výsledkov, bola použitá knižnica Assimp verzia 3.0 (Open Asset Import Library – domovská stránka `http://assimp.sourceforge.net`). Zdrojom samotných modelov je McGuire Graphics Data (`http://graphics.cs.williams.edu/data/meshes.xml`). Všetky použité modely a súčasti sú voľne dostupné a šíriteľné. Pre načítanie a spracovanie textúr je použitá knižnica DevIL (`http://openil.sourceforge.net`).

Nasleduje stručný náčrt štruktúry aplikácie s bližším zameraním sa na súčasti, ktoré sú ťažiskové z pohľadu implementovaného algoritmu pre výpočet osvetlenia v scéne (teda samotný algoritmus tvorby a použitia ISM pre výpočet nepriameho osvetlenia so zohľadnením viditeľnosti).

5.1 Preprocess

Po spustení aplikácie a inicializácii využívaných knižníc a prostriedkov (vytvorenie renderovacieho kontextu,kompilácia shaderov, tvorba Framebuffer objektov, . . .) je nutné vykonať dva dôležité úkony ešte v rámci preprocessing kroku – teda pred samotným začatím zobrazovania. Prvým je samozrejme vytvorenie bodovej reprezentácie zobrazovanej scény, druhým potom zakódovanie Haltonovej postuponosti do textúry pre potreby vzorkovania RSM a teda generovania virtuálnych svetelných zdrojov.

Načítanie modelov pomocou knižnice Assimp

Knižnica Assimp umožňuje jednoduché importovanie 3D modelov zo širokej škály bežných formátov uchovania 3D grafických dát. Množina podporovaných formátov zahŕňa všetky

najčastejšie používané, ako napr. Collada (*.dae*), Blender 3D (*.blend*), 3ds Max (*.3ds*, resp. *.ase*), Wavefront (*.obj*), či directX (*.X*), ako aj množstvo menej rozšírených. Umožňuje jednoduché načítanie kompletnej štruktúry (často hierarchickej), zahŕňajúcej jednotlivé objekty (meshe), tvoriace objekt, resp. grafickú scénu. Zároveň umožňuje prístup k materiálovým vlastnostiam objektov (napr. automatické spracovanie *.mtl* súboru prislúchajúceho k Wavefront modelu).

Základným rozhraním pre prístup k funkcionalite knižnice Assimp je trieda `Importer`. Samotný model je po úspešnom spracovaní vstupného súboru jej metódou `ReadFile` reprezentovaný hierarchickou stromovou dátovou štruktúrou `aiScene`. Metóda `ReadFile` tiež umožňuje špecifikovať požiadavky na predspracovanie modelu (ako je napr. dopočítanie binormálových a tangentových vektorov a podobne). V prípade úspešného načítania a spracovania obsahuje výsledný objekt aiScene informácie ako o samotnej geometrii objektov (meshe), prípadne o materiáloch, tak (v prípade komplexnejších 3D grafických scén) aj o svetlách, kamerách, či animáciách.

Pre potreby zobrazenia prostredníctvom OpenGL je dátami predstavujúcimi informácie o modeloch obsiahnutých v objekte typu `aiScene` nutné naplniť OpenGL štruktúry k tomu určené, teda Vertex Buffer Object (*VBO*) pre samotné informácie o vrcholoch a Element Array Object (*EBO*) pre ich indexáciu a priradenie jednotlivým trojuholníkom tvoriacim model. Spolu súvisiace VBO, EBO a nastavenia atribútov jednotlivých vrcholov, resp. popis ich konkrétneho uloženia v rámci VBO, zastrešuje tzv. Vertex Array Object (*VAO*).

Pre každý jednotlivý mesh je najskôr vytvorená štruktúra, obsahujúca počet indexov (prvkov EBO) a prvý index v poli indexov, ako aj v poli vrcholov, prislúchajúci danému meshu. Následne sú mu priradené prípadné textúry, charakterizujúce difúznu zložku jeho materiálu. Jednotlivé trojuholníky tvoriace meshe, teda trojice vrcholov, sú následne serializované, uchováme si ich pozície, normály a textúrovacie koordináty (teda vlastnosti, ktoré budú následne využité pri tvorbe G-bufferu ako z pohľadu svetla, tak aj kamery a napokon pri deferred výpočte osvetlenia) a týmito sú naplnené VBO a EBO, tvoriace načítaný model. Keďže v medzikroku uchované dáta obsahujú všetky informácie potrebné k vytvoreniu bodovej reprezentácie scény, použijeme ich aj k tomuto účelu. Zjednodušená ukážka importu v C++:

```
void Model::import(const std::string & filename)
{
  //create and bind VAO
  glGenVertexArrays(1, &mainVAO);
  glBindVertexArray(mainVAO);
  //generate index/vertex buffers
  glGenBuffers(NUM_BUFFERS, buffers); //positions, normals, texCoords, EBO

  Assimp::Importer importer;
  //load scene
  const aiScene* scene = importer.ReadFile(filename.c_str());

  if(scene) //on success
  {
    loadTextures(scene);
    fromScene(scene, filename);
  }else handleError();

  //unbind VAO
  glBindVertexArray(0);
}
```

Prevedenie importovanej scény/modelu, zahŕňajúce postupný prechod jednotlivými meshmi

a konverziu dát z formátu používaného knižnicou Assimp do podoby zobraziteľnej pomocou OpenGL je obstarané v uvedenej ukážke kódu funkciou `fromScene()`. Jej obsah (samotné spracovanie modelu) potom zjednodušene:

```
void Model::fromScene(const aiScene* scene, const std::string & filename)
{
  unsigned int numVertices = 0;
  unsigned int numIndices = 0;
  //get basic info about meshes
  for(unsigned int i = 0; i < scene->mNumMeshes; i++)
  {
    info.numIndices = scene->mMeshes[i]->mNumFaces * 3;
    info.baseVertex = numVertices;
    info.baseIndex = numIndices;
    meshInfos.push_back(info);

    numVertices += scene->mMeshes[i]->mNumVertices;
    numIndices += info.numIndices;
  }

  std::vector<Vector3f> positions, normals;
  std::vector<Vector2f> texCoords;
  std::vector<unsigned int> indices;
  //fill in the data
  for(unsigned int i = 0; i < meshInfos.size(); i++)
  {
    const aiMesh* mesh = scene->mMeshes[i];
    //get material info
    struct aiMaterial *mtl = scene->mMaterials[mesh->mMaterialIndex];
    if(mtl->GetTexture(aiTextureType_DIFFUSE, 0, &texPath) == AI_SUCCESS)
    {
      meshInfos[i].texture = textureMap[file+texPath.data];
    }
    initMesh(mesh, positions, normals, texCoords, indices);
  }
  //generate point representation
  generatePointCloud();
  //fill the buffers
  glBindBuffer(GL_ARRAY_BUFFER, buffers[VBO_POS]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(positions[0])*positions.size(),
        &positions[0], GL_STATIC_DRAW);
  glEnableVertexAttribArray(0); //enable position attribute as VS input
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
  //do the same for the rest
  // ...
}
```

Metóda `initMesh` pritom len naplní jednotlivé položky vektorov `positions`, `normals`, `texCoords` a `indices` hodnotami získanými z vrcholov daného meshu (uchovaných v poli `mVertices` objektu typu `aiMesh`).

Načítanie textúr pomocou knižnice devIL a vykreslenie

Pre načítanie a sprístupnenie textúr, ako aj ich konverziu do formátu použiteľného pre zobrazenie pomocou OpenGL je použitá knižnica DevIL (*The Developer's Image Library*). Pre užívateľa navyknutého na pomenovávacie konvencie OpenGL, ako aj spôsob práce s touto grafickou knižnicou, je použitie DevIL mimoriadne pohodlné a intuitívne (knižnica používa rovnakú „*naming convention*", často len nahrádzajúc typickú predponu gl- funkcií knižnice OpenGL predponou il- a pod.).

Inicializácia knižnice devIL prebieha jednoduchým volaním funkcie `ilInit()`. Následne je možné z poľa `mMaterials`, obsiahnutého v objekte typu `aiScene`, načítanom zo súboru

pomocou knižnice Assimp, získať informácie o materiálových vlastnostiach objektu. Pre naše účely zaujímavá je existenica, typ a cesta k súboru obsahujúcemu difúznu textúru. Pre každú nájdenú difúznu textúru uchováme cestu k nej ako kľúč v asociatívnom poli typu `std::map<std::string,GLuint>`. Následne je pre každú položku takto vytvoreného asociatívneho poľa vygenerovaný OpenGL texture object (ktorého identifikátor je uchovaný ako hodnota v spomínanom objekte typu std::map), pre ktorý je načítaný (pomocou `ilLoadImage()`) a konvertovaný do použiteľného formátu (`ilConvertImage()`) zodpovedajúci obrázok a vygenerovaná textúra (volaním OpenGL funkcie `glTexImage2D()`).

Pri volaní metódy starajúcej sa o vykreslenie modelu je potom postupne prejdený celý zoznam štruktúr obsahujúcich informácie o jednotlivých meshoch a na základe informácií v nich obsiahnutých sú aktivované zodpovedajúce textúry a vykreslené zodpovedajúce časti VBOs popisujúcich objekt. Zjednodušené vykreslenie modelu (C++):

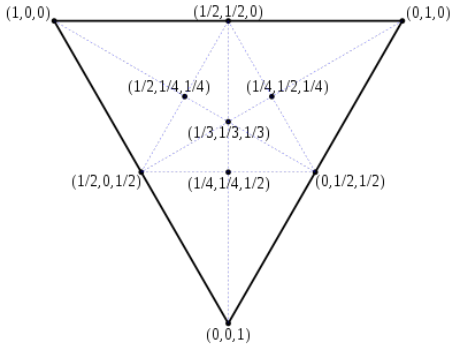
```
void Model::draw()
{
  //bind main VAO
  glBindVertexArray(mainVAO);
  //draw meshes
  for(unsigned int i = 0; i < meshInfos.size(); i++)
  {
    //bind diffuse texture
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, meshInfos[i].texture);
    //draw corresponding elements
    glDrawElementsBaseVertex(GL_TRIANGLES, meshInfos[i].numIndices,
                GL_UNSIGNED_INT,
                (void*)(sizeof(unsigned int) * meshInfos[i].baseIndex),
                meshInfos[i].baseVertex);
  }
  //unbind
  glBindVertexArray(0);
}
```

5.1.1 Bodová reprezentácia scény

Pre prevod trojuholníkovej reprezentácie scény na množinu bodov je nutné trojuholníky všetkých modelov najskôr spracovať, teda určiť ich veľkosť a označiť príslušnosť k modelu. K tomuto účelu využijeme vektory obsahujúce pozície vrcholov a indexy, získané pri načítaní modelov knižnicou Assimp. Pre potreby rovnomerného vzorkovania celej scény so zohľadnením veľkosti trojuholníkov (ako bolo naznačené v 3.1 – str.9) sú takto spracované trojuholníky následne zoradené podľa veľkosti. Samotné radenie, ktoré je vykonané len jednorázovo pred započatím hlavného zobrazovacieho cyklu aplikácie, je vykonané priamo pri spracúvaní jednotlivých trojuholníkov algoritmom insert sort. Zoradené pole trojuholníkov je rozdelené do skupín pokrývajúcich aspoň určité percento celkového povrchu scény. Napríklad dva veľké trojuholníky, tvoriace podlahu scény, na ktorej je umiestnený menší, zložitejší objekt, tvoriace 30% celkového povrchu, budú každý tvoriť samostatnú skupinu, niekoľko najväčších trojuholníkov menšieho objektu ďalšiu atď. V rámci skupiny má každý trojuholník rovnakú pravdepodobnosť byť zasiahnutým pri výbere bodu.

Následne vzorkujeme body postupným cyklickým výberom z jednotlivých skupín (systémom round-robin), kým nedosiahneme želaný počet. Ten je daný zvoleným počtom virtuálnych svetelných zdrojov vynásobeným počtom bodov na jeden VPL pre tvorbu ISM. Bod sa vyberá náhodným zvolením trojuholníka v skupine a vygenerovaním náhodných na ňom ležiacich súradníc.

Pre zjednodušenie prepočtov pri zobrazovaní dynamicky sa meniacej, alebo animovanej geometrie sú každému bodu okrem 3D súradníc v priestore modelu spočítané a uschované ešte barycentrické súradnice v rámci trojuholníka, na ktorého povrchu sa generovaný bod nachádza.



Obr. 5.1: Barycentrické súradnice v trojuholníku (zdroj: Wikipedia).

Barycentrické súradnice bodu $\mathbf{P}$, ležiaceho v trojuholníku danom vrcholmi $\mathbf{A}, \mathbf{B}, \mathbf{C}$, definujeme vzťahom

$$\mathbf{P} = s\mathbf{A} + r\mathbf{B} + t\mathbf{C}, \quad s + r + t = 1$$

úpravou potom

$$\mathbf{P} = s\mathbf{A} + r\mathbf{B} + (1 - s - r)\mathbf{C},$$

resp.

$$s\mathbf{A} + r\mathbf{B} + \mathbf{C} - s\mathbf{C} - r\mathbf{C} - \mathbf{P} = 0,$$

čím v praxi dospejeme k sústave rovníc

$$s(A.x - C.x) + r(B.x - C.x) + C.x - P.x = 0$$

$$s(A.y - C.y) + r(B.y - C.y) + C.y - P.y = 0$$

$$s(A.z - C.z) + r(B.z - C.z) + C.z - P.z = 0,$$

ktorej riešením sú barycentrické súradnice (s, r, t) vygenerovaného bodu $\mathbf{P}$ vzhľadom k vrcholom trojuholníka $\mathbf{A}, \mathbf{B}, \mathbf{C}$.

V prípade animácie, posunu, alebo zmeny geometrie modelu je možné vďaka uschovaným barycentrickým koordinátom súradnice navzorkovaných bodov, tvoriacich point cloud reprezentujúci daný model, jednoducho prepočítať dosadením nových súradníc vrcholov trojuholníka do pôvodného vzťahu. Tým je zabránené zložitému prevzorkovávaniu celého modelu, resp. scény.

Navzorkovaný bod je uchovaný, pričom zaznamenávame jeho súradnice (pre priamu manipuláciu), barycentrické súradnice (pre prípadné zmeny geometrie) a index trojuholníka, z ktorého pochádza.

Obe reprezentácie geometrie scény – pôvodnú trojuholníkovú aj vytvorenú bodovú – je následne možné pripraviť na rendering (vytvorenie VBO, EBO, VAO,...).

5.1.2 Generovanie pozícií virtuálnych svetelných zdrojov

Pozície virtuálnych svetelných zdrojov v scéne určíme pseudonáhodným vzorkovaním RSM vytvorenej pre reprezentáciu priameho osvetlenia scény. Jednoduché pseudonáhodné, kváziuniformné pokrytie dvojdimenzionálnej textúry pevne daným počtom bodov (v našom prípade zvoleným počtom virtuálnych svetelných zdrojov) dosiahneme s využitím dvoch Haltonových postupností z rôznym základom. Základom oboch postupností sú prvočísla, volíme napríklad 2 a 3. Sekvenciu tvoríme postupným delením vhodne zvoleného intervalu mocninami základu [4]. Keďže účelom je generovať súradnice do 2D textúry, delený interval určíme na (0,1). Výsledné postupnosti potom vyzerajú nasledovne.

Pre základ 2

$$^1/_2,\ ^1/_4,\ ^3/_4,\ ^1/_8,\ ^5/_8,\ ^3/_8,\ ^7/_8,\ ^1/_{16},\ \ldots$$

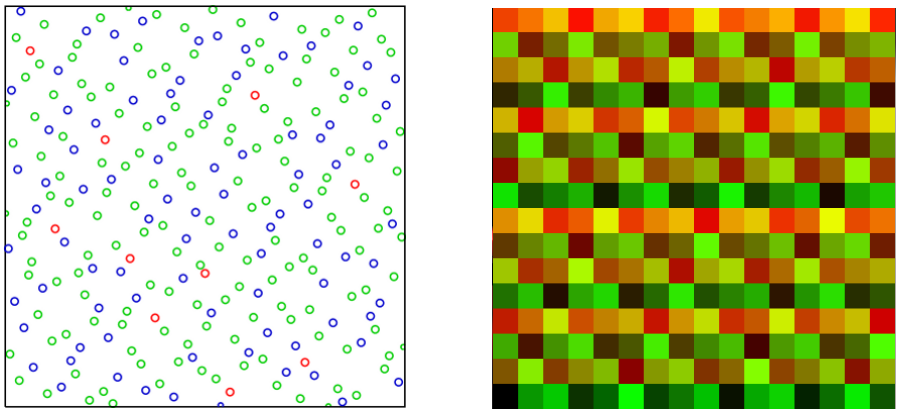
Pre základ 3

$$^1/_3,\ ^2/_3,\ ^1/_9,\ ^4/_9,\ ^7/_9,\ ^2/_9,\ ^5/_9,\ ^8/_9,\ \ldots$$

Ich skombinovaním potom dostávame postupnosť bodov v dvojdimenzionálnom priestore

$$(^1/_2,^1/_3),\ (^1/_4,^2/_3),\ (^3/_4,^1/_9),\ (^1/_8,^4/_9),\ (^5/_8,^7/_9),\ (^3/_8,^2/_9),\ (^7/_8,^5/_9),\ (^1/_{16},^8/_9),\ \ldots$$

Takto vygenerovanú postupnosť je možné pre jednoduché využitie v aplikácii kódovať ako dvojkanálovú RG textúru, kde R a G zložky predstavujú x a y súradnice do RSM, s ktorou pracujeme. Takto vytvorená textúra je veľmi malá, napr. pre 1024 virtuálnych svetelných zdrojov postačuje rozlíšenie 32×32. Atribúty VPL sú potom získané vzorkovaním týchto miest v jednotlivých zložkách RSM.



(a) Halton2,3 (b) Reprezentácia textúrou

Obr. 5.2: (a) predstavuje prvých 256 bodov Haltonovej postupnosti s dvoma základmi (2 a 3). (b) zobrazuje textúru kódujúcu súradnice týchto bodov v R a G kanáloch.

Textúra so zakódovanou Haltonovou sekvenciou, spoločne s hodnotou vzorkovacieho offsetu určujúceho virtuálny svetelný zdroj pre určitú množinu bodov, je potom vstupom pre kroky tvorby a aplikácie ISM v rámci hlavného cyklu aplikácie.

5.2 Hlavný cyklus aplikácie

Po ukončení predprípravy scény a vygenerovaní textúry kódujúcej Haltonovu sekvenciu pre zvolený počet virtuálnych svetelných zdrojov môže aplikácia pokročiť k vykonávaniu svojej hlavnej funkčnosti. Okrem reakcie na užívateľské vstupy a obsluhy iných udalostí táto v prípade, že došlo k zmene (pohol sa primárny svetelný zdroj, došlo k zmene pohľadu kamery, pohybu objektov, ...), pozostáva z nasledujúcich zobrazovacích krokov.

Realizácia MRT v prostredí OpenGL

Pre rendering do užívateľsky definovaného framebufferu, teda bez zásahov do hlavného framebufferu zobrazeného na obrazovke (obyčajne hlavné okno aplikácie) slúžia v prostredí OpenGL tzv Framebuffer Objecty (*FBO*). Každý FBO sa skladá z určitej množiny obrazov, kde každý z nich reprezentuje miesto vo FBO, ku ktorému môže byť pripojený obrazový buffer (realizovaný textúrou, alebo Renderbuffer Objectom). Typy pripojiteľných obrazov (*attachments*) sú

- `GL_COLOR_ATTACHMENTi` – implementačne závislý počet bufferov pre rendering farebnej informácie (teda pripojenie obrazov s formátom vyhovujúcim pre zápis farebnej informácie). Minimálny počet (garantovaný špecifikáciou knižnice OpenGL) je 1 (vždy je tak k dispozícii minimálne `GL_COLOR_ATTACHMENT0`), maximálny je potom daný konštantou `GL_MAX_COLOR_ATTACHMENTS`.
- `GL_DEPTH_ATTACHMENT` – umožňuje pripojenie obrazov vyhovujúcich hĺbkovým formátom. Takto pripojený obraz sa stáva depth bufferom uvažovaného FBO.
- `GL_STENCIL_ATTACHMENT` – umožňuje pripojenie obrazov vyhovujúcich stencil formátom. Takto pripojený obraz sa stáva stencil bufferom uvažovaného FBO.
- `GL_DEPTH_STENCIL_ATTACHMENT` – pripojený obraz reprezentuje zároveň ako depth, tak aj stencil buffer.

Pre úspešné vytvorenie a použiteľnosť FBO je nutné priradiť mu aspoň jeden farebný a práve jeden hĺbkový attachment.

Podobne ako pri iných typoch OpenGL objektov (Textúra, Renderbuffer, ...) slúži pre získanie prístupového identifikátoru funkcia `glGenFramebuffers()`. Pre indikáciu práve aktívneho (*bind*) FBO slúži potom štandardne funkcia `glBindFramebuffer()`, ktorej prvý argument špecifikuje mód, v ktorom s daným FBO mienime pracovať. Tento môže nadobúdať jednu z hodnôt `GL_FRAMEBUFFER`, `GL_READ_FRAMEBUFFER`, alebo `GL_DRAW_FRAMEBUFFER`. Proces tvorby a prípravy Framebuffer Objectu určeného pre offscreen rendering budem ilustrovať na zjednodušenom príklade tvorby FBO pre Reflective Shadow Mapping. Jednoduchá ukážka vytvorenia FBO potom v C++:

```
GLuint RSMFBO;
glGenFramebuffers(1, &RSMFBO);
glBindFramebuffer(GL_FRAMEBUFFER, RSMFBO);
```

Následne je možné pristúpiť k vytvoreniu samotných farebných textúr, ktoré budú slúžiť ako cieľ renderingu a uchovávať užívateľské informácie – v prípade tvorby RSM teda pozície, normály a flux (resp. difúznu farbu objektov).

Zjednodušene (C++):

```
//world space coordinates
glGenTextures(1, &wscTexture);
glBindTexture(GL_TEXTURE_2D, wscTexture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, SHADOW_WIDTH, SHADOW_HEIGHT, 0,
             GL_RGBA, GL_FLOAT, NULL);
//normals
glGenTextures(1, &normTexture);
glBindTexture(GL_TEXTURE_2D, normTexture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, SHADOW_WIDTH, SHADOW_HEIGHT, 0,
             GL_RGBA, GL_FLOAT, NULL);
//flux
glGenTextures(1, &fluxTexture);
glBindTexture(GL_TEXTURE_2D, fluxTexture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, SHADOW_WIDTH, SHADOW_HEIGHT, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, NULL);
```

Keďže RSM plánujeme použiť aj pri výpočte viditeľnosti pri priamom osvetlení a k tomuto účelu využiť HW prostriedky, hĺbkovému attachmentu sú naviac nastavené parametre porovnania.

```
//depth component
glGenTextures(1, &depthTexture);
glBindTexture(GL_TEXTURE_2D, depthTexture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,GL_COMPARE_REF_TO_TEXTURE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LESS);
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24,
             SHADOW_WIDTH, SHADOW_HEIGHT, 0,
             GL_DEPTH_COMPONENT, GL_FLOAT, 0);
```

Vytvorené textúry následne pripojíme k FBO

```
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
                       depthTexture, 0);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
                       wscTexture, 0);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT1, GL_TEXTURE_2D,
                       normTexture, 0);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT2, GL_TEXTURE_2D,
                       fluxTexture, 0);
```

Posledným krokom je špecifikácia množiny attachmentov, ktoré budú použité pri renderingu, čím je vlastne umožnené využitie viacerých cieľov renderingu súčasne. Poradie, v ktorom sú jednotlivé textúry pripojené k FBO by malo logicky zodpovedať layoutu fragment shaderu, realizujúceho zápis hodnôt. Depth buffer nieje nutné uvádzať, hĺbkové hodnoty sú mu priradené automaticky.

```
GLenum MRT[] = {
  GL_COLOR_ATTACHMENT0,   //wsc location0
  GL_COLOR_ATTACHMENT1,   //normal location1
  GL_COLOR_ATTACHMENT2    //flux location2
};
glDrawBuffers(3, MRT);
```

Funkcia `glCheckFramebufferStatus()` slúži pre overenie úspešného vytvorenia Framebuffer Objectu. Táto úspech, teda korektnosť a použiteľnosť, vytváraného FBO indikuje vrátením hodnoty `GL_FRAMEBUFFER_COMPLETE`.

Rovnakým spôsobom sú vytvorené FBO pre všetky kroky algoritmu, realizujúce offscreen rendering.

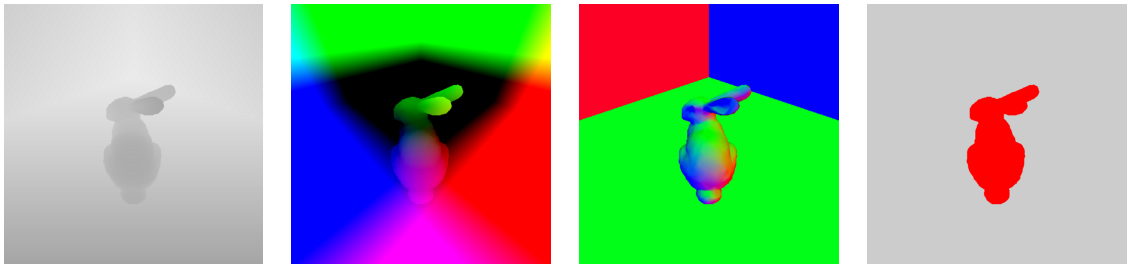
5.2.1 Generovanie RSM

Ako prvá je vytvorená Reflective Shadow Mapa, slúžiaca ako pre výpočet viditeľnosti, resp. tieňov vrhaných primárnym svetelným zdrojom, tak aj ako jeden zo vstup kroku tvorby ISM a následne výpočtu nepriameho osvetlenia.

Pri tvorbe využívame možnosti zápisu do viacerých cieľov renderovania súčasne (*Multiple Render Targets* – MRT, resp. G-buffer v directX terminológii) pre uchovanie štyroch textúr. Uchovávame *hĺbkovú mapu* pre výpočet viditeľnosti vzhľadom na primárny svetelný zdroj, *world-space koordináty* a *normálové vektory* pre tvorbu virtuálnych svetelných zdrojov (určenie polohy a smeru „pohľadu" jednotlivých vzorkovaných VPL) a napokon tzv. *flux*, určujúce farbu a intenzitu odrazeného svetla.

Momentálne je uvažovaný jediný primárny zdroj svetla. Ten môže byť jedným z troch základných druhov

- *Directional light* – Smerové svetlo (napr. slnečné svetlo), postačuje jediná textúra získaná paralelnou projekciou z pohľadu svetla.
- *Spotlight* – Svetelný kužeľ (napr. svetlo baterky), postačuje jediná textúra získaná perspektívnou projekciou.
- *Point light* – Bodový, všesmerový svetelný zdroj, pre získanie shadow mapy je možné využiť buďto cubemapu – čo však vyžaduje šesť prechodov – alebo alternatívne dual-paraboloid shadow mapping. Obdobne je možné tento typ špecifikovať na jeho hemisferickú variantu (paraboloid shadow mapping).



(a) Hĺbka (b) W-space súradnice (c) Normály (d) Flux

Obr. 5.3: Reflective Shadow Mapa jednoduchej scény (spotlight).

Ukážka využitia možností MRT v jazyku GLSL (fragment shader):

```
layout(location = 0) out vec3 worldCoord;
layout(location = 1) out vec3 normal;
layout(location = 2) out vec4 flux;

in vec3 Position;
in vec3 Normal;

uniform vec3 diffuseMaterialComponent;

void main()
{
    //depth is stored autamically
    worldCoord = Position;              //world space coordinates
    normal = Normal;                    //normal
    flux = vec4(diffuseMaterialComponent, 1);
}
```

5.2.2 Tvorba ISMs

Nasleduje samotná tvorbu textúry obsahujúcej jednotlivé ISM pre výpočet viditeľnosti z virtuálnych svetelných zdrojov. Keďže každý VPL pracuje len so sebe priradenou podmnožinou bodov reprezentujúcich scénu, postupujeme tak, že z množiny bodov postupne vyberáme podmnožiny o veľkosti zodpovedajúcej jednému VPL a tieto „prirenderujeme" do ISM textúry. Aby sme jedným prechodom pokryli celé pohľadové teleso VPL (ktoré má hemisferický charakter), využívame Paraboloid Shadow Mapping. Konkrétny virtuálny svetelný zdroj, teda textúrovacie koordináty do haltonovej textúry, ktoré ho určujú, je daný indexom práve spracúvaného bodu.

Paraboloid Shadow Mapping

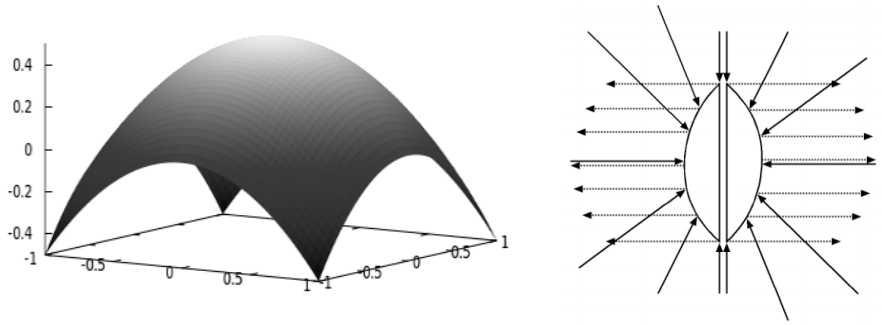
Vstupmi tohto kroku sú Reflective Shadow Mapa získaná v predchádzajúcom kroku (konkrétne textúry uchovávajúce world-space koordináty a normálové vektory), textúra so zakódovanou Haltonovou sekvenciou a offset do tejto textúry, daný poradím práve spracúvaného VPL a napokon world matica modelu.

Pri spracúvaní bodu je najskôr na základe offsetu vybratá hodnota z Haltonovej textúry. Táto hodnota je vlastne ukazateľom do RSM na hodnoty prislúchajúce aktuálnemu pracovnému VPL. Z textúry world-space koordinátov teda získame polohu a z normálovej textúry smer pohľadu VPL. Na základe týchto hodnôt vieme zostaviť pohľadovú (view, tzv. lookAt) maticu virtuálneho svetelného zdroja a následne realizovať výpočet paraboloidnej projekcie [8] [1] [9].

Základom úspešného zachytenia scény (resp. polpriestoru viditeľného hemisferickým virtuálnym svetelným zdrojom) je voľba vhodného paraboloidu použitého pre projekciu. Na základe [1] je zvolený paraboloid

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2 + y^2), \quad x^2 + y^2 \leq 1,$$

obraz zachytený kamerou smerujúcou na ktorý zachytáva informácie o celej hemisfére so stredom v $(0,0,0)$ a orientovanej v smere $d = (0,0,1)$. Paraboloid teda funguje ako šošovka, kedy všetky odrazené lúče vychádzajú z ohniska (viď obr. 5.4 (b), str. 24).



(a) Použitý paraboloid (b) Zachytenie okolia

Obr. 5.4: Paraboloidná projekcia.

Pre namapovanie 3D priestoru do 2D shadow mapy odrazom od paraboloidu je preto nutné nájsť súradnice bodu $P = (x, y, z)$, ktorý je priesečníkom spojnice projektovaného

bodu v priestore a ohniska (vektor $\vec{v}$) s povrchom paraboloidu, teda ktorý odráža vektor smerujúci zo stredu VPL k zobrazovanému bodu v smere normály VPL na základe normálového vektoru v bode P ($\vec{n} = \frac{1}{z}(x, y, 1)$). Ako hĺbková je teda použitá informácia o vzdialenosti bodu od počiatku v mieste danom získanými súradnicami. Pri samotnom shadow mappingu potom zobrazovanú geometriu (z pohľadu kamery) rovnako transformujeme do koordinátového systému jednotlivých svetiel. Zjednodušene realizácia projekcie v GLSL:

```
vec4 Position = LightView * World * vec4(position, 1.0);
Position /= Position.w;

ClipDepth = Position.z; //for clipping

//normalisation
float l = length(Position);
Position /= l;

//texture coords
Position.xy /= Position.z + 1.0;

//depth and w neutralisation
Position.z = (l - nearP)/(farP - nearP);
Position.w = 1.0;
```

Pohľadová matica svetla je pritom spočítaná na základe jej charakteristík získaných z reflective shadow mapy primárneho svetelného zdroja nasledovne (GLSL):

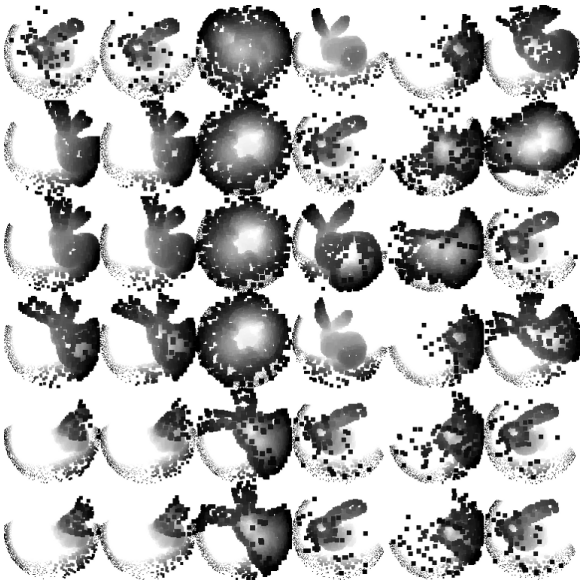
```
mat4 computeView(vec3 origin, vec3 direction)
{
  //classic lookAt computation - perpendicular direction, right, up
  direction = normalize(origin+direction);
  vec3 right = cross(direction, vec3(0,1,0));
  right = normalize(right);
  vec3 up = cross(direction, right);
  up = normalize(up);
  //fill in the matrix
  mat4 v~= mat4(right.x, up.x, -direction.x, 0.0,
                right.y, up.y, -direction.y, 0.0,
                right.z, up.z, -direction.z, 0.0,
                0.0, 0.0, 0.0, 1.0);
  //translate it with respect to VPL origin
  mat4 t = mat4(1.0, 0.0, 0.0, 0.0,
                0.0, 1.0, 0.0, 0.0,
                0.0, 0.0, 1.0, 0.0,
                -origin.x, -origin.y, -origin.z, 1.0);
  return v*t;
}
```

Po získaní 2D koordinátov, alebo teda koordinátov do výstupnej tieňovej textúry je tieto ešte nutné namapovať do zodpovedajúceho políčka v rámci celej shadow mapy zahŕňajúcej všetky virtuálne svetelné zdroje.

Vďaka použitiu bodovej reprezentácie scény sa tiež vyhneme prípadným problémom vznikajúcim pri interpolácii polygonálnej reprezentácie scény (teda stavu, keď vstupujúce vrcholy geometrie sú projektované do paraboloidného priestoru, samotné plochy sú však hardwareom interpolované lineárne).

Posledným krokom pred zápisom do ISM textúry je dynamické určenie veľkosti „splatu", ktorý v nej spracúvaný bod zanechá. Keďže túto chceme dynamicky meniť na základe nepriamej úmery voči zapisovanej hĺbke (resp. vzdialenosti), je nutné túto možnosť povoliť (v OpenGL prostredníctvom `glEnable(GL_VERTEX_PROGRAM_POINT_SIZE)`). V shader prog-

rame spracúvajúcom vykreslenie „splatu" do textúry potom meníme jeho veľkosť prostredníctvom vstavanej premennej `gl_PointSize`.



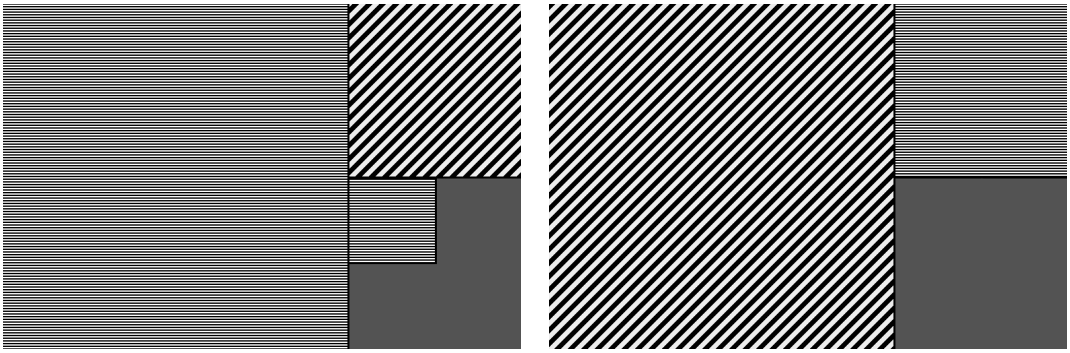
Obr. 5.5: Aplikácia ISM na jednoduchej scéne – 36 VPLs, pred Pull-Push prechodom.

5.2.3 Pull-Push

Na vytvorenú ISM je následne aplikovaný Pull-Push algoritmus tak, ako bol popísaný v 3.3. Vďaka umiestneniu jednotlivých ISM v jednej veľkej textúre sú upravené všetky súčasne. Tým je dosiahnuté vyplnenie dier v textúre vzniknutých v predchádzajúcom kroku a textúra je pripravená pre výpočet viditeľnosti pri nepriamom osvetlení.

Úzkym miestom algorimtu pre úpravu textúry pull-push metódou (alebo inou metódou využívajúcou budovanie pyramidálnej štruktúry nad upravovaným obrazom) je prenos dát medzi pamäťou grafickej karty a hlavnou pamäťou počítača. Moderný grafický hardware nám prostredníctvom offscreen renderingu umožňuje tento problematický aspekt potlačiť a celú úpravu vykonať na GPU. Pyramidálnu štruktúru, ktorú je nutné vytvoriť, „odsimulujeme" pomocou dvoch textúr v rozmere n×1.5n pixelov (kde n je veľkosť hrany štvorcovej ISM), do ktorých striedavo – metódou takzvaného „ping-pong" renderingu – renderujeme jednotlivé úrovne pyramídy. Rozloženie pyramídy v textúrach schematicky zobrazuje obrázok 5.6. Ako bolo povedané v 3.3, postačuje nám rátať s dvoma úrovňami pyramídy nad základňou.

Pre potreby algoritmu je teda vytvorený offscreen framebuffer s dvoma textúrami ako cieľmi renderingu. Základňou pyramídy je ISM vytvorená v predchádzajúcom kroku. Textúra, ktorá ju obsahuje, je potom vstupom pre tvorbu nasledujúceho stupňa, kdežto druhá textúra je jeho výstupom. Samotná tvorba každej ďalšej úrovne pyramídy prebieha vykreslením štvoruholníka polovičnej veľkosi oproti aplikovanej textúre na zodpovedajúce miesto cieľa (napr. pomocou `glViewport()`). Každá ďalšia úroveň je potom vytvorená vzájomnou výmenou vstupnej a výstupnej textúry, úpravou textúrovacích súradníc vykresľovaného štvoruholníka a zmenou viewportu (teda množiny pokrytých pixelov v cieľovej textúre). Použijeme pri tom RGBA textúru, kde v RGB kanáloch ukladáme hĺbkovú (pre sngle-bounce) informáciu a v alfa kanále indikujeme zápis, resp. validitu daného pixelu pre interpoláciu



(a) Pull-Push (b) Pull-Push

Obr. 5.6: Geometrické rozloženie pyramidálnej štruktúry použitej pri Pull-Push úprave ISM. Vodorovné pruhy predstavujú časti vytvorené v Pull, zatiaľčo šikmé v Push fáze.

v push fáze zápisom hodnoty `1.0`, ako je to ukázané v [13]. Pre výpočet „hrubšej" úrovne je teda nutné podľa 3.3 na základe veľkosti „jemnejšej" úrovne textúry určiť pole offsetov textúrovacích súradníc, z ktorých budeme získavať vzorky na interpoláciu. Následne prebieha samotné vzrokovanie a uchovanie počtu valídnych vzoriek. Ak aspoň jedna získaná vzorka bola valídna, súčet navzorkovaných hodnôt je následne prevážený počtom tých valídnych. V opačnom prípade je zapisovaný fragment indikovaný ako invalídny. Zjednodušene v GLSL:

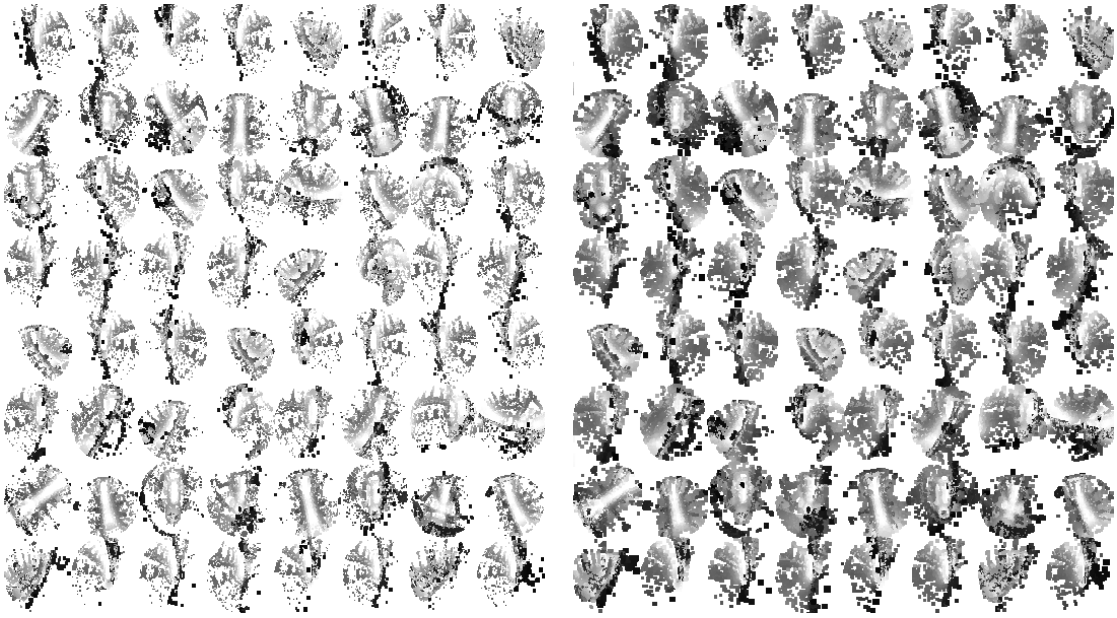
```
//get 4 surrounding values
for(int i = 0; i < 4; i++)
{
  //get value
  sample = texture2D(finerTex, TexCoord + offset[i]);
  //if valid
  if(sample.a == 1.0)
  {
    finerVal += sample.r;
    valids++;
  }
}

if(valids > 0)
{
  finerVal /= valids;
  coarserTex = vec4(vec3(finerVal), 1.0);
}else{  //else invalidate
  coarserTex = vec4(1.0, 1.0, 1.0, 0.0);
}
```

V push fáze potom „dotvoríme" interpolované časti pyramídy. Opäť postupujeme renderovaním štvorca do zodpovedajúcich častí výstupnej textúry, na vstupe však tentoraz máme ako textúru obsahujúcu aktuálnu (práve interpolovanú) úroveň pyramídy, tak úroveň hrubšiu (*coarser*). O väčšinu práce sa postará fragment shader, ktorý v prípade, že je úroveň alfa kanálu nenulová, vykreslí na dané miesto textúry aktuálnu hodnotu (teda nieje nutné interpolovať). V opačnom prípade na základe aktuálnych textúrovacích súradníc a poľa offsetov (závislého na veľkosti vstupujúcej „hrubšej" úrovne textúry) navzorkuje vstupnú (coarser) textúru a spočíta jednotlivým získaným vzorkám váhy zodpovedajúce 3.3 (str. 11). Váhy sú ešte upravené podľa počtu skutočne prispievajúcich valídnych pixelov. V prípade, že bol valídny aspoň jeden, je nová hodnota spočítaná na základe váženého

súčtu prispevajúcich valídnych vzoriek. Opäť zjednodušene v GLSL:

```
//get coarser level samples
for(int i = 0; i < 4; i++)
{
  samples[i] = texture2D(coarserTex, tc+offsets[i]);
  weights[i] = weigh(samples[i].a, i);
}
//adjust weights - only valid contribute
float w = 0;
for(int i = 0; i < 4; i++)
{
  w += weights[i];
}
//if at least one valid
if(w > 0)
{
  //add up weighted values
  for(int i = 0; i < 4; i++)
  {
    weights[i] /= w;
    outVal += samples[i].r*weights[i];
  }
  outColor = vec4(vec3(outVal), 1.0);
}
```

Výsledok tohoto kroku algoritmu je potom použiteľný pre výpočet viditeľnosti z pohľadu jednotlivých VPL (pre ilustráciu viď obr. 5.7).



Obr. 5.7: Výsledok Pull-Push algoritmu. a) zobrazuje textúru pred, b) po aplikácii Pull-Push pyramidálnej metódy. (64 virtuálnych svetelných zdrojov, 8 000 bodov na VPL, interiér katedrály v Šibeniku.)

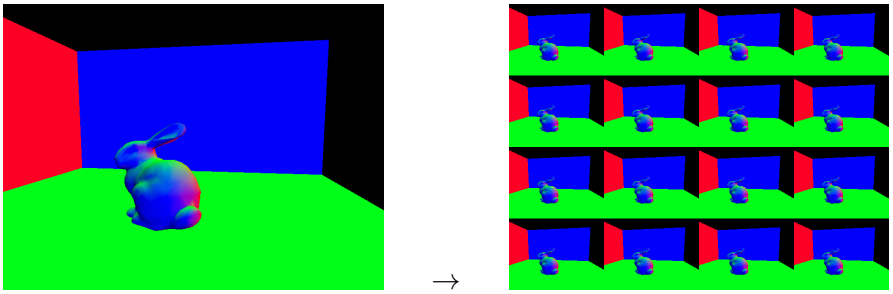
5.2.4 Render z pohľadu kamery

Keďže pri výpočte osvetlenia a tieňovania scény sa pracuje s veľkým množstvom (virtuálnych) svetelných zdrojov, s výhodou pri ňom môžme využiť techniku tzv. odloženého (*deferred*) renderingu. Hlavnou výhodou tohto prístupu je jediné kreslenie geometrie (a uchovanie vlastností a atribútov nutných pre výpočet) a následná kompozícia osvetlenia a tieňovania v screen space.

Uchovaná je pozícia, nutná ako pre výpočet tieňovania, tak pre projekciu do light space a výpočet nepriameho osvetlenia, normály a materiálové vlastnosti objektu. Materiálové vlastnosti nutné pre základný výpočet osvetlenia pomocou Phongovho modelu, sú uchované v jednej textúre, kde farba je uchovaná tradične v troch farebných kanáloch a alfa kanál je využitý pre uloženie informácie o lesklosti/odrazivosti (*shininess*) materiálu. Opäť využívame G-buffer (MRT), umožňujúci nám tieto nutné údaje získať a uchovať súčasne. Vytvorené textúry sú potom vstupmi tieňovacích operácií.

5.2.5 Interleaved G-buffer – split fáza

Pre samotný výpočet osvetlenia využijeme techniku Interleaved G-buffer [12] v princípe tak, ako bola stručne popísaná v 4.1 na strane 12. V predchádzajúcom kroku bol vytvorený tzv. initial G-buffer, pozostávajúci z textúr uchovávajúcich pozíciu, normálu a farbu (prípadne ďalšie pridružené materiálové vlastnosti, ako napr. *shininess*) z pohľadu kamery. Na samotné „preskladanie" je následne využité vykreslenie zarovnaného štvoruholníka do offscreen framebufferu veľkosťou zodpovedajúceho veľkosti okna aplikácie (resp. vstupných textúr tvoriacich initial buffer). Ten pozostáva z rovnakého počtu farebných textúr, ako initial buffer. Ťažisko práce spočíva v tomto kroku na fragment shaderi. Okrem samotného initial bufferu obsahujúceho preskladávané hodnoty sú jeho vstupom aj uniformné premenné obsahujúce informácie o počte blokov v horizontálnom a vertikálnom smere, ako aj informácie o šírke a výške okna, alebo intial bufferu, v pixeloch. Na základe týchto informácií sú spočítané textúrovacie koordináty do intial bufferu, hodnoty z ktorých sú zapísané na výstup na mieste práve spracúvaného fragmentu.



(a) Initial buffer (b) „Preskladaný" buffer

Obr. 5.8: G-buffer splitting.

Každej „dlaždici" vytvoreného, tzv split G-bufferu, je následne pri výpočte osvetlenia priradená určitá podmnožina virtuálnych svetelných zdrojov tak, aby všetky dlaždice dostávali príspevok od rovnakého počtu. Následne pre jednotlivé dlaždice spočítame osvetlenie za použitia RSM vytvorenej v prvom kroku, resp ISM v druhom. Buffer obsahujúci informácie o osvetlení scény potom „preskladáme" do pôvodnej podoby. Tým zabezpečíme pre každú dlaždicu initial bufferu príspevok všetkých VPL, pričom jednotlivé pixely sú počítané

len z ich určitej podmnožiny. Problémom použitej metódy je vznik inkonzistencií medzi susediacimi pixelmi.

5.2.6 Výpočet osvetlenia

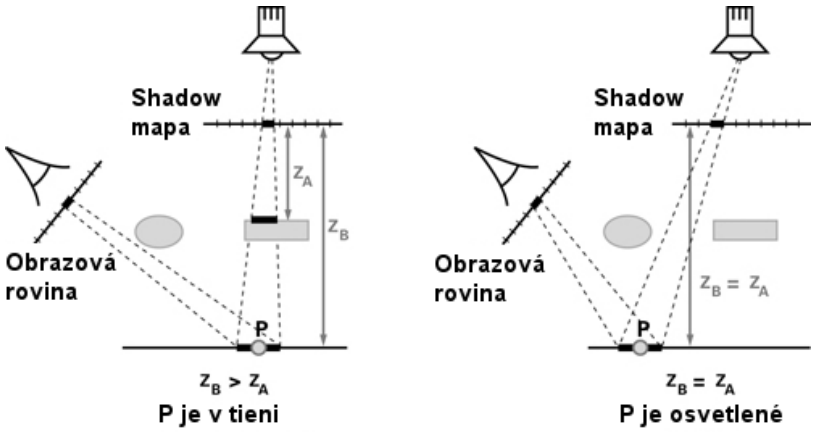
Ako bolo spomenuté, pre výpočet osvetlenia je, ako vyplýva z doteraz rozoberaných fáz algoritmu kreslenia scény, využitá metóda takzvaného odloženého (*deferred*) renderingu. Jej princíp spočíva v tom, že jednotlivé shading operácie sa nevykonávajú priamo pri renderingu geometrie scény z pohľadu kamery, ale takto sú získané len informácie pre tieňovanie nevyhnutné (hĺbka/pozícia fragmentu vo world space, hodnota normály, materiálové vlastnosti ako napr. difúzna farba a pod.) a samotný výpočet osvetlenia je následne vykonaný na ich základe v screen space. Hlavnou výhodou tejto techniky je jednoduchá aplikovateľnosť veľkého množstva svetelných zdrojov prakticky nezávisle na zložitosti geometrie scény.

Celková kompozícia osvetlenia scény teda pozostáva z príspevkov priameho osvetlenia (teda osvetlenia scény spôsobeného primárnym svetelným zdrojom) Phongovou tieňovacou technikou vrátane riešenia viditeľnosti a tieňov a osvetlenia nepriameho (teda osvetlenia spôsobeného odrazom svetla od povrchov osvetlených primárnym svetelným zdrojom, aproximovaného prostredníctvom VPLs a RSM) s riešením viditeľnosti pomocou techniky ISM.

Priame osvetlenie

Okrem tieňovania objektov v závislosti na osvetlení primárnym svetelným zdrojom je dôležitou súčasťou výpočtu primárneho osvetlenia scény výpočet tieňov. Prítomnosť tieňov v scéne je veľmi dôležitá nielen pre dosiahnutie prirodzenejšieho vizuálneho dojmu zo scény, ale aj pre zlepšenie vnímania hĺbky pozorovateľom (optické „ukotvenie" objektov).

Pre tvorbu primárnych tieňov je použitá metóda shadow mapping. Na jej uskutočnenie máme k dispozícii hĺbkovú mapu získanú pri generovaní RSM v skoršom kroku algoritmu, ako aj informáciu o pozícii fragmentu vo world space, získanú pri generovaní g-bufferu pre výpočet osvetlenia.



Obr. 5.9: Shadow Mapping (zdroj: nVidia)

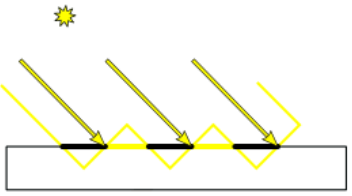
Pre porovnanie hĺbky aktuálne riešeného fragmentu s hĺbkou uchovanou v tieňovej mape (teda hĺbkou objektu najbližšie k svetlu) je nutné transformovať jeho pozíciu z world space do light space. K tomu je k dispozícii transformačná matica svetla, použitá pri generovaní

shadow mapy. Po vynásobení pozície fragmentu transformačnou maticou svetla sa získajú jeho koordináty v normalizovaných súradniciach zariadenia (teda v rozmedzí [-1;1]), zatiaľčo pre korektný náhľad do tieňovej mapy sú nutné textúrovacie súradnice v rozmedzí [0;1]. Škálovanie a posun koordinátov je realizované vynásobením transformačnej matice tzv. bias maticou:

$$bias = \begin{bmatrix} 0.5 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 \\ 0.5 & 0.5 & 0.5 & 1.0 \end{bmatrix}$$

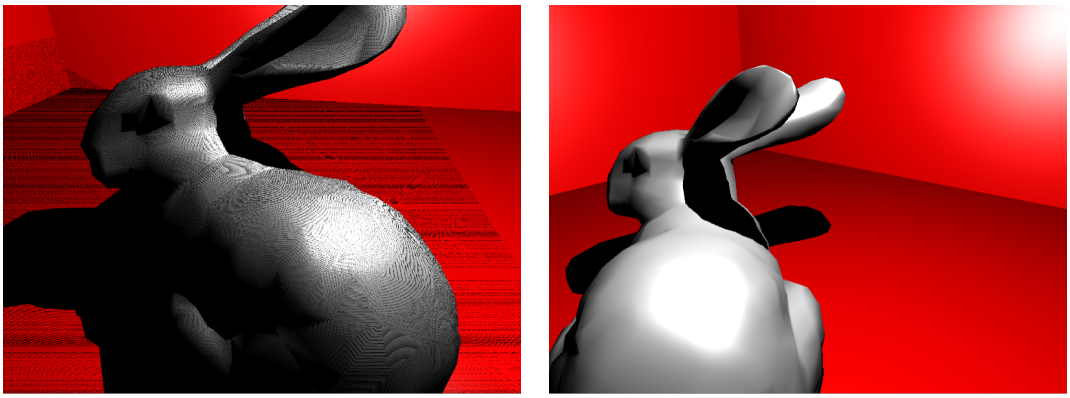
Samotné porovnanie hĺbky transformovaného fragmentu a hĺbky uchovanej v hĺbkovej textúre je možné (v prípade, že nepočítame projekciu „ručne" – napr. pri paraboloidnej projekcii) vykonať napr. použitím vstavanej funkcie `textureProj` jazyka GLSL. Vhodným nastavením vlastností textúr uchovávajúcich shadow mapu (`MIN_FILTER`, `MAG_FILTER`, typ a možnosti pri porovnávaní) je tiež možné využiť HW podporu PCS (Percentage-Close Filtering) pre zjemnenie okrajov získaných tieňov.

Jedným z problémov vznikajúcich pri aplikácii shadow mappingu v jeho základnej podobe je vznik takzvaného *shadow acne*. Tento jav vzniká v dôsledku zapadnutia častí zobrazovaných fragmentov „medzi pixely" shadow mapy (viď obr. 5.10, resp. 5.11).



Obr. 5.10: Princíp vzniku shadow acne.

Jeho najjednoduchším riešením je pozmenenie hĺbkovej informácie aktuálne spracúvaného fragmentu projektovaného do light space odčítaním korekčného členu blízkeho nule, obvykle napr. 0.005, pred porovnaním s hĺbkovou informáciou uloženou v shadow mape z pohľadu svetla.



Obr. 5.11: Aplikácia Shadow mappingu pred a po korekcii shadow acne.

Zistenie, či sa práve skúmaný fragment nachádza v tieni potom zjednodušene za pomoci GLSL vstavanej funkcie `textureProj` pre výber z textúry s projekciou:

```
//get fragment world coords from g-buffer
vec3 position = texture2D(WSCtex, TexCoord);
//transform into light space (bias applied)
vec4 shadowPos = depthBiasMVP * vec4(position, 1);
//shadow acne correction
shadowPos.z -= 0.005;
//get shadow value
float shadow = textureProj(shadowMap, shadowPos);
```

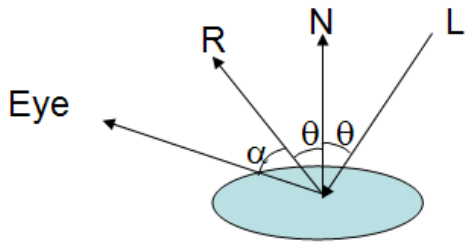
Len tie časti obrazu, ktoré sú vyhodnotené ako osvetlené priamym svetlom (teda sa nenachádzajú v tieni), sú ďalej spracované výpočtom zakladajúcim sa na Phongovom osvetľovacom modeli. Tento má vo svojej základnej podobe tri zložky - ambientnú, difúznu a zložku spekulárnych odleskov. Vyjadrené osvetľovacou rovnicou osvetlenie (*illumination*) I v bode p:

$$I_p = k_a i_a + k_d(L \cdot N) i_d + k_s(R \cdot Eye)^s i_s$$

L predstavuje vektor od svetelného zdroja k povrchu.
N predstavuje normálu povrchu v bode dopadu svetla.
Eye je vektor medzi kamerou (okom pozorovateľa) a povrchom.
k_x, i_x značia materiálové konštanty, resp. farby jedn. zložiek a farbu svetla.
s napokon predstavuje konštantu „odrazivosti" (shininess) materiálu.
R predstavuje vektor svetla odrazeného podľa normály N v danom bode vzťahom:

$$R = 2(L \cdot N)N - L$$

Ambientnú zložku, ktorá je v modeli zahrnutá ako jednoduchá, výpočetne veľmi „lacná" náhrada nepriameho osvetlenia scény, resp. rozptylu svetla v scéne, môžeme pri našom výpočte úplne vypustiť, keďže nahrádzané nepriame osvetlenie bude neskôr dopočítané.



Obr. 5.12: Phongov osvetľovací model

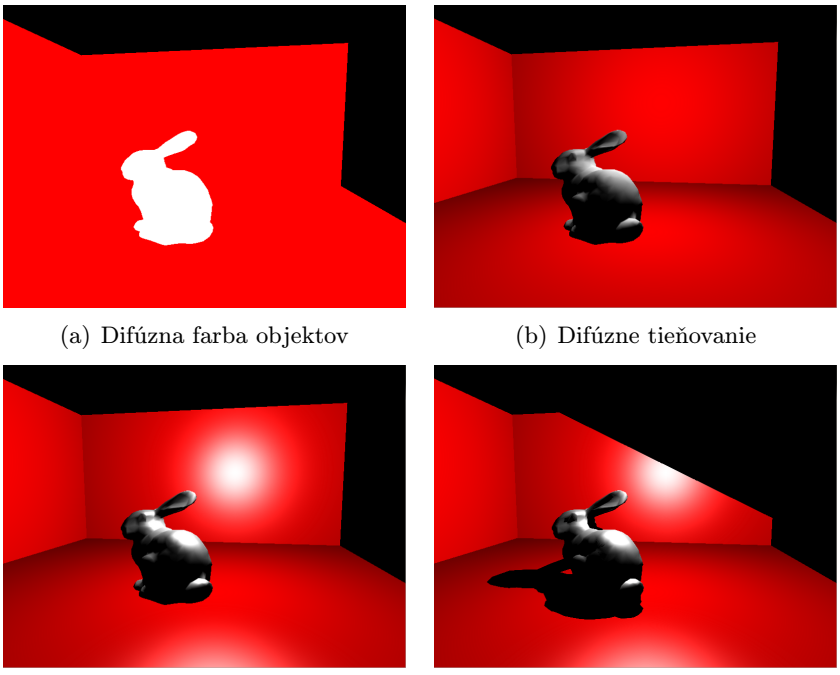
Difúzna zložka osvetlenia je aproximáciou osvetlenia difúznych povrchov priamym svetlom. Jej intenzita je daná orientáciou osvetľovanej plochy vzhľadom na svetelný zdroj (najvyššia pri dopade lúčov kolmo na plochu, najnižšia pri rovnobežnosti), farba je daná materiálovými vlastnosťami povrchu a farbou svetelného zdroja, čím sa simuluje fakt, že rôzne materiály pohlcujú rôzne vlnové dĺžky svetla a výsledná difúzne vyžiarená farba určuje vnímanú farbu materiálu. Je nezávislá na pozícii pozorovateľa. Príklad výpočtu v GLSL:

```
//get light vector
vec3 L = normalize(lightPosition - position);
//get normal
vec3 N = normalize(normal);
//compute angle between them
float cosTheta = max(0.0, dot(N,L));
//compute diffuse color
diffuse = vec4(materialColor.rgb, 1.0) * lightColor * cosTheta;
```

Zložka *spekulárnych odleskov* aproximuje svetlo odrazené od povrchu objektu a zachytené kamerou/okom pozorovateľa. Jeho intenzita je daná vzájomnou orientáciou odrazeného svetelného lúča a spojnice medzi bodom odrazu a kamerou/okom. Čím bližšie k stredu kamery sa lúč odráža, tým jasnejší je pozorovaný odlesk. Intenzita je ďalej upravená exponenciálnym vzťahom závislým na materiálových vlastnostiach objektu, keďže jej pokles vzhľadom na uhol medzi odrazeným lúčom a vektorom ku kamere nebýva obvykle lineárny. Farba je daná farbou svetelného zdroja. Príklad (GLSL):

```
//get eye vector
vec3 Eye = normalize(eyeDir); //camera position - eyeSpacePosition
//get light reflection vector
vec3 R = reflect(-L, N);
//compute angle between them
float cosAlpha = max(0.0, dot(Eye,R));
//get material shininess
float shininess = materialColor.a;
//compute specular color
specular =  lightColor * pow(cosAlpha, shininess);
```

Výsledná farba daná priamym osvetlením objektu je potom získaná súčtom jednotlivých zložiek.



(a) Difúzna farba objektov

(b) Difúzne tieňovanie

(c) Spekulárne odlesky

(d) Tiene

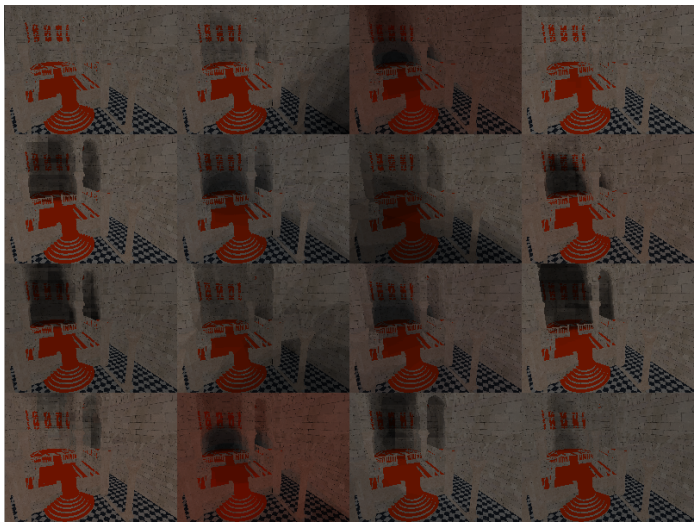
Obr. 5.13: Priame osvetlenie

Nepriame osvetlenie

Priame svetlo samozrejme v realite nieje jedinou zložkou osvetlenia objektov. V takom prípade všetky plochy, na ktoré nedopadá svetlo priamo zo svetelného zdroja (teda sa nachádzajú v tieni), by boli nepretržite v absolútne čiernej temnote. V dôsledku rozptylu svetla, jeho odrazov a difúzneho vyžarovania objektov dochádza k nepriamemu osvetleniu aj týchto – inak neviditeľných – plôch. Techniky ako Instant Radiosity, alebo Reflective Shadow Mapping tento jav aproximujú generovaním virtuálnych svetelných zdrojov na povrchoch osvetlených priamym svetlom (bližšie viď. 2.2, str. 6). Rovnakým princípom je riešený výpočet nepriameho osvetlenia metódou využívajúcou ISM, avšak riešiac ich zásadný problém, a to buďto neexistujúce riešenie viditeľnosti (teda vrhaných tieňov objektami, resp. možnosti osvetlenia plôch) vzhľadom k týmto virtuálnym svetelným zdrojom, alebo ich riešenie „drahými" presnými metódami (Shadow Mapping, Shadow Volumes).

Okrem informácií o objekte (normály, pozície, materiál), získaných pri vykreslení z pohľadu kamery sú pre výpočet nutné obdobné informácie z pohľadu primárneho svetelného zdroja, uchované v RSM v predošlých krokoch. Na základe jej vzorkovania sú určované pozície, normály (smerovania) a farby jednotlivých sekundárnych zdrojov osvetlenia. Ďalším vstupom je potom ISM, ktorá uchováva tieňové mapy zodpovedajúce týmto VPL.

Interleaved G-buffer je využívaný, aby nebolo nutné pre výpočet nepriameho osvetlenia každého pixelu vzorkovať, riešiť viditeľnosť a započítavať príspevky všetkých VPL, ktorých môže byť pomerne veľké množstvo. Preto pred započatím výpočtu najskôr určíme, v ktorom segmente „preskladaného" G-bufferu sa práve počítaný pixel nachádza a na základe tejto informácie sa určí podmnožina VPL, ktorá sa použije pre výpočet. Jednoduchou realizáciou tohto je napr. zvolenie jedného riadku textúry uchovávajúcej haltonovu postupnosť a zodpovedajúceho riadka ISM a iteratívne vzorkovanie nimi určených virtuálnych svetelných zdrojov. Počet VPL skutočne použitých pri výpočte je potom druhou odmocninou celkového počtu VPL.

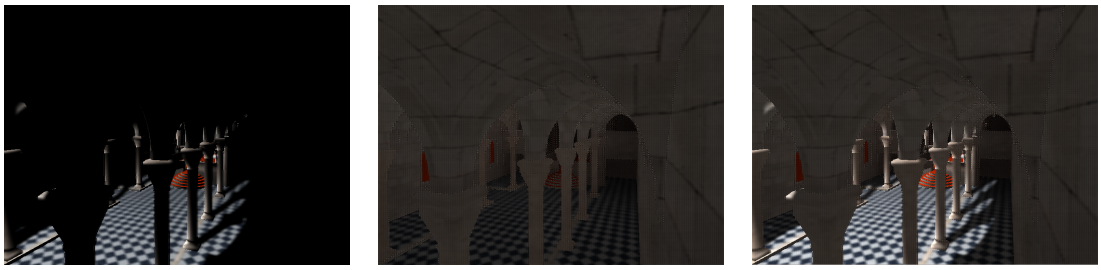


Obr. 5.14: Výpočet nepriameho osvetlenia v rozdelenom G-bufferi. Každý diel je počítaný s inou podmnožinou virtuálnych svetelných zdrojov.

Prvým krokom je teda určenie offsetu do týchto dvoch textúr, zodpovadjúceho segmentu G-bufferu. Následne sa vykoná cyklus cez šírku haltonovej textúry a v každej iterácii sa vykoná nasledujúca postupnosť operácií. Je posunutý offset do haltonovej textúry a do

ISM. následne je haltonova textúra navzorkovaná, čím sa získajú textúrovacie koordináty do RSM, odkiaľ sa budú vzorkovať atribúty VPL. Z RSM je navzorkovaná pozícia, normála a farba svetelného zdroja. Následne, podobne ako pri tvorbe ISM, je na základe pozície aktuálne spracúvaného fragmentu a vlastností aktuálneho VPL fragment projektovaný do paraboloidných koordinátov light space VPL. Jeho hĺbka v tomto priestore je porovnaná s informáciou získanou zo zodpovedajúceho miesta zodpovedajúcej bunky ISM a v prípade, že je menšia, k nepriamemu osvetleniu fragmentu je zarátaný príspevok tohto VPL, v opačnom prípade je vyhodnotený ako zatienený. Farebný príspevok VPL je obmedzený na difúznu zložku osvetľovacieho modelu, spočítanú obdobne ako pri priamom osvetlení, keďže sa jedná o svetlo spätne vyžiarené difúznym materiálom v scéne. Prípadná chyba spôsobená veľmi malým rozlíšením ISM a jej nepresnosťou spôsobenou vykreslením na základe zjednodušenej bodovej geometrie, ako aj následnou aplikáciou pull-push algoritmu, je „spriemerovaná" využitím väčšieho množstva VPL a následne v ďalšej fáze algoritmu aj preskladaním a filtráciou G-bufferu.

Výsledná farba fragmetnu je daná súčtom farebných príspevkov získaných priamym a nepriamym osvetlením objektu.



(a) Priama zložka (b) Nepriama zložka (c) Výsledná kompozícia

Obr. 5.15: Výpočet osvetlenia

5.2.7 Interleaved G-buffer – gather fáza

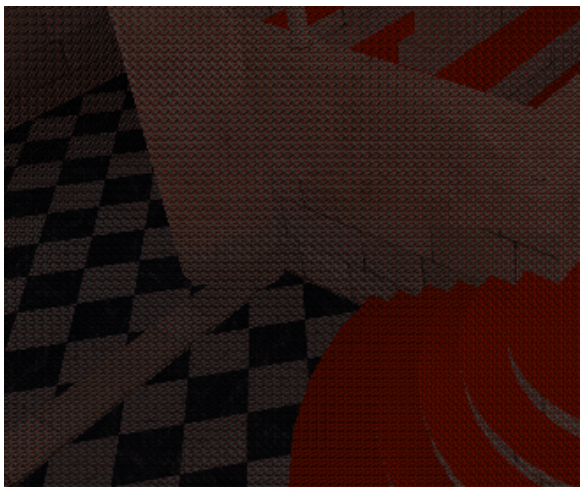
Po ukončení výpočtu osvetlenia je obraz v G-bufferi rozdelený na diely, z ktorých každý zodpovedá priamemu osvetleniu skombinovanému s nepriamym osvetlením získaným na základe určitej podmnožiny VPL. Pre získanie jediného výsledného obrazu je teda nutné buffer „preskladať" do pôvodnej podoby. Tento krok je tiež označovaný ako *buffer gathering*. Jedná sa prakticky o proces mapovania inverzný k split fáze práce s G-bufferom.

Toto je dosiahnuté vyrenderovaním štvoruholníka zarovnaného voči kamere a zaberajúceho celé okno aplikácie. Vertex shader pritom len posunie súradnice vrcholov a im priradené textúrovacie koordináty ďalej k interpolácii. Fragment shader následne pri vykreslení na základe veľkosti okna, informácie o počte blokov v horizontálnom aj vertikálnom smere (získanej v podobe uniformnej premennej) a pozície aktuálne spracúvaného fragmentu (`gl_FragCoord`) spočíta textúrovacie koordináty do textúr uchovávajúcich rozdelený G-buffer. Odtiaľ získaná farebná informácia je teda použitá ako výstup shaderu a je tak priradená na jej správne miesto na obrazovke (resp. v offscreen Framebufferi).

5.2.8 Geometry-aware blur a finálny render

Dôsledkom použitia Interleaved G-bufferu pre výpočet osvetlenia je fakt, že po jeho spätnom zložení je nepriame osvetlenie susediacich pixelov výsledného obrazu počítané z rôznych

množín virtuálnych svetelných zdrojov, čo spôsobuje vidteľné vizuálne artefakty prejavujúce sa ako štrukturovaný šum. Ich odstránenie je možné realizovať aplikovaním vhodnej vyhladzovacej metódy. Keďže je nutné brať do úvahy, že scéna je tvorená zložitými objektami (aby sa zabránilo „pretekaniu" farby, resp. informácie o osvetlení medzi objektami), je nutné voliť nejakú geometry-aware, resp. edge-aware metódu. Autori algoritmu odporúčajú postup publikovaný v rámci [6], vychádzajúci z jednoduchého filtrovania obrazu gaussovským kernelom s rozmermi dieliku (bunky) G-bufferu a porovnania rozdielu pozícií a normál práve spracúvaného pixelu a pixelov pod kernelom pre separáciu spojitých plôch, na ktoré je takéto filtrovanie možné aplikovať.



Obr. 5.16: Artefakty vznikajúce spätným zložením G-bufferu (nepriama zložka osvetlenia).

Voľbe vhodnej šírky kernelu Gaussovho filtra použitého pre vyhladenie týchto artefaktov sa stručne venujem v kapitole 6 nas str. 40.

Discontinuity buffer

Pred samotným započatím filtrovania je nutné najskôr detekovať nespojitosti v obraze. Na to poslúži prakticky detekcia hrán, pričom výsledná „intenzita" hrany nieje určená na základe porovnania farebnej informácie skúmaného pixelu s jeho okolím, ale na základe porovnania informácií o hĺbke a normále z pohľadu kamery, získaných pri tvorbe textúr neskôr využitých pri deferred výpočte osvetlenia. Keďže takáto detekcia slúži len na zlepšenie optickej kvality výsledného obrazu, nepožadujeme od použitých konvolučných jadier zvláštnu robustnosť a aj v záujme čo možno najjednoduchšieho výpočtu postačí použitie jednoduchého Roberts-Crossovho operátoru.

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{, resp.} \quad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Obr. 5.17: Roberts-Cross operátor

Aplikácia Roberts-Crossovho operátoru prebieha prakticky konvolúciou obrazu s dvoma jadrami o rozmere 2×2 (viď obr. 5.17). Pre detekciu „hrán" tak stačí – na rozdiel od napr. Sobelovho operátoru – získať tri dodatočné vzorky „vpravo hore" od práve skúmaného pi-

xelu. Takto získaná hodnota gradientu v x-ovom (G_x), resp. y-ovom (G_y) smere následne slúži pre výpočet intenzity v danom mieste obrazu podľa štandardného vzťahu

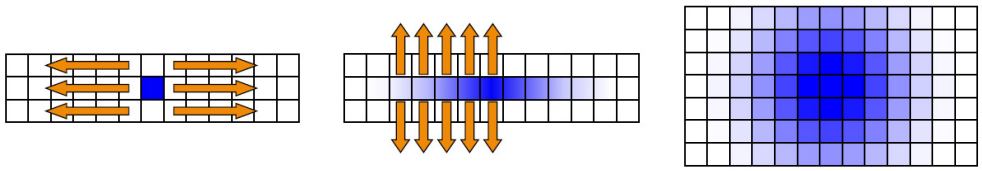
$$I(x, y) = \sqrt{G_x^2 + G_y^2}.$$

Týmto spôsobom sú v obraze detekované „hrany" značiace prudký rozdiel, teda nespojitosť (*discontinuity*) v normálach a hĺbkovej informácii z pohľadu kamery. Nespojitosť v hĺbke vzniká medzi dvoma plochami orientovanými tým istým smerom, avšak v scéne od seba vzdialenými (napr. tabuľa stola a podlaha pri pohľade zhora), zatiaľčo nespojitosť v normálach signalizuje blízke, avšak rôzne orientované plochy (napr. hrana medzi podlahou a stenou). Keďže pri filtrovaní výsledného obrazu využívame discontinuity buffer na zabránenie „pretekania" farieb z objektu do objektu pri aplikácii gaussiánu, sú pre nás nežiaduce obe situácie a do výsledného discontinuity bufferu zapíšeme 1/0 v prípade, že nájdeme nespojitosť či už v hĺbke, v normále, alebo oboch (teda aplikujeme logické *OR*). Hodnoty intenzít získané konvolúciou sú pred rozhodnutím o ich zápise (teda pred rozhodnutím o existencii „hrany" v danom ohľade na danom mieste obrazu) ešte prahované vhodne (experimentálne) zvoleným prahom.

Obr. 5.18: Discontinuity buffer.

Separabilný Gaussián

Samotné filtrovanie je potom uskutočnené aplikáciou gaussovho filtra. Pre značné zrýchlenie výpočtu využijeme jeho separabilitu, teda namiesto jedného prechodu a aplikácie dvojrozmerného filtra uskutočníme prechody dva, pričom v prvom je na pôvodný obraz aplikovaná len jednorozmerná zložka (napr. v x-ovom smere) a následne na dosiahnutý medzivýsledok v druhom prechode opäť kolmá jednorozmerná zložka (teda napr. v y-ovom smere). Každé filtrovanie pozostáva zo vzorkovania vstupnej textúry, váhovania vzoriek vhodným koeficientom (zodpovedajúcim gaussovmu rozloženiu a vzdialenosti od stredového pixelu) a sčítanie takto vážených vzoriek, napríklad pri gaussovom filtri 9×9 separabilným výpočtom zredukujeme počet operácií z 81 pri dvojrozmernom jadre na 18 pri dvoch jednorozmerných, pričom dosiahnutý výsledok je rovnaký.



(a) Blur originálu v smere x. (b) Blur medzivýsledku. (c) Finálny výsledok

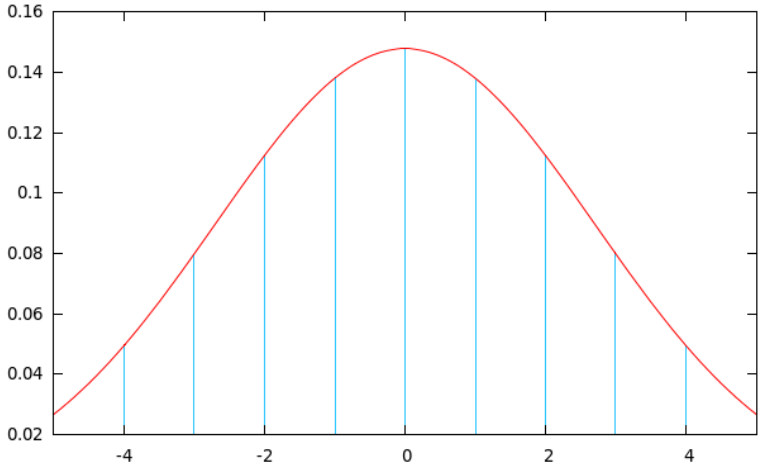
Obr. 5.19: Separabilná aplikácia Gaussovho filtra (zdroj obr.: ATI).

Aby sa zabránilo „prelievaniu" farby z objektov pri filtrovaní, ako pomocná informácia vstupuje do tohto kroku discontinuity buffer. Ak sa pixel nachádza priamo na nespojitosti, nieje filtrovaný vôbec. V opačnom prípade sa získajú príspevky z jeho okolia v kladnom a zápornom smere v osi, v ktorej je práve aplikovaná jednorozmerná filtrácia. V jednotlivých smeroch najskôr kontrolujeme, či na súradniciach suseda nieje označená nespojitosť a jeho príspevok je započítaný len v negatívnom prípade. V opačnom je posun v danom smere pozastavený a váha jednotlivých prispievajúcich pixelov zodpovedajúco upravená. Pre filtrovanie a výpočet výslednej farby pixelu je teda použitá len tá časť jeho okolia, ktorá je súčasťou spojitej plochy (teda po najbližšiu diskontinuitu, alebo šírku kernelu).

Výpočet váh jednotlivých vzoriek je odvodený z gaussovej funkcie

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

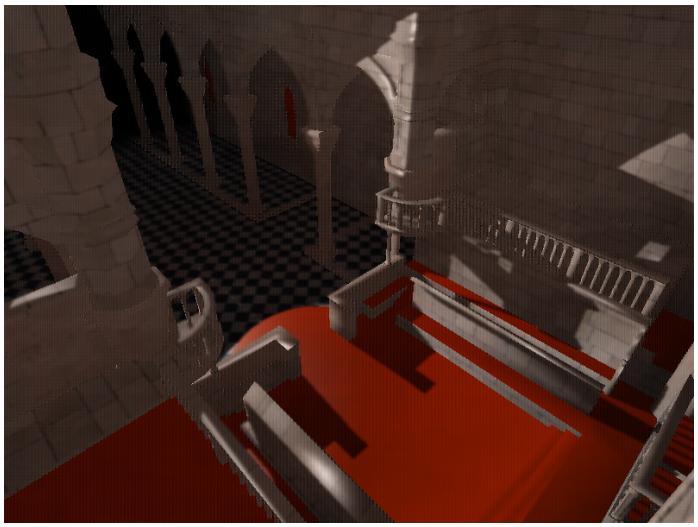
,kde x je offset od práve počítaného pixelu, σ smerodatná odchýlka (určujúca šírku „zvonca" gaussovej funkcie a teda mieru rozmazania) a μ pozícia stredu (teda výchylka stredu od práve počítaného pixelu). Pre zachovanie jasových hodnôt je získané váhy nutné normalizovať predelením ich súčtom. Tak napr. pre šírku kernelu 9 pixelov a smerodatnú odchýlku $\sigma = 2.7$ dospejeme k hodnotám $g(0) = 0.16, g(1) = g(-1) = 0.15, g(2) = g(-2) = 0.12, g(3) = g(-3) = 0.09, g(4) = g(-4) = 0.05$. Takýmto spôsobom je jednoducho možné zmeniť parametre filtra (šírka, hodnoty váh) pre čo možno najlepší výsledok v závislosti na miere prekladu G-bufferu použitého pri výpočte osvetlenia.



Obr. 5.20: Výber váh pre vzorky filtrované gaussovým filtrom šírky 9 (hodnoty pred normalizáciou).

Zobrazenie konečného výsledku

Finálna kompozícia vzniknutá spracovaním obrazu je napokon zobrazená ako textúra nanesená na štvoruholníku zarovnanom voči kamere a zaberajúcom celú plochu okna aplikácie.



Obr. 5.21: Výsledný render.

Kapitola 6

Pokusy a testovanie

Po vyhotovení funkčnej demonštračnej aplikácie som následne vykonal niekoľko pokusov a meraní, zaoberajúcich sa ako výkonom mnou implementovaného algoritmu, tak rôznymi aspektmi obrazovej kvality.

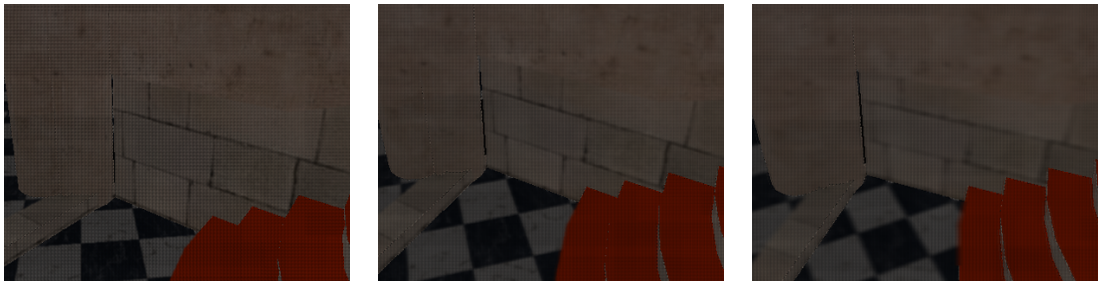
Testovacie scény

Ako je možné vidieť na obrázkoch sprevádzajúcich a ilustrujúcich tento text, väčšina testov bola uskutočnená na grafickej scéne tvorenej modelom katedrály v Šibeniku, ktorý je dnes už takmer štandardným testovacím modelom rôznych grafických algoritmov. Model tvorý 75 284 trojuholníkov pozostávajúcich z 83 490 vrcholov a je voľne dostupný na URL `http://graphics.cs.williams.edu/data/meshes.xml`. Ďalšie testovanie (najmä počas samotného vývoja demonštračnej aplikácie) bolo vykonané s jednoduchým modelom scény pozostávajúcej z niekoľkých rovín tvoriacich steny „boxu" a sošky zajačika (*Stanford Bunny*), obrázky ktorej sú použité najmä v skorších kapitolách tohto textu. Tento pozostáva z 1 056 vrcholov tvoriacich spolu 2 098 trojuholníkov.

Artefakty vznikajúce zložením G-bufferu

Ako bolo spomenuté v kapitole pojednávajúcej o implementácii algoritmu, pri výpočte osvetlenia v rámci rozdeleného G-bufferu vznikajú po jeho spätnom zložení viditeľné a značne rušivé artefakty (viď obr. 5.16, str. 36). Na tieto je nutné aplikovať separabilný Gaussov filter s vhodne zvolenou šírkou jadra. Ako pokusný prípad som zvolil delenie G-bufferu na 4×4 bloky. Následne som postupne aplikoval filtre o šírke kernelu 3 a smerodatnej odchýlke $\sigma = 0.9$, 5 a $\sigma = 1.5$ a 7 a $\sigma = 2.1$.

Ukázalo sa, že použitím flitrov o šírke 7 a vyššej síce dosiahneme plynulejší farebný prechod, avšak začínajú sa strácať detaily difúznych textúr a obrazových vzorov, čím kvalita zobrazenia značne utrpí. Použitím filtra o šírke 3 (teda užšieho ako zložený blok pixelov) naopak nedosiahneme dostatočné farebné „vyhladenie" a artefakty sú stále veľmi viditeľné. Pri použití filtra o šírke 5 sa dosiahol „rozumný kompromis" medzi oboma zložkami ovplyvňujúcimi vnímanú kvalitu obrazu (resp. tie jej zložky, ktoré závisia na filtrovaní, menovite artefakty, resp. detaily textúr). Plynulosť farebných prechodov už pôsobí dostatočne prirodzene, zostávajúce čiastkové artefakty po skombinovaní s priamou zložkou osvetlenia už nepôsobia prílišne rušivo. Je teda zrejmé, že pri delení G-bufferu na väčší počet dielov by bolo nutné aplikovať zodpovedajúco rozšírený filter (šírkou približne rovnaký, alebo mierne



(a) 3×3, $\sigma = 0.9$ (b) 5×5, $\sigma = 1.5$ (c) 7×7, $\sigma = 2.1$

Obr. 6.1: Aplikácia separabilného Gaussovho filtra na nepriamu zložku osvetlenia.

presahujúci šírku spätne zloženej časti obrazu), čím by utrpela kvalita textúr reprezentujúcich povrchy so zložitejším, alebo výraznejším vzorom.

Z výkonnostného hľadiska sa zmena šírky kernelu separabilného Gaussovho filtra ukázala ako takmer zanedbateľná.

Výkonnostné testovanie

Výkon demonštračnej aplikácie som testoval opäť na scéne Šibenickej katedrály. Bol do nej vložený pohyblivý animovaný svetelný zdroj a následne vykonaných niekoľko „preletov" kamery katedrálou. Skúmaný bol priemerný počet snímkov vykreslených aplikáciou za sekundu (FPS) pri rôznych konfiguráciách atribútov aplikácie.

V prvom teste bol najskôr skúmaný prínos Interelaved G-bufferu. Výkon aplikácie bol teda meraný najskôr bez rozloženia a spätného zloženia G-bufferu, teda pri počítaní každého pixelu výsledného obrazu zo všetkých VPL a následne pri rozložení G-bufferu a počítaní svetelných príspevkov zo zodpovedajúcich podmnožín všetkých VPL. V oboch prípadoch bolo každému jednotlivému svetelnému zdroju priradených 8 000 bodov. V prípade nevyužitia prekladania G-bufferu tiež nebol vykonaný ani záverečný filtrovací krok. V opačnom prípade bolo použité prekladanie 4×4.

#VPL	16	16	64	64	256	256
IGB	F	T	F	T	F	T
FPS	12	19	7	14	2	5

Tabuľka 6.1: Beh aplikácie (v FPS) v závislosti na použití interleaved G-bufferu.

Prvý riadok tabuľky 6.1 predstavuje počet generovaných svetelných zdrojov. Druhý riadok indikuje pravdivostnú hodnotu využitia techniky interleaved G-bufferu. Napokon v poslednom riadku sú uvedené dosiahnuté rýchlosti. Namerané hodnoty sú mediánovými hodnotami niekoľkých nezávislých behov aplikácie, pričom pre výber mediánu boli použité mediánové hodnoty jednotlivých niekoľkosekundových behov. Z výsledkov je zrejmé, že pri väčších počtoch VPL je využitie interleaved G-bufferu výrazným zrýchlením (pri 256 VPL presahujúcim 100%).

V ďalšom teste som skúmal výkonnosť pri rôznych nastaveniach počtu virtuálnych svetelných zdrojov a počtu pridelených bodov bodovej reprezentácie scény jednotlivým VPL. G-buffer bol pre jednoduchosť vo všetkých prípadoch rozdelený rovnako na 4×4 bloky.

Prvý riadok (#VPL) tabuľky 6.2 obsahuje počet VPL generovaných v scéne. Druhý riadok (#p/VPL) predstavuje počet bodov pridelených jednému virtuálnemu svetelnému

#VPL	16	16	64	64	256	256
#p/VPL	4 000	8 000	4 000	8 000	4 000	8 000
FPS	22	19	16	14	6	5

Tabuľka 6.2: Beh aplikácie (v FPS) v závislosti na počte VPL a im pridelených bodov.

zdroju. Napokon v treťom riadku tabuľky je možné vidieť príslušné hodnoty FPS, charakterizujúce rýchlosť behu aplikácie. Podobne ako pri predošlom teste sa vždy jedná o mediánovú hodnotu niekoľkých nezávislých behov aplikácie, pričom z každého behu bola vybratá mediánová hodnota FPS. Z nameraných dát vyplýva, že už pri počte virtuálnych svetelných zdrojov 64 a nižších sa hodnoty začínajú pohybivať okolo 15 FPS, čo síce nepredstavuje úplne plynulé zobrazenie, aplikácia s takouto hodnotou framerate však určite umožňuje interaktivitu a relatívne bezproblémový pohyb v scéne. Zaujímavé je tiež si všimnúť, že počet bodov pridelených jednotlivým VPL ovplyvňuje rýchlosť zobrazenia skôr minoritne, podstatne náročnejším ako tvorba je teda použitie ISM pre výpočet viditeľnosti.

Ďalším skúmaným aspektom bola časová náročnosť jednotlivých krokov algoritmu. Poznatky získané týmto meraním môžu poslúžiť ako základ pre prioritizáciu prípadných optimalizácií demonštračnej implementácie. Namerané hodnoty boli získané pri 64 VPL, 8 000 bodoch na VPL a rozlíšení vykresľovacieho okna 800×600 px.

Krok	RSM	ISM	Pull-Push	Render	Disc.	IGB	DS	Blur
t[%]	9.2	1.4	10	12	7.8	27.7	17.6	14.3

Tabuľka 6.3: Približná priemerná časová náročnosť jednotlivých krokov algoritmu.

Jednotlivé kroky sú v tabuľke 6.3 uvedené v poradí za sebou – Tvorba Reflective Shadow Map, tvorba Imperfect Shadow Map, Pull-Push, Render z pohľadu kamery, výpočet discontinuity bufferu, súčet časov potrebných pre manipuláciu G-bufferu (split+gather), deferred shading a napokon blur v oboch smeroch v súčte. Namerané hodnoty predstavujú priemerný pomer času vykreslenia jedného snímku, rozdelený medzi jednotlivé kroky algoritmu. Ako je vidieť, časovo najnáročnejším sa ukazuje rozloženie a spätné zloženie G-bufferu. Z predchádzajúcich testov však vyplynulo, že pridaním tohto kroku je dosiahnuté značné urýchlenie samotného výpočtu osvetlenia. Druhým najnáročnejším je potom deferred výpočet osvetlenia scény. V predchádzajúcom teste som ukázal, že tento krok, ako aj celý algoritmus, je najviac ovplyvnený počtom generovaných virtuálnych svetelných zdrojov. Z toho je možné vyvodiť, že pre optimalizáciu algoritmu je najkritickejšia redukcia počtu VPL použitých pri výpočte.

Kapitola 7

Návrh rozšírenia metódy pomocou VPL clusteringu

Ako jedno z potenciálnych vylepšení/rozšírení – uvádzané aj pôvodnými autormi článku [11] – sa ponúka možnosť VPL clusteringu, teda redukcia počtu vygenerovaných virtuálnych svetelných zdrojov ich zhlukovaním do menšieho počtu zástupných svetelných zdrojov na základe podobnosti vlastností (pozícia, normála/smer pohľadu, farba), pokiaľ možno pri zachovaní obrazovej kvality. Pri vypracúvaní tejto kapitoly som tiež čiastočne vychádzal z článku [3], riešiaceho obdobnú problematiku.

Znížením počtu VPL použitých pri výpočte automaticky znamená zníženie počtu prístupov do textúr RSM uchovávajúcich ich vlastnosti, zníženie počtu výpočtov pohľadových matíc a paraboloidnej projekcie pri nepriamej zložke osvetlenia pre každý jednotlivý pixel výsledného obrazu. Samotná tvorba ISM by týmto bola zasiahnutá menej výrazne, keďže tá je závislá viac na hustote distibúcie bodov aproximujúcich geometriu scény, ako na počte VPL (ťažiskové operácie sú tam vykonávané per-vertex).

Presnosť stratená zredukovaním počtu VPL môže byť čiastočne vykúpená zvýšením rozlíšenia jednotlivých čiastkových tieňových máp, tvoriacich kompletnú textúru ISM zahŕňajúcu viditeľnosť voči všetkým jednotlivým svetlám. Takto napr. štyri čiastkové ISM v rozlíšení 128×128, prislúchajúce VPL tvoriacim jeden cluster, by bolo možné nahradiť jedinou ISM v dvojnásobnom rozlíšení.

Ako ideálne kritérium pre určenie príslušnosti ku konkrétnemu clusteru sa javí vzdialenosť skúmaného VPL od stredu zhluku, táto však sama osebe nieje zďaleka dostačujúca (aj keď je určite najzásadnejšia). V prípade, že by boli takto zhlukované VPL nachádzajúce sa na rohoch, alebo zakryveniach geometrie scény, vznikali by viditeľné nezrovnalosi vo viditeľnosti z pozície stredov jednotlivých clusterov. Je teda žiaduce snažiť sa pokiaľ možno o vytvorenie planárnych clusterov. Preto je nutné pri zhlukovaní zohľadňovať normály, určujúce smer pohľadu jednotlivých VPL, resp. teda uhol normály prislúchajúcej aktuálne skúmanému VPL voči normále stredu clusteru. Obe tieto kritériá (pozícia aj uhol medzi normálami) je nutné zohľadňovať súčasne, celková „vzdialenosť" od stredu clusteru by teda mala byť daná vzťahom podobným:

$$d = w_1 * \Delta p + w_2 * \Delta n,$$

kde w_1, w_2 sú vhodne zvolené váhy určujúce dôležitosť príspevkov jednotlivých zložiek, $\Delta p, \Delta n$ potom čiastkové „vzdialenosti" v polohe a normále. Rozdielnosť vo farbe jednotlivých virtuálnych svetelných zdrojov tvoriacich cluster nepredstavuje až tak zásadný problém

a malo by byť možné finálnu farbu svetelného zdroja tvoreného clusterom určiť váženým priemerovaním, alebo interpoláciou medzi farbami jednotlivých čiastkových VPL.

Pri samotnom zhlukovaní by bolo možné vychádzať z (dnes už klasickej) metódy *k-means clusteringu*. Ako stredy clusterov je na začiatku výpočtu vybraných k vzoriek zo zhlukovaných dát. Následne je pre každú vzorku z množiny dát spočítaná vzdialenosť voči (všeobecne v n-rozmernom priestore danom vektorom parametrov jednotlivých vzoriek) stredom jednotlivých clusterov a vzorka je na základe ich porovnania prisúdená tomu „najbližšiemu". Ako nový stred clusteru je následne zvolený priemer (*mean*) k nemu prislúchajúcich vzoriek a proces sa opakuje, až kým nedôjde k ustáleniu stredov clusterov. Ťažiskovým problémom je v tomto prípade vhodná počiatočná voľba stredov clusterov. Tie je možné zvoliť buďto náhodne, alebo tak, aby boli čo možno najrovnomernejšie rozložené medzi zhlukovanými vzorkami. V našom prípade je vhodná skôr druhá varianta, realizovaná napríklad výberom podmnožiny pixelov z haltonovej textúry tak, aby vybraná podmnožina čo možno najrovnomernejšie pokrývala priestor RSM. Tým by sme sa pokúsili o zabezpečenie najväčšieho rozpýlenia stredov clusterov v rozmere priestorovej vzdialenosti (čo je – ako bolo uvedené vyššie – rozmer, ktorému je pripisovaná najväčšia váha), možného bez znalosti konkrétnych hodnôt atribútov vzoriek. Vďaka vlastnostiam Haltonovej postupnosti je toto jednoducho realizovateľné výberom prvých k jej členov, kde k je požadovaný počet clusterov. Pri každej zmene osvetlenia (teda pri každej zmene RSM, a teda zhlukovaných vzoriek), je potom ako počiatočné rozloženie možné určiť buďto ustálené stredy z poslednej známej stabilnej konfigurácie clusterov (čo môže byť postačujúce v prípade dostatočne malej zmeny), alebo naštartovať celý proces clusteringu od začiatku. Opäť vďaka vlastnostiam rozloženia bodov generovaných Haltonovou postupnosťou dosiahneme podobné rozloženie príslušnosti jednotlivých vzoriek ku clusterom ako pri poslednej stabilnej konfigurácii (teda najčastejšie v časovo predchádzajúcom frame, „blikanie" nepriameho osvetlenia by teda nemalo predstavovať zásadný problém).

V praxi by toto zhlukovanie bolo možné realizovať pridaním dvoch ďalších krokov medzi tvorbu RSM a tvorbu ISM. Prvým z nich by bolo vytvorenie textúr uchovávajúcich vlastnosti virtuálnych svetelných zdrojov (teda dáta, ktoré chceme zhlukovať). Jednalo by sa o tri textúry v rozlíšení textúry kódujúcej Haltonovu postupnosť (teda napr. pre 256 VPL v rozlíšení 16×16), jednu uchovávajúcu world-space pozície, jednu normály a jednu farby VPL, získané vzorkovaním zodpovedajúcich zložiek RSM. Každá vzorka by mala zároveň priradený identifikátor clusteru, ku ktorému prislúcha, pred započatím clusteringu inicializovaný na neutrálnu hodnotu a uchovaný napr. v alfa kanále niektorej z vytvorených textúr. Zároveň by v tomto kroku bola vyčlenená skupina počiatočných stredov, napríklad prvých $k = n/4$ VPL, kde n je celkový počet virtuálnych svetelných zdrojov. Tieto by boli reprezentované obdobnými textúrami ako vzorky. Každý stred by zároveň uchovával počet VPL, prislúchajúcich k jeho clusteru, inicializovaný na nulu a uchovaný opäť napríklad v alfa kanále niektorej z textúr (pre jednoduchosť v tom istom, v ktorom vzroky uchovávajú identifikátor „svojho" clusteru).

Ďalším krokom by bol potom samotný clustering. Vstupom by boli textúry predstavujúce zhlukované vzorky a aktuálne stredy, výstupom prevážené stredy a vzorky s priradenými novými identifikátormi zhlukov (stredov). Je pritom možné postupovať iteratívne „ping-pong" metódou (podobne, ako bolo popísané pri aplikácii pull-push algoritmu na ISM), kde výstupy každej nepárnej sú zároveň vstupmi každej párnej iterácie a naopak.

Výsledné textúry obsahujúce pozície, normály a priemerované farby prislúchajúce jednotlivým zhlukom by potom okrem využitia pri tvorbe ISM tiež na vstupe výpočtu nepriameho osvetlenia scény nahradili Haltonovu textúru a pôvodnú RSM, čím by sa navyše pri

vzorkovaní ušetril jeden prístup do textúry (výber textúrovacích súradníc RSM z textúry kódujúcej Haltonovu postupnosť) pre každý pixel výsledného obrazu. Svetelný príspevok z jednotlivých zhlukov (intenzita a farba) by bolo nutné váhovať vzhľadom na počet VPL, ktoré ho tvoria. Tento krok by nemal predstavovať priveľkú záťaž pre celý zobrazovací algoritmus, keďže sa jedná o veľmi malé textúry a vzhľadom na pomerne rovnomerné počiatočné rozloženie stredov clusterov je možné rátať s nie príliš vysokým počtom iterácií vedúcich k ustáleniu a určeniu finálneho rozloženia zhlukov.

Kapitola 8

Záver

V rámci prác na tomto diplomovom projekte som sa oboznámil so v súčasnosti používanými metódami výpočtu globálneho osvetlenia v 3D grafických scénach, a to ako s klasickými, tzv. offline metódami, tak aj s metódami snažiacimi sa o prácu v reálnom čase a teda použiteľnosť v interaktívnych aplikáciách. V úvode práce som stručne zhrnul princípy, z ktorých v súčasnosti najpoužívanejšie metódy výpočtu nepriameho (resp. globálneho) osvetlenia vychádzajú.

Hlavným zameraním práce bolo naštudovanie a implementácia efektívneho algoritmu pre výpočet viditeľnosti vzhľadom na sekundárne (tzv. virtuálne) svetelné zdroje, zlúžiace pre výpočet nepriameho osvetlenia scén, pomocou tzv. Imperfect Shadow Mappingu. Druhá a tretia kapitola práce zhŕňajú teoretické východiská tohto algoritmu, ako aj jeho zapojenia do použiteľnej zobrazovacej postupnosti a stručne približujú štruktúru demonštračnej aplikácie. Algoritmus som následne implementoval formou jednoduchej demonštračnej aplikácie a túto implementáciu podrobne popísal v štvretj kapitole práce. Pri práci som sa oboznámil so širokým spektrom techník a algoritmov, zahŕňajúcim (Reflective) Shadow Mapping, Deferred Shading, Paraboloid Shadow Mapping, generovanie a kódovanie pseudonáhodných postupností, či generovanie bodovej reprezentácie scény a tieto, spolu s niekoľkými algoritmami spadajúcimi do oblasti spracovania obrazu (tvorba discontinuity bufferu, separabilný Gaussov filter), vo svojej práci v rámci vyhotovovania demonštračnej aplikácie implementoval.

Súčasťou práce bolo tiež testovanie výslednej obrazovej kvality a meranie výkonnosti demonštračnej aplikácie. Stručne som zhrnul dosahovanú rýchlosť zobrazovania v závislosti na nastavení atribútov algoritmu výpočtu nepriameho osvetlenia, ako počet generovaných virtuálnych svetelných zdrojov, im priradených bodov, či vplyv prekladaného G-bufferu. Pri vhodnom nastavení sa mi podarilo dosiahnuť akceptovateľnú kvalitu výsledného obrazu, zahŕňajúc priame aj nepriame osvetlenie scény, pri dosiahnutí interaktívnej rýchlosti.

Na záver som na základe zhodnotenia doby, potrebnej pre realizáciu jednotlivých krokov použitého postupu, vybral jeden z jeho najnáročnejších aspektov a pokúsil som sa o stručný popis návrhu možného rozšírenia, resp. zefektívnenia algoritmu pridaním krokov realizujúcich redukciu počtu virtuálnych svetelných zdrojov variantou k-means clusteringu.

Ak by som mal v práci na projekte pokračovať, ako takmer triviálne rozšírenie sa ponúka zahrnutie použitia metódy pre výpočet viditeľnosti vzhľadom k plošným svetelným zdrojom a environment lighting. Okrem pokusu o implementáciu navrhnutého rozšírenia by bolo možné ďalej naštudovať a spracovať view-adaptive podobu algoritmu ISM [10].

Literatúra

[1] Brabec, S.; Annen, T.; Seidel, H.-P.: Shadow Mapping for Hemispherical and Omnidirectional Light Sources. In *Proceedings of Computer Graphics International*, 2002, s. 397–408.

[2] Dachsbacher, C.; Stamminger, M.: Reflective Shadow Maps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2005, s. 93–100.

[3] Dong, Z.; Grosch, T.; Ritschel, T.; aj.: Real-time Indirect Illumination with Clustered Visibility. In *Vision, Modeling, and Visualization Workshop*, 2009.

[4] Halton, J.: Algorithm 247: Radical-inverse quasi-random point sequence. *ACM*, 1964: str. 701.

[5] Keller, A.: Instant Radiosity. *SIGGRAPH '97*, 1997: s. 49–56.

[6] Laine, S.; Saransaari, H.; Kontkanen, J.; aj.: Incremental Instant Radiosity for Real-Time Indirect Illumination. In *EGSR'07 Proceedings of the 18th Eurographics Conference on Rendering Techniques*, 2007, s. 277–286.

[7] Marroquim, R.; Kraus, M.; Cavalcanti, P. R.: Efficient Point-Based Rendering Using Image Reconstruction. *Symposium on Point-Based Graphics 2007*, 2007: s. 189–203.

[8] Osman, B.; Bukowski, M.; McEvoy, C.: Practical Implementation of Dual Paraboloid Shadow Maps. *ACM SIGGRAPH Symposium on Video Games*, 2006: s. 103–106.

[9] P., C.: Dual paraboloid shadow mapping. 2010.
URL `<gamedevelop.eu/en/tutorials/dual-paraboloid-shadow-mapping.htm>`

[10] Ritschel, T.; Eismann, E.; Ha, I.; aj.: Making Imperfect Shadow Maps View-Adaptive: High-Quality Global Illumination in Large Dynamic Scenes. *Computer Graphics Forum*, 2011: s. 2258–2269.

[11] Ritschel, T.; Grosch, T.; Kim, M. H.; aj.: Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. In *Proceedings of SIGGRAPH ASIA 2008*, 2008, s. 27–35.

[12] Segovia, B.; Iehl, J. C.; Mitanchey, R.; aj.: Non-interleaved Deferred Shading of Interleaved Sample Patterns. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware*, 2006, s. 53–60.

[13] Strengert, M.; Kraus, M.; Ertl, T.: Pyramid Methods in GPU-Based Image Processing. In *Proceedings of Vision, Modeling, and Visualiation*, 2006, s. 169–177.

[14] Wald, I.; Kollig, T.; Benthin, C.; aj.: Interactive Global Illumination. In *Proceedings of Eurographics Symposium on Rendering*, 2002, s. 9–20.

Dodatok A

Obsah CD

Priložené CD obsahuje nasledujúcu adresárovú štruktúru

- Application/ – Adresár obsahujúci súbory prislúchajúce k demonštračnej aplikácii.
  - bin/ – Spustiteľná podoba demonštračnej aplikácie (Linux, 32b).
  - data/ – 3D modely a textúry demonštračnej scény použitej v aplikácii.
  - doc/ – Aplikačná dokumentácia vygenerovaná utilitou doxygen.
  - shaders/ – Zdrojové súbory GLSL shaderov.
  - src/ – Zdrojové súbory demonštračnej aplikácie.
  - Doxyfile – Konfiguračný súbor utility doxygen pre vytvorenie aplikačnej dokumentácie.
  - Makefile – Súbor Makefile pre preklad demonštračnej aplikácie
- Report/ – Adresár obsahujúci túto správu a k nej prislúchajúce súbory.
  - fig/ – Adresár obsahujúci obrázky použité v textovej správe.
  - src/ – Textová správa v zdrojovej podobe (LaTeX).
  - DIP-xmarta00.pdf – Textová správa vo formáte pdf.
  - Makefile – Súbor Makefile pre preklad zdrojovej podoby správy.
- README – Stručný popis obsahu CD a demonštračnej aplikácie.

Dodatok B

Manuál

Preklad demonštračnej aplikácie

Ako bolo spomenuté v texte práce, demonštračná aplikácia využíva niekoľko voľne dostupných knižníc. Pre jej úspešný preklad a spustenie je okrem podpory OpenGL vo verzii 3.3 a GLEW nutná prítomnosť knižníc SFML v2.0 (kontext, rendering window, I/O), Assimp v3.0 (načítanie modelov) a DevIL v1.7 (načítanie textúr modelov).

Pre preklad na platforme Linux slúži priložený súbor Makefile.

Ovládanie aplikácie

Vytvorená aplikácia predstavuje interiér Šibenickej katedrály. V tomto je umiestnený jeden animovaný primárny svetelný zdroj typu spotlight. Pohyb kamery v scéne je riešený „štandardne" kombináciou kláves `W`, `S`, `A`, `D` pre pohyb vpred, vzad a do strán. Výšku umiestnenia kamery je naviac možné meniť klávesami `Q` a `E`. Zmena smeru pohľadu kamery je realizovaná pohybom myši.

Klávesami `U`, `I` a `O` je možné prepínať medzi zobrazením len nepriameho, len priameho a kombinovaného nasvietenia scény.

Klávesou `P` je možné zapínať a vypínať zobrazovanie výsledkov niektorých medzikrokov algoritmu v pravom hornom rohu okna aplikácie. Numerickými klávesami prebieha výber zobrazenia RSM, ISM, textúry s Haltonovou postupnosťou, alebo discontinuity bufferu.