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

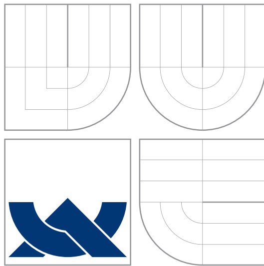
RADIOSITA NA GPU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

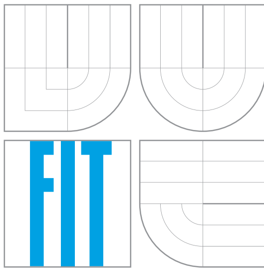
AUTOR PRÁCE
AUTHOR

DAVID ŠABATA

BRNO 2011



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

RADIOSITA NA GPU

RADIOSITY ON GPU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DAVID ŠABATA
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ POLOK
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se zabývá výpočtem osvětlení scény pomocí metody zvané radiosita a její praktickou implementací v knihovnách OpenGL a OpenCL. Nejdříve budou popsány metody pro globální osvětlení s důrazem na radiositu a možnosti jejího použití při vykreslování v reálném čase. Dále budou představeny knihovny poskytující základ pro implementaci. Následovat bude popis implementace v jazyce C++, zhodnocení dosažených výsledku a možnosti vylepšení a dalšího rozšíření výsledné aplikace.

Abstract

This work deals with the radiosity algorith, a global illumination method and its implementation in OpenGL and OpenCL libraries. At first two of the most common global illumination methods will be presented, considering radiosity as the main topic, including its usage in realtime rendering. An introduction to the libraries used will be next, followed by description of the application implemented in C++ language. In the end the findings of this thesis and its possible improvements will be discussed.

Klíčová slova

radiozita, globální osvětlení, GPU, OpenGL, OpenCL

Keywords

radiosity, global illumination, GPU, OpenGL, OpenCL

Citace

David Šabata: Radiosita na GPU, bakalářská práce, Brno, FIT VUT v Brně, 2011

Radiosita na GPU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Poloka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

....
David Šabata
18.5.2011

Poděkování

Tímto bych rád poděkoval Ing. Lukáši Polokovi za jeho odbornou pomoc a cenné informace při tvorbě této práce.

© David Šabata, 2011.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **2**
- 1.1 Cíl práce . . . 2
- 1.2 Struktura dokumentu . . . 2

2 Metody osvětlení scény **4**
- 2.1 Dvousměrná odrazová distribuční funkce . . . 4
- 2.2 Lokální osvětlovací model . . . 4
- 2.3 Zobrazovací rovnice . . . 6
- 2.4 Radiosita . . . 7
  - 2.4.1 Základ metody . . . 7
  - 2.4.2 Konfigurační faktory . . . 8
  - 2.4.3 Řešení radiositní rovnice . . . 10
  - 2.4.4 Modifikace metody . . . 11
- 2.5 Raytracing . . . 12

3 Použité technologie **13**
- 3.1 OpenGL . . . 13
  - 3.1.1 Základní pojmy . . . 13
  - 3.1.2 Vertex Buffer Object, Vertex Array Object . . . 14
- 3.2 OpenCL . . . 15

4 Implementace **17**
- 4.1 Struktura aplikace . . . 17
- 4.2 Reprezentace dat . . . 17
  - 4.2.1 Scéna . . . 17
  - 4.2.2 Model . . . 19
  - 4.2.3 Ploška . . . 19
- 4.3 Příprava scény . . . 20
  - 4.3.1 Maska konfiguračních faktorů . . . 20
  - 4.3.2 Rozlišení plošek . . . 21
  - 4.3.3 OpenGL objekty . . . 22
  - 4.3.4 OpenCL objekty . . . 23
- 4.4 Vykreslovací smyčka . . . 23
  - 4.4.1 Vykreslení do pomocného framebufferu . . . 23
  - 4.4.2 Zpracování na grafickém čipu a distribuce energie . . . 23
  - 4.4.3 Obnovení bufferů a vyhlazení . . . 25
- 4.5 Ukládání a načítání . . . 26
- 4.6 Možná rozšíření . . . 26

5 Závěr **28**

Kapitola 1

Úvod

1.1 Cíl práce

V oblasti trojrozměrné počítačové grafiky jsou stále více kladeny požadavky na co nejvíce realistické zobrazení. Uživatelé už se nespokojí s nejzákladnějšími postupy, které byly definovány a používány v počátcích tohoto oboru, tedy již v 60. letech 20. století [1]. S vývojem trojrozměrné grafiky se totiž vyvíjeli i její uživatelé. Narozdíl od dřívějších akademiků a nadšenců z oboru jsou v současnosti největšími konzumenty grafického obsahu běžní lidé, kteří sledují digitálně upravované filmy a hrají počítačové hry. Tito lidé, mnohdy bez jakéhokoliv technického vzdělání, mají pro hodnocení grafické prezentace jediné kritérium, a sice míru, jak moc počítačem vytvořený obraz odpovídá pohledu lidským okem na běžný svět.

Pro realistické kreslení scény se postupem času vyvinulo množství metod a postupů, řešících různé aspekty problému. Liší se požadavky na reprezentaci prostorových dat, možnostmi a samozřejmě kvalitou vizuální prezentace, úrovní hardwarové akcelerace, ... Ve svém základu ale vycházejí ze základních přístupů, založených na simulaci fyzikálních zákonitostí světla a materiálů.

Tato práce se zabývá jednou ze základních metod globálního osvětlení scény, a sice radiositou. Tato metoda nabízí i přes své nedostatky (zejména neschopnost kreslit lesklé či odrazivé povrchy) velmi dobré výsledky, které lze navíc předgenerovat a při opětovném zobrazení scény je osvětlení ihned připraveno bez nutnosti nového výpočtu. V souvislosti s tímto ale vznikají další omezení, v tomto případě požadavek na neměnnost geometrie scény. Tato i jiné vlastnosti radiosity budou blíže popsány v kapitole 2.4.

Zmíněny budou i možnosti využití této metody při výpočtech osvětlení scény v reálném čase. To se uplatní zejména v průmyslu počítačových her. Zde jsou totiž nejpřísnější požadavky na kvalitu výsledku a současně na rychlost odezvy. A přestože je radiosita (nebo přesněji radiositou předpočítané stíny a osvětlení) v počítačových hrách již delší dobu používaná, na plnohodnotné řešení v reálném čase a v netriviální scéně se stále čeká.

1.2 Struktura dokumentu

Nejdříve představím základní pojmy a principy pro osvětlení vycházející z fyzikálního modelu. Poté detailně popíši radiositní metodu včetně jejích předností i nedostatků. Zmíním také možné modifikace radiositní metody pro zefektivnění jejího výpočtu a srovnám tuto metodu s další zásadní metodou globálního osvětlení, s metodou sledování paprsku. Zkrá-

ceně také představím použité knihovny pro práci s grafickým hardwarem se zaměřením na funkce a postupy využité v této práci.

Ve druhé polovině textu popíši implementaci jednoduché aplikace demonstrující radiositní metodu v praxi. V závěru pak navrhnu další možná rozšíření práce a zhodnotím dosažené výsledky.

Kapitola 2

Metody osvětlení scény

2.1 Dvousměrná odrazová distribuční funkce

Veškerý obraz, který lidské oko vidí je tvořen světlem, které vychází ze světelného zdroje, dopadá na materiál a odráží se do sítnice oka. Z toho plyne nutnost materiálu odrážet alespoň nějaké světlo, aby byl okem viditelný, a také fakt, že odrazivost materiálu je z hlediska viditelnosti klíčovou vlastností. Proto je nutné ji matematicky popsat, z čehož budeme moci dále vycházet při popisu osvětlovacích metod. Tuto problematiku detailně popisuje [2], odkud budou převzaty některé rovnice, a dále také [3], kde jsou uvedeny i jiné modifikace rovnic, a to včetně obrázků.

Dvousměrná odrazová distribuční funkce BRDF (*Bidirectional Reflectance Distribution Function*), zobrazená na obrázku 2.1, popisuje odrazivé vlastnosti materiálu v určitém bodě. Světelný paprsek přichází do bodu x ze směru $\vec{\omega_i}$ a odráží se ve směru $\vec{\omega_r}$. Rovnice 2.1 pak definuje poměr těchto dvou energií – odražené L_r ku příchozí L_i. Ze vztahu je patrné, že energie dopadající na plochu je tím větší, čím menší úhel svírá příchozí paprsek s normálou povrchu $\vec{n}$.

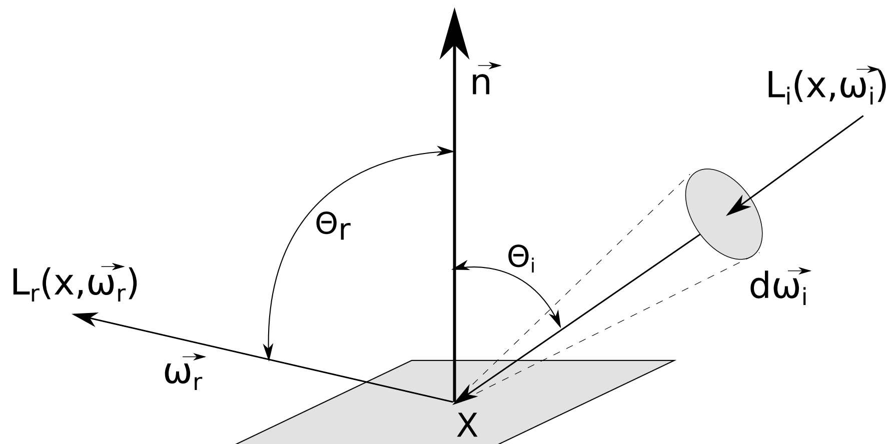
$$f_r(x, \vec{\omega_r}, \vec{\omega_i}) = \frac{dL_r(x, \vec{\omega_r})}{dL_i(x, \vec{\omega_i})(\vec{\omega_i} \cdot \vec{n})d\vec{\omega_i}} \tag{2.1}$$

Ze *zákona zachování energie* plyne, že hodnota BRDF bude vždy menší než 1, případně rovna jedné pro ideálně odrazivé povrchy. Stejně tak funkce nemůže být záporná. Dále pak *Helmholtzův princip reciprocity* říká, že bude vztah platit i pro opačný směr paprsků. Této vlastnosti využívá například metoda globálního osvětlování *raytracing* popsaná v kapitole 2.5. Poslední důležitou vlastností BRDF je linearita, tedy že vyzářený paprsek $\vec{\omega_r}$ je závislý pouze na příchozím paprsku $\vec{\omega_i}$ a žádném jiném. Tato vlastnost pokládá základ lokálním osvětlovacím metodám a umožňuje jim provádět velmi jednoduchý, a tedy i rychlý výpočet.

2.2 Lokální osvětlovací model

Vyjděme nyní z *dvousměrné odrazové distribuční funkce* BRDF, popsané v předchozí kapitole, a pokusme se ji rozvinout.

Světlo, které vnímáme při pohledu na materiál je energie od tohoto materiálu odražená. Abychom získali celkovou odraženou energii pro jeden konkrétní bod, musíme sečíst všechny energie, které do tohoto bodu přicházejí. A jelikož jde o odraz, projeví se zde právě zmí-



Obrázek 2.1: Dvousměrná odrazová distribuční funkce (BRDF)

něná BRDF. Vztah popisuje rovnice 2.2 podle [2], kde Θ vyjadřuje úhel mezi dopadajícím paprskem a normálou povrchu $\vec{n}$.

$$L_r(x, \vec{\omega_r}) = \int_{\Omega} f_r(x, \vec{\omega_r}, \vec{\omega_i}) L_i(x, \vec{\omega_i}) \cos \Theta d\vec{\omega_i} \tag{2.2}$$

Lokálnost tohoto přístupu spočívá v osvětlení jediného bodu objektu (narozdíl od později v textu uvedených globálních metod) a lokální osvětlovací model je základem všech dalších osvětlovacích postupů.

Nejznáměnším příkladem lokálního osvětlovacího modelu je *Phongův osvětlovací model* [4]. Tento model je ovšem empiricky odvozený a přímo nevychází z fyzikální reprezentace světla a jeho chování. Přesto jde o model velmi rozšířený a hojně používaný. Odraz dopadajícího paprsku světla dělí na tři složky:

- *Difúzní složka*, resp. difúzní odraz je takový, který vzniká dopadem světla na difúzní materiál. Takovýto materiál rozptyluje příchozí světlo rovnoměrně do všech směrů, bez ohledu na vstupní či výstupní úhel paprsků. Difúzní materiál je proto při pohledu z různých směrů osvětlen vždy stejně. V reálném světě je difúzní materiál každý matný povrch.

- *Spekulární* neboli lesklý odraz je naopak složka závislá jak na úhlu dopadu světla, tak i na úhlu pohledu na objekt. Nejde ale o skutečný zrcadlový obraz okolí, který by v reálném světě mohl na určitých materiálech nastávat. Spekulární odraz reflektuje pouze světelný zdroj, což je v případě tohoto modelu pouze jediný světelný paprsek.

- *Ambientní světlo*, tedy konstantní osvětlení prostředí. Tato složka nahrazuje světla od různých zdrojů, které by ve skutečném světě na materiál dopadla až po několika odrazech. Takový výpočet by byl náročný, a proto jej tento model nahrazuje pevně danou hodnotou. Na druhou stranu ale jde o zásadní snížení realističnosti výsledného obrazu, jelikož ambientní složka neumí reflektovat například difúzní odraz od velkého objektu v blízkosti, který by se ve skutečném světě na osvětlení vykreslovaného objektu zcela jistě projevil.

Výsledná energie, a tedy i barva bodu je poté dána součtem jednotlivých složek. V případě, že na bod působí několik světelných zdrojů současně, započítá se ambientní složka pouze jednou, zatímco zbylé dvě složky jsou přičítány pro každý příchozí paprsek. Tyto výpočty jsou velmi rychlé a vzhledem ke své jednoduchosti podávají velmi dobré výsledky. Také proto je Phongovo lokální osvětlení implementováno v drtivé většině grafického hardwaru a stalo se jistým standardem pro osvětlení v reálném čase [2].

2.3 Zobrazovací rovnice

Při popisu ambientního osvětlení v předchozí kapitole padla zmínka o složitosti reálného výpočtu, který tato světelná složka nahrazuje. Tento výpočet, tedy vzájemný energetický vztah každých dvou bodů ve scéně, popisuje *zobrazovací rovnice*. Ve skutečnosti totiž dochází k mnohonásobným odrazům světla, které při každém odrazu předá svoji energii povrchu objektu a ten, pokud je odrazivý (jakkoliv, třeba jen difúzně), opět vysílá část světla dál. Vznikají tak sekundární, terciální a další světelné zdroje. Po jisté době budou všechny body scény osvětlené a zbývající energie bude pohlcená; scéna se tedy dostane do ustáleného stavu. V reálném světě toto nastává prakticky okamžitě díky obrovské rychlosti světla. Matematická formulace je ale natolik složitá, že neexistuje obecné řešení a je třeba výsledek aproximovat. I to je ale velmi náročné, a proto ani postupy blížící se fyzikální simulaci osvětlení nepracují se zobrazovací rovnicí v její původní podobě, ale zjednodušují ji.

Rovnice byla poprvé představena v [5]

$$L_o(x,\vec{\omega}) = \int_\Omega f(x,\vec{\omega},\vec{\omega_i}) L_o(x,-\vec{\omega_i}) \cos\Theta d\vec{\omega_i} \tag{2.3}$$

Pro vyjádření vztahu mezi jednotlivými body scény využívá pouze výstupní energii. Výsledná energie počítaného bodu je tedy integrálem příchozích energií ze všech směrů polokoule Ω. V praxi je ale běžnější integrovat přes přispívající plochy, čímž se podle [2] rovnice změní do tvaru 2.4. Grafický vztah ploch znázorňuje obrázek 2.2.

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_S f(x,x'\to x,\vec{\omega}) L_i(x'\to x) V(x,x') G(x,x') dA' \tag{2.4}$$

Novými členy v této rovnici jsou $V(x,x')$, pro popsání vzájemné viditelnosti obou bodů

$$V(x,x') = \begin{cases} 1 & \text{jestliže je bod } x' \text{ viditelný z bodu } x \\ 0 & \text{jestliže box } x' \text{ z bodu } x \text{ viditelný není} \end{cases}$$

a geometrický člen $G(x,x')$, zohledňující vzájemnou pozici a orientaci obou bodů

$$G(x,x') = \frac{(\vec{\omega}\cdot\vec{n'})(\vec{\omega'}\cdot\vec{n})}{\|x'-x\|^2}.$$

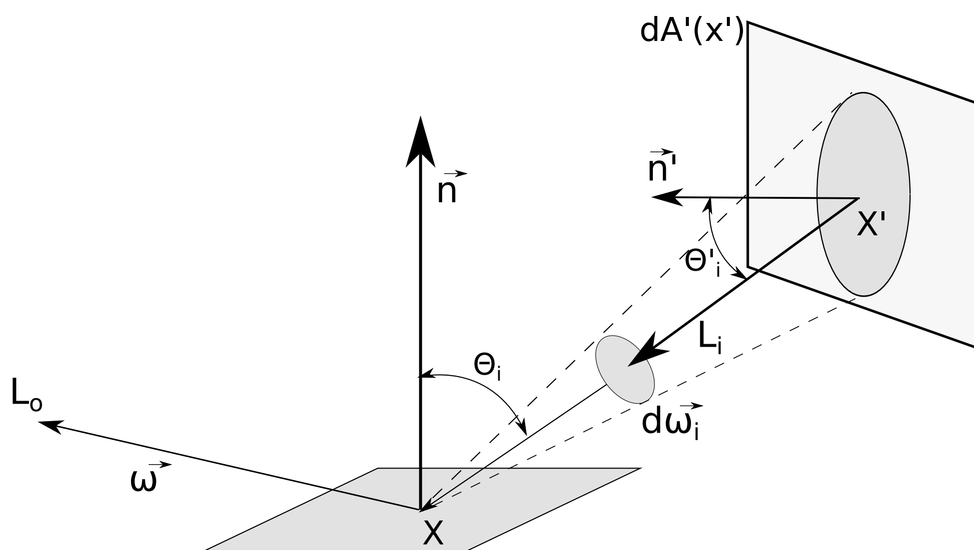
Zobrazovací rovnice tedy obsahuje energie odcházejících ze zdrojových bodů, vynásobené BRDF a zohledňující viditelnost a vzájemnou polohu bodů, a přidává vlastní záření L_e. Poslední úpravou, kterou [2] uvádí, je aplikace integrálního operátoru T a přepis rovnice do tvaru nekonečné řady.

$$L = L_e + TL \tag{2.5}$$

Tento vztah je rekurentní, neboť má neznámou na obou stranách rovnice. Je ale možné jej přepsat do nekonečné řady.

$$L = L_e + TL_e + T^2 L_e + \ldots = \sum_{i=0}^{\infty} T^i L_e \qquad (2.6)$$

Tento zápis opakuje, co již bylo řečeno, a sice že výstupní energie bodu je dána jeho vlastní energií a součtem všech příchozích paprsků, kde mocnina integrálního operátoru značí počet odrazů, které paprsek provedl než dopadl do osvětlovaného bodu. Využití tohoto vztahu bude popsáno v následující kapitole.



Obrázek 2.2: Zobrazovací rovnice pro interakci mezi plochami

2.4 Radiosita

Radiosita je metoda globálního osvětlení scény, představená v roce 1984 [6]. Její základ, stejně jako původní význam tohoto pojmu, pochází z fyziky, konkrétně z přenosu tepelného záření mezi povrchy těles. Pojem radiosita v počítačové grafice označuje fyzikální veličinu značenou $B(x)$, představující intenzitu radiativní energie opouštějící povrch v bodě x. Současně je ale pojem chápán jako konkrétní metoda globálního osvětlení.

2.4.1 Základ metody

Tato metoda vychází ze zobrazovací rovnice (popsána v kapitole 2.3), kterou zjednodušuje zavedením několika předpokladů:

- Scéna je reprezentována rovinnými ploškami, které mezi sebou interagují (přenášení energie). Tyto plošky by měly být stejně velké, jelikož množství energie se vztahuje právě k obsahu plochy. Tento předpoklad také vylučuje použití bodových zdrojů světla. V praxi jsou nejčastěji používány plošky trojúhelníkové nebo čtyřúhelníkové, díky jednoduchosti výpočtů nad nimi a podpoře grafických knihoven a hardwaru při jejich vykreslování.

- Scéna je energeticky uzavřená. Pokud by scéna uzavřená nebyla, energie by „unikala pryč“, což by v lepším případě značně zkreslilo výsledky, v horším případě úplně znehodnotilo celý výstup.
- Povrchy objektů ve scéně jsou neprůhledné a způsobují pouze difúzní odraz světla. V důsledku tedy celá radiositní metoda modeluje pouze difúzní osvětlení.
- Přenos energií probíhá ve vakuu nebo jiném prostředí, které neovlivňuje procházející světlo.

Z výše uvedeného plyne několik výhod, ale i nevýhod radiositní metody. Nevýhody lze často obejít použitím jiné osvětlovací metody nebo ještě lépe kombinací metod. Ne všechny metody ale respektují fyzikální pozadí problému a výsledek v takovém případě nemusí být vždy věrný své reálné předloze. Samotná radiosita ale pro fotorealistické zobrazení nestačí. Většina povrchů ve skutečném světě sice je difúzních, ale existují samozřejmě i lesklé materiály, které dotvářejí výsledný dojem a podporují plastický vzhled objektů. Požadavek na energetickou uzavřenost scény zase znemožňuje použítí radiosity pro venkovní otevřené scény.

Na druhou stranu omezení na difúzní materiály zobrazovací rovnici značně zjednodušuje. A dále energeticky uzavřená scéna, za předpokladu že je také statická, umožňuje radiositní osvětlení předpočítat. Takto předpočítaná data mohou být ve vysoké kvalitě a jejich vykreslení může být velmi rychlé, jelikož bude stačit použít pouze jednoduchý algoritmus pro vyřešení viditelnosti. Osvětlení již bude připraveno a díky difúznímu charakteru světla nebude závislé na úhlu pohledu, tedy na pozici kamery.

Jak dokazuje [7], můžeme při dodržení předpokladů upravit zobrazovací rovnici a dostaneme takzvanou *radiositní rovnici*

$$B(x) = E(x) + \rho(x) \int_S B(x')G(x, x')\, dx' \tag{2.7}$$

Je zde vidět jistá analogie se zobrazovací rovnicí, tedy $E(x)$ představuje vlastní radiositu v bodě x, $\rho(x)$ pak koeficient difúzní odrazivosti materiálu. Funkce geometrického členu zůstává stejná, jako již bylo popsáno. Integrální tvar rovnice ale není ve většině případů použitelný, neboť je jeho analytické řešení možné jen pro velmi jednoduché scény. Nyní využijeme ploškové reprezentaci scény, jimiž budeme aproximovat plochu původního integrálu. Můžeme tedy přepsat vztah do v praxi používané diskrétní podoby

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \tag{2.8}$$

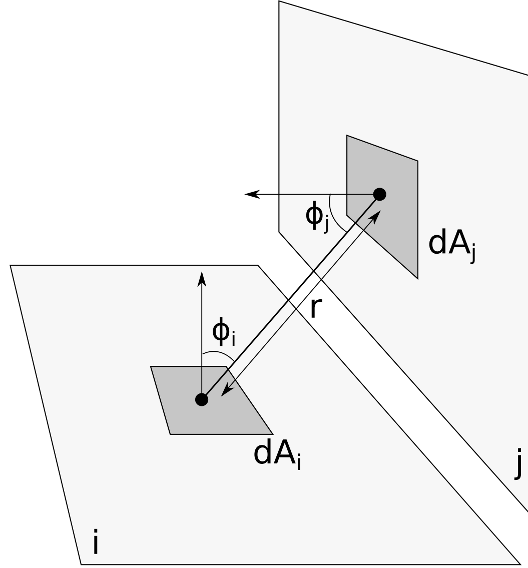
Zásadní změnou v rovnici je, že se energetické výpočty již nevztahují k bodům, ale k ploškám, kde na celé jedné plošce je vždy konstantní energie. Dále také zmizel geometrický člen a byl nahrazen *konfiguračním faktorem* (*form factor*) F_{ij}. Jeho význam je podobný – představuje průměr geometrických faktorů všech bodů obou ploch.

2.4.2 Konfigurační faktory

Vztah pro výpočet faktorů ukazuje rovnice 2.9, graficky pak problém reprezentuje obrázek 2.3. [8]

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos \phi_i \cos \phi_j}{\pi |r|^2} dA_i \, dA_j \tag{2.9}$$

Je vhodné opět připomenout, že stejně jako geometrické faktory ani konfigurační faktory nejsou závislé na pozici pozorovatele. Jde o čistě geometrický vztah, kde záleží pouze na velikosti plošek, jejich vzájemné pozici ve scéně a jejich vzájemné orientaci.



Obrázek 2.3: Konfigurační faktor pro plochy i a j

Před začátkem výpočtu samotného přenosu energií je třeba scénu nejdříve rozdělit, tedy pokrýt ji sítí plošek, a poté pro každou dvojici plošek spočítat konfigurační faktory.

Již samotné dělení ploch je komplexní problém a právě hustota sítě plošek je důležitým faktorem ovlivňujícím kvalitu výsledného obrazu. Pokud by byla síť příliš hrubá, tedy plošky příliš velké, docházelo by k chybám. Jak již bylo řečeno, na celém povrchu plošky je energie, a tedy i barva konstantní – příliš velká plocha by proto narušila plynulost přechodu osvětlenosti povrchu. Stíny by mohly být nesprávně vykreslené, byly by zubaté a hranice jednotlivých plošek by byly zřetelné. Tyto neduhy řeší použití jemnější sítě plošek, kdy při skutečně malých plochách jednotlivé části snadněji splývají a lépe tvoří dojem plynulého osvětlení. Stejně tak stíny jsou při použití malých plošek měkčí a vypadají přirozeněji. Současně s kvalitou ovšem rostou nároky na paměť a prodlužuje se doba výpočtu. V praxi se tak vždy hledá kompromis mezi kvalitou a výpočetním časem.

Základním přístupem k dělení ploch scény jsou takzvané *uniformní metody*, které scénu pokrývají pravidelnou sítí nejčastěji čtvercových či trojúhelníkových plošek. Tyto metody jsou velmi rychlé, ale často neefektivní. Používají totiž stejnou hustotu mřížky pro celý objekt, případně i pro celou scénu, a i místa kde se osvětlení mění jen minimálně jsou pokryty hustou mřížkou. Druhým extrémem je mřížka příliš hrubá, která sice neplýtvá ploškami na velké povrchy na kterých se osvětlení nemění, ale současně nejsou schopny dostatečně jemně pokrýt místa, kde se osvětlení mění rychle – například zákoutí v polostínu.

Tyto nedostatky řeší metoda zvaná *progresivní dělení plošek* (*adaptive refinement*), která v prvním kroku rozdělí scénu velmi hrubě a po provedení průchodu radiositní metody zpětně analyzuje energie ploch. Plošky, které spolu sousedí, ale je mezi nimi velký rozdíl

v osvětlení jsou dále děleny, dokud není dosaženo jisté hranice pro nejvyšší přípustný rozdíl sousedících plošek. Detailní měkké přechody jsou potom skutečně jen v místech, kde jsou potřeba, zatímco na velké plochy se neplýtvá ploškami, resp. výpočetní sílou a pamětí. Plocha rozdělená progresivním dělením je většinou reprezentovaná *kvadrantovým stromem*, tedy dvojrozměrnou verzí *oktalového stromu*, který se běžně používá i pro řešení jiných grafických problémů. [2]

Jakmile je scéna pokrytá vhodnou sítí plošek, je třeba spočítat konfigurační faktor každé dvojice. Jelikož je analytické řešení rovnice 2.9 pro netriviální scény nemožné, přistupuje se k aproximačnímu řešení. Tím je takzvaná *projekční metoda*, využívající *Nusseltovu analogii*. Diferenciální plošku dA_i obklopíme jednotkovou polokoulí. Druhou počítanou plošku A_j poté na tuto polokouli promítneme. A konečně obraz plošky A_j ještě jednou promítneme, tentokrát do základny polokoule. Hodnota konfiguračního faktoru je poté rovná ploše tohoto druhého průmětu.

I toto řešení je ale složité, a proto se v praxi používá promítání na *polokrychli* (*hemicube*). Jde o zjednodušení Nusseltovy analogie, kdy namísto polokoule použijeme polovinu pomyslné krychle, v jejímž středu by byl vyšetřovaný bod. Tato polokrychle je pokrytá sítí plošek, nejčastěji čtvercových, kde každá ploška má předpočítaný konfigurační faktor, takzvaný *delta konfigurační faktor* nebo krátce *delta faktor*. Při promítání vzdálené plochy na povrch polokrychle pak pouze zjišťujeme, které plochy byly průmětem „zasaženy" a celkový konfigurační faktor je pak roven součtu delta faktorů plošek na polokrychli. Přestože se toto řešení může jevit jako složitější, je ve skutečnosti mnohem efektivnější. Předpočítání delta faktorů může proběhnout pouze jednou pro opakovaný výpočet, třeba i několika různých scén. Z tohoto pohledu můžeme předpočítané delta faktory brát jako konstantu, která by při implementaci mohla být přímo součástí programu. Takto předpočítaná data navíc nebudou zásadně paměťově náročná, jelikož díky symetrii polokrychle postačuje k výpočtu pouze čtvrtina čelní stěny a polovina boční [7].

Zásadní výhodou při použití polokrychle je ale možnost využít pro výpočet přímo grafického přístupu. Pokud napozicujeme kameru do bodu diferenciální plošky dA_i a vykreslíme scénu z pohledu této kamery (ve skutečnosti z celkem pěti pohledů – dopředu podél normály plochy a poté jeden pohled pro každou stěnu polokrychle), budou ve výsledném obraze pouze plošky, které mají s vyšetřovanou ploškou nenulový konfigurační faktor. Algoritmus pro určení viditelnosti v grafické knihovně tedy ušetří velkou část výpočtů pro zjišťování faktorů plošek, které se vzájemně nevidí. Při vhodné volbě pohledů kamery a schopnosti odlišit jednotlivé vykreslené plošky pak získáváme velice rychlý způsob výpočtu konfiguračních faktorů všech viditelných plošek vůči právě vyšetřované. Více o praktické implementaci tohoto výpočtu bude popsáno v kapitole 4.3.1.

2.4.3 Řešení radiositní rovnice

Samotným osvětlováním scény, resp. přenosem energií je myšleno řešení radiositní rovnice, představené v úvodu této kapitoly (rovnice 2.8). Za předpokladu, že již známe konfigurační faktory všech plošek ve scéně, zbývá pouze najít energetickou rovnováhu. Jak ukazuje [7], je možné rovnici převést na soustavu n lineárních rovnic o n neznámých, kde neznámými hodnotami jsou právě odcházející energie materiálů, tedy radiosity B.

$$\begin{pmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix} \tag{2.10}$$

Tuto rovnici je pak možné řešit běžnými metodami pro řešení soustav lineárních rovnic. Kupříkladu Gauss–Seidelovou či Jacobiho metodou. [7] současně tyto metody srovnává. Obě metody poskytují srovnatelné řešení. Již o několik let dříve ale představil Michael F. Cohen v [9] metodu zvanou *progressive refinement* neboli *postupující radiosita.*

Zatímco řešení soustavy rovnic vedle své časové náročnosti poskytuje řešení až po dokončení celého výpočtu, postupující radiosita umožňuje získávat průběžné výsledky a tyto zobrazovat. Tato metoda principiálně následuje zobrazovací rovnici 2.6 uvedenou dříve. Funguje iterativně a její myšlenka je taková, že se první fázi vyzáří primární zdroje světla, poté sekundární (tedy první odrazy), terciální (druhé odrazy), a tak dále. Tento postup ale není vždy optimální, a proto metoda důsledně nedbá na číslování odrazů, ale namísto toho vyzáří vždy zdroj s největší energií, ať už jde o primární světlo anebo o některou z již ozářených ploch. Takovéto řešení konverguje k výsledku nejrychleji, jelikož plochy s největší energií také nejvíce přispívají k osvětlení scény. Největší změny v osvětlení se tedy projeví hned na začátku výpočtů, zatímco s postupujícím časem jsou změny stále menší a menší. Proto je také vhodné a v praxi běžné nastavit ukončovací podmínku výpočtu na krok, kdy je již změna ve scéně minimální. Tím se doba výpočtu opět zkrátí, protože není nutné hledat *absolutní* energetickou rovnováhu, ale pouze takový stav, kdy by další šíření energie již scénu měnilo pro lidské oko neznatelně.

2.4.4 Modifikace metody

Stejně jako u jiných vykreslovacích metod se i u radiosity objevilo množství modifikací. Některé zásadní, jiné tak minoritní, že splývají s jinými řešeními. Nejrozšířenější z nich popisuje [2].

Metoda *hierarchické radiosity* se snaží řešit problém výpočtu konfiguračních faktorů plošek. Předpokládá scénu dělenou adaptivně, tedy s vyšší jemností v místech, kde vznikají přechody v osvětlení. Popisuje pak konkrétní situaci, kdy se počítá konfigurační faktor velké plochy a malé plošky v místě, kde je dělení velmi jemné. Pokud bude vzdálenost těchto dvou ploch dostatečně velká, lze nahradit faktory jemně dělených plošek jediným „skupinovým“ faktorem, čímž se výpočet znatelně urychlí. Tento princip ovšem neplatí pro opačný směr, jelikož energie vyzářená velkou plochou bude pro jemně dělené plošky zásadní změnou. (Nezapomínejme, že plošky jsou děleny jemně právě proto, že na nich vzniká stín, a ten je energií z velké plochy jistě ovlivněn.)

Vlnková radiosita zase naráží na omezení při aproximaci scény ploškami. Pokud nejsou plošky dostatečně malé (jak již bylo popsáno dříve), vznikají mezi nimi viditelné hrany a v osvětlení vznikají chyby. Metoda popisuje radiositu plošky lineární kombinací funkcí a namísto jedné hodnoty osvětlení hledá vhodné koeficienty těchto funkcí. Pro scénu při použití této metody stačí použít hrubší dělení a díky funkčním reprezentacím dosahuje shodné či lepší výsledky než klasický přístup.

Další možnou modifikací, byť možná ne v pravém slova smyslu je *paralelní radiosita* popsaná v [10]. Staví na iteračním principu výpočtu s tím, že jednotlivé kroky distribuuje mezi síť výpočetních jednotek. Při vhodně navrženém mechanismu synchronizace je urych-

lení výpočtu zejména pro složité scény zásadní. Samozřejmě by bylo možné a vhodné spojit tuto metodu s jinými modifikacemi k dosažení ještě vyšší efektivity výpočtu.

Pro dosažení výpočtu radiosity v reálném čase se využívá množství postupů, včetně těch, které s radiositou přímo nesouvisí, například algoritmy pro vhodné omezení viditelnosti. Často používanou metodou je již zmíněná metoda progresivního dělení plošek. Další metody zmiňuje [11]. Progresivní dělení používá zejména pro zářiče, zatímco příjemce energie ponechává dělené méně. Využívá také rozdělení scény na statické a dynamické objekty, přičemž se při aktualizaci osvětlení zaměřuje na dynamické s cílem získat vizuálně přesvědčivý plynulý obraz. Poslední zmíněnou modifikací je rozdělení scény do tzv. clusterů, tedy oblastí, čímž je možné lépe lokalizovat změny v osvětlení a dohledat objekty v okolí, které mohly být změnou zasaženy také. Clustery ale nacházejí uplatnění zejména ve větších scénách.

Tyto modifikace jsou bezesporu přínosem. Musíme ale přihlédnout i k faktu, že citovaná práce je již více než deset let stará, a přestože uvedené principy stále platí, některé se neprosadily anebo se od nich s postupem času upustilo. Důvodem je zejména velmi dynamický vývoj grafického hardwaru, díky kterému je často jednodušší počítat v případě dynamické scény osvětlění pro každý snímek znovu bez znalosti předchozích stavů či pomocných struktur pro scénu. Stejně tak nejnovější grafické adaptéry umožňují na svém čipu provádět silně paralelizované negrafické výpočty. Data pro tyto výpočty navíc zůstávají v paměti grafické karty, a tudíž odpadá nutnost jejich přenosu na CPU a zpět. Právě zde byla doposud nejpomalejší část celého procesu. Ne všechny metody a modifikace ovšem můžou z této technologie těžit. Detailněji bude technické pozadí výpočtu na procesoru grafické karty popsáno v kapitole 3.2.

2.5 Raytracing

Raytracing neboli metoda *sledování paprsku* je pravděpodobně nejrozšířenější metodou globálního osvětlení scény. Byla představena v [12] v 1980, tedy o čtyři roky dříve než metoda radiosity. Metoda vychází z obrazu zobrazovaného kamerou, jakožto z mřížky, kde z každého bodu vrhá do scény paprsek. Tento paprsek simuluje chování světla – odráží se, láme se a v případě průhledných materiálů jimi i prochází skrz. Při každém kontaktu s povrchem paprsek sbírá informace, které ve výsledku tvoří barvu získaného bodu. Právě možnost rozštěpení paprsku je významná změna oproti metodě zvané *vrhání paprsků*, která pracuje pouze s prvním zásahem povrchu.

Výhoda sledování paprsku je očividná – metoda umožňuje vykreslovat všechny druhy materiálů, vytváří skutečné odlesky včetně zrcadlových odrazů a podporuje i průhledné objekty. Metoda je navíc ve své základní podobě jednoduše implementovatelná a díky separátnosti jednotlivých paprsků také snadno paralelizovatelná. Kvalitu výsledku lze ovlivnit několika faktory, zejména rozlišením (počtem paprsků vržených do scény) a počtem sledovaných odrazů. Je také možné selektivně vrhat větší množství paprsků pro místa, kde požadujeme vyšší úroveň detailu.

Na druhou stranu je ale metoda pohledově závislá, a je tedy nutné pro každou pozici kamery vrhat nové paprsky. Současně metoda není čistě fotorealistická, jelikož plně neimplementuje zobrazovací rovnici. Není schopná následovat odrazy mezi difúzními materiály, stejně jako difúzně přenášet barvu (narozdíl od radiositní metody). Je také omezená pouze na bodové zdroje světla. Tyto nedostatky zčásti řeší množství modifikací, které díky oblíbenosti metody sledování paprsků vznikly. Jejich popis již ale přesahuje rámec této práce.

Kapitola 3

Použité technologie

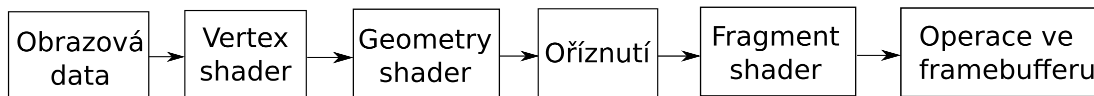
3.1 OpenGL

Open Graphics Library neboli *OpenGL* je multiplatformní aplikační rozhraní pro jednotný přístup ke grafickému hardwaru. Tato knihovna je šířena pod volnou licencí a mimo svoji nativní implementaci v jazyce C nabízí možnost použít téměř kterýkoliv z běžně používaných programovacích jazyků. Je důležité poznamenat, že OpenGL pracuje pouze s grafickým hardwarem a tudíž neumí vytvářet okna aplikace, reagovat na vstupy například od myši, klávesnice či jiné události. To je také důsledek přenositelnosti mezi platformami, kde může každý systém využít vlastní prostředky na práci s okny i ostatními zařízeními. Je potom na programátorovi, aby zajistil spolupráci OpenGL a okenního či jiného systému. Existují ale i nádstavby OpenGL, které jej doplňují o nejčastější úkony, především vytvoření okna a podporu vstupu z klávesnice a myši. Příkladem takového rozšíření je knihovna *GLUT* [13]. V této práci jsem však použil čisté OpenGL v kombinaci s *Win32API* (tj. okenní systém pro platformu Windows), konkrétně ve verzi 3.0 s dopřednou kompatibilitou. Budu tedy popisovat zejména postupy a součásti, které jsem při implementaci využil, přičemž budu čerpat zejména z [14]. Podstatným rozdílem od předchozích verzí jsou programovatelné bloky zpracovávající data od jejich prvotního seskládání až po závěrečnou rasterizaci. Princip programování takzvaných *shaderů* bude popsán v příslušné kapitole.

3.1.1 Základní pojmy

Počáteční objekt v OpenGL je takzvaný *kontext*, který vizuálně znázorňuje oblast obrazovky, do které bude OpenGL kreslit. Vnitřně kontext znamená konkrétní prostředí pro práci v OpenGL. Obsahuje odkaz na okno, do kterého bude kreslit, rozšíření která se mají načíst, nastavení pro reprezentaci dat a jiné. Dalším klíčovým prvkem je tzv. *framebuffer*, což je část paměti, obsahující právě jeden zobrazovací snímek. Zobrazuje již rasterizovaný obraz a jeho reprezentace je podobná bitmapě. Velikost této paměti, potažmo jednoho obrazového pixelu je závislá na barevné hloubce, použití průhlednosti a reprezentaci samotné barvy. Velikost celého framebufferu je pak samozřejmě závislá i na jeho rozměrech, tedy rozlišení obrazu. Pro plynulé zobrazování zejména nestatických scén se používá technika zvaná *double–buffering*, která využívá dva framebuffery. Jeden je zobrazován, zatímco do druhého se na pozadí kreslí. Jakmile je vykreslování hotovo, framebuffery se během jediného snímku vymění a zobrazí se nový obraz, zatímco původní, nyní skrytý framebuffer se vyprázdní a kreslí se opět do něj. Díky tomuto postupu nedochází k postupnému kreslení do již zobrazeného výstupu a nevznikají nepříjemné artefakty či problikávání obrazu.

Postup vytváření obrazu od počátečních vrcholů až k výsledné rasterizaci popisuje tak-zvaný *vykreslovací řetězec* (*OpenGL pipeline*) na obrázku 3.1.



Obrázek 3.1: Vykreslovací řetězec (pipeline) OpenGL

V obrázku jsou tři významné bloky – *vertex shader*, *geometry shader* a *fragment shader*. Tyto bloky postupně zpracovávají obrazová data v různých formách – nejdříve vrcholy, poté jednotlivé body. Důležité ovšem je, že jsou tyto bloky plně pod kontrolou programátora, který vytváří program v dialektu jazyka *C*. Ten se v rámci inicializace grafické aplikace zkompiluje a nahraje do grafické karty. Takovéto programy, tzv. shadery, poté zajišťují například transformace vrcholů a objektů, výpočet osvětlovacího modelu, texturování objektu či jiné operace. Starší grafický hardware tento postup nepodporoval a bloky a jimi prováděné operace byly do velké míry statické.

Na počátku řetězce stojí obrazová data. Jde především o souřadnice vrcholů a vazby mezi nimi. Následovat mohou barvy, souřadnice textur, normály povrchů či jiné údaje použité například k pozdějším výpočtům. Tato data se předají vertex shaderu, který se spustí jedenkrát pro každý vrchol. Ten může s vrcholem provést transformaci, změnit jeho barvu či souřadnice jeho textury. Nemůže však vrchol odebrat či naopak vytvořit nový a je omezen i tím, že dostává vždy pouze jediný vrchol, přičemž celou strukturu polygonu nebo objektu nezná. Vrcholy se následně seskládají do primitiv (nejčastěji trojúhelníků) a předají se opět po jednom geometry shaderu. Ten tedy operuje vždy nad jedním primitivem a umí jej rozšířit o další vrcholy. Takto funguje například teselace, kdy do grafické karty přicházejí modely v omezené kvalitě kvůli zachování malé velikosti a až před samotným vykreslením se v geometry shaderu dopočítají detaily. Dalším krokem je ořezání zobrazovaného prostoru podle aktuálního nastavení kamery a pohledového objemu. Zde plně postačuje parametrizace. Poté se každý objekt, který ve scéně po ořezání zůstal, rozloží na primitiva a ta následně na jednotlivé body neboli fragmenty. Fragment shader pro každý bod určí jeho barvu, případně jiné parametry jako třeba průhlednost. Zdrojem mu může být například bod v textuře (jehož souřadnici získá od vertex shaderu), použitý materiál (ten může vzejít přímo z aplikace jakožto parametr shaderu) nebo i souřadnice světelného zdroje, který na povrch dopadá. Právě fragment shader tedy implementuje například již zmíněný osvětlovací model. Jeho výstupem je rastr, se kterým může framebuffer dále pracovat, například jeho části nezobrazovat na základě hloubkového či jiného testu.

3.1.2 Vertex Buffer Object, Vertex Array Object

Vertex Buffer Object (*VBO*) je jednoduchá obálka nad blokem paměti, ve kterém jsou uložena data potřebná k vykreslení obrazu. Programátor je však od úložiště odstíněn. Při vytváření bufferu může pouze „napovědět“ grafickému systému, zda se jedná o statická data, často upravovaná dynamická data, data používaná přímo pro kreslení či pomocná data pro vedlejší výpočty. Konkrétní umístění dat poté vybírá OpenGL na základě dostupných pamětí v počítači s ohledem na programátorem zadný účel těchto dat, stejně jako na pozadí zajišťuje přesun těchto dat do grafické paměti při aktivaci bufferu.

Je důležité zdůraznit, že VBO neobsahují žádnou informaci o typu uložených dat. (Vý-

jimkou je speciální buffer pro uložení informací o propojení jednotlivých vrcholů.) Sémantiku datům dodává až jejich konkrétní způsob použití. Programátor pro každý vertex shader zadává vstupní data, která přestože mohou být vložena přímo, jsou nejčastěji nahrazena vertex buffer objectem. Součástí definice vstupu shaderu je také typ (a tím i velikost) jednoho prvku v bufferu a počet prvků, které se předají jedné instanci shaderu. V případě souřadnic vrcholů by mohly být v VBO uloženy hodnoty typu `float`, kdy shader dostane právě tři po sobě jdoucí hodnoty, které pak může převést na vektor a s ním dále pracovat.

K vykreslení objektu je tedy třeba aktivovat příslušné shadery a nastavit jim odpovídající vstupy. Těchto kombinací však může být větší množství, stejně jako existují další nastavitelné parametry, například zdroj textur pro kreslený objekt. A jelikož by bylo zdlouhavé a nepřehledné nastavovat tyto parametry před každým kreslením, existuje objekt zvaný *Vertex Array Object* (*VAO*), který tato nastavení zapouzdřuje. V praxi se pak při inicializaci programu vytvoří VAO pro každý „typ kreslení“ a samotné vykreslení objektu pak spočívá pouze ve výběru shaderů a příslušné sady nastavení, tedy VAO.

3.2 OpenCL

Open Computation Language neboli *OpenCL* je průmyslovým standardem pro hardwarově i platformně nezávislé paralelní programování. Potřeba standardizace přišla v době, kdy již nebylo možné zásadně zvyšovat výkon výpočetních jednotek a začalo se prosazovat jejich množení. I programovací postupy proto musely přistoupit k paralelním výpočtům a neexistoval nástroj, který by tyto postupy univerzálně sjednotil. V následujícím textu budu vycházet především z [15].

OpenCL používá tzv. *hostitelský systém*, ke kterému je připojeno jedno nebo více zařízení podporujících OpenCL. Dále předpokládá, že každé zařízení je složeno z *výpočetních jednotek* a každá tato jednotka obsahuje *procesní elementy*. V průběhu paralelního výpočtu poté každý procesní element odpovídá jednomu vláknu, přičemž tato vlákna pracují nad n-rozměrným prostorem, kde každé vlákno zpracovává předem definovaný rozsah dodaných n-rozměrných dat.

Podobně jako u shaderů v OpenGL je i zde programátorem dodán program, který se po zkompilování nahraje do paměti zařízení, odkud se spouští. Pro tento program se používá termínu *kernel*. Opět jde o verzi jazyka *C*, konkrétně o jazyk podle standardu ISO 9899:1999, častěji označovaný jako *C99*. Tento jazyk je rozšířen o některé specifické identifikátory a funkce, zejména pro lepší podporu paralelního programování, například atomické operace.

OpenCL také definuje několik typů pamětí – paměti globální, přístupné všem instancím kernelu, paměti konstantní, plněné spouštějící aplikací a z pohledu kernelu pouze pro čtení, paměti lokální, přístupné pouze skupině instancí, a paměť privátní, sloužící pouze jediné instanci. Zdrojem dat jsou pak paměťové buffery podobné bufferům z OpenGL. Vstupem jsou nejčastěji konstantní či globální paměti, výstupem pak téměř vždy globální.

Spouštějící systém spustí výpočet tak, že nastaví vstupní data a zadá „mřížku“ ve které budou instance kernelu nad daty pracovat. Mřížka tedy tvoří pomyslné hranice oblastí v n-rozměrném prostoru, které připadnou vždy jedné instanci kernelu. Dále odstartuje výpočet vložením příkazu do příkazové fronty OpenCL. Tyto příkazy se totiž ve skutečnosti vykonávají asynchronně a volající aplikace se může nechat informovat o průběhu pomocí ukazatele na obslužnou funkci anebo může počkat na dokončení (vyprázdnění fronty) a poté pokračovat ve zpracování výstupních dat kernelů.

Důležitým přínosem OpenCL pro grafické aplikace je možnost provádět výpočty cíleně na grafickém čipu (namísto obecného OpenCL zařízení) nad datovými objekty OpenGL. Takto lze velmi rychle upravovat obrazová data (operacemi, které jsou paralelizovatelné) aniž by se prováděl přenos z grafické paměti k CPU a zpět. Prostor vstupních dat pak může znamenat například dvojrozměrnou texturu. Konkrétní implementace a přínos pro radiositní metodu bude popsán v následující kapitole.

Kapitola 4

Implementace

Cílem této práce bylo implementovat radiositní metodu s co největším využitím grafického procesoru (*GPU*). Vzhledem k tomu, že jsem neměl žádné dřívější zkušenosti s radiositou či jinou metodou globálního osvětlení, rozhodl jsem se implementovat tuto metodu bez jakýchkoliv modifikací, pouze s využitím promítání na polokrychli. Přesto jsem se ale v průběhu vývoje snažil o co největší rozšířitelnost pro případ budoucího experimentování. Využil jsem již popsané knihovny OpenGL a OpenCL, pro vytvoření okna a dialogů pak rozhraní Win32API. Závislost na operačním systému Windows je ovšem minimální a spočívá skutečně pouze ve vytvoření okna, do kterého se vykresluje, a dialogů pro načtení či uložení scény. Přenositelnost by bylo možné zajistit drobnou úpravou jen několika málo funkcí.

Výsledná aplikace po spuštění obsahuje neosvětlenou scénu a uživatel může spustit či pozastavit šíření energie. Může též zobrazit náhledový kříž, ve kterém lze pozorovat pohled do scény z plošky, která právě vyzařuje svoji energii. Stejně tak lze přepínat mezi zobrazením energií již vyzářených a energií, které vyzářeny teprve budou. Je také možné zobrazit drátěný model scény, kde je patrné rozdělení na plošky a následně na jednotlivé trojúhelníky. Scénu lze v průběhu či po dokončení výpočtu uložit a tento stav poté opět načíst. Díky tomu je možné počítat osvětlení pro složité scény postupně, stejně jako načíst již hotovou scénu například pro účely prezentace.

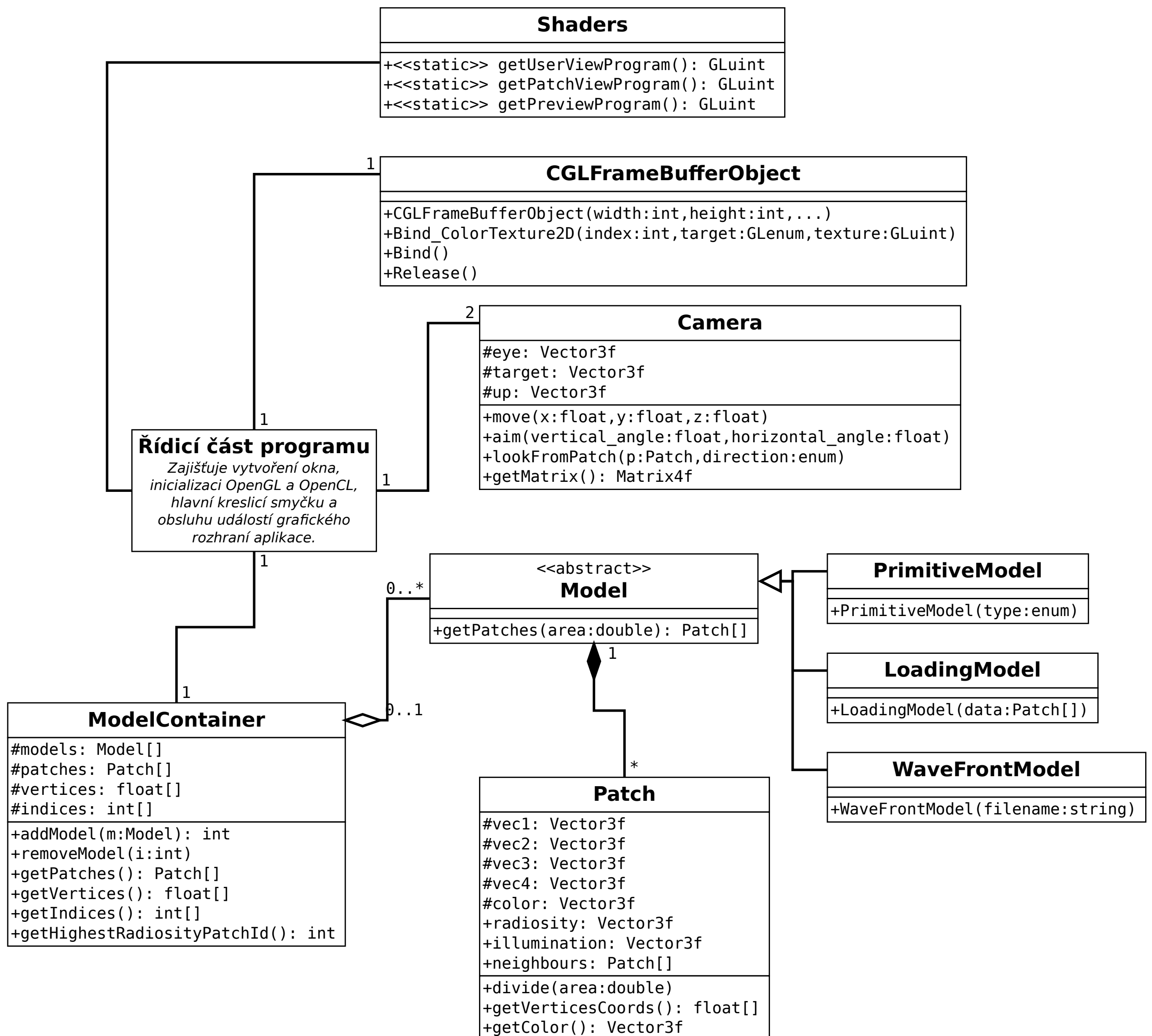
4.1 Struktura aplikace

Základní části programu ukazuje obrázek 4.1. Tento diagram tříd obsahuje pouze významné třídy a jejich metody. Nepokrývá vazby na OpenGL, OpenCL či soubory a třídy s pomocnými funkcemi. Obsahuje také *jádro aplikace*, které není tvořeno třídou, ale pouze souborem funkcí, jehož nejdůležitějším prvkem je kreslicí smyčka. Detailnější funkce bude popsána později, po představení okolních částí programu. Návrhová metodologie tedy není čistě objektová, a to především z optimalizačních důvodů. V místech, kde je to vhodné, je ovšem objektovost dodržena. Jde zejména o objekty reprezentující scénu.

4.2 Reprezentace dat

4.2.1 Scéna

Přestože je scéna běžně tvořena několika modely, kreslicí smyčka s ní pracuje jako s celkem. Jednotlivé modely schraňuje *kontejner* (třída `ModelContainer`) s běžnými operacemi pro vložení a odebrání modelu a operacemi nad scénou jako celkem, například získání pole všech



Obrázek 4.1: Diagram významných tříd a jejich metod

vrcholů ve scéně. Kromě modelů kontejner uchovává i pomocné struktury pro konkrétní vrcholy, jejich vazby a plošky z nich vytvořené. Jde o princip *cache*, kdy si sám kontejner hlídá platnost svého obsahu. Ve chvíli, kdy je přidán či odebrán model, je pouze nastaven příznak identifikující nutnost přegenerovat pomocná data. Tato operace obnáší průchod všemi modely a sesbírání jejich plošek a z těchto dále získání vrcholů a vazeb. Tyto vrcholy a vazby již odpovídají trojúhelníkům, se kterými OpenGL nativně pracuje. Přístup k jednotlivým ploškám je zachován, ale do pomocných struktur se každá rozdělí vždy na dva trojúhelníky pro urychlení pozdějšího vykreslování. K přegenerování ale dochází až ve chvíli kdy je k datům kontejneru poprvé přistupováno. První přístup je tedy výpočetně náročnější, ovšem všechny další přístupy k datům jsou velmi rychlé. K datům kontejneru je navíc poprvé přistupování již v inicializační části aplikace, tudíž nehrozí, že by přepočet pomocných dat ovlivnil průběh aplikace při samotném kreslení nebo výpočtu radiosity.

Třída kontejneru také obsahuje jednu pomocnou metodu (`load()`) pro sestavení ukázkové třídy. Aplikace totiž neřeší načítání scén definovaných externími soubory či modely, jelikož toto nebylo hlavním cílem práce. Doplnit kontejner o možnost načítat externí scény by znamenalo nahradit tělo této metody parserem nějakého konfiguračního souboru a následným vytvářením instancí vhodných tříd modelů.

4.2.2 Model

Model ve scéně je reprezentován některým z potomků abstraktní třídy `Model`. Třída `Model` sama o sobě zajišťuje pouze rozhraní pro získání plošek modelu a implementuje metodu pro rozdělení plošek modelu. Tato metoda ale pouze deleguje pokyn k rozdělení na jednotlivé plošky a nově vzniklé plošky sbírá do pole. Konkrétní potomek této třídy může sám rozhodnout, zda bude při požadavku na vrácení plošek tuto metodu před samotným vrácením dat volat nebo ne. Tím je možné implementovat i takové modely, na které se nebude vztahovat jemnost dělení nastavená ve zbytku scény.

Ukázkovým potomkem třídy `Model` je třída `PrimitiveModel`, která se umí prezentovat jako jedna ze čtyř součástí tzv. *Cornell Boxu.* Cornell Box je fyzicky sestavená a důkladně zdokumentovaná zkušební scéna pro testování grafických programů. Její historický i technický popis je v [16]. Třída na základě parametru konstruktoru vrací geometrii modelu boxu bez jeho čelní stěny, samostatné čelní stěny, krychle anebo kvádru. Data těchto objektů jsou v souboru pevně zadána, stejně jako barvy materiálu jednotlivých ploch.

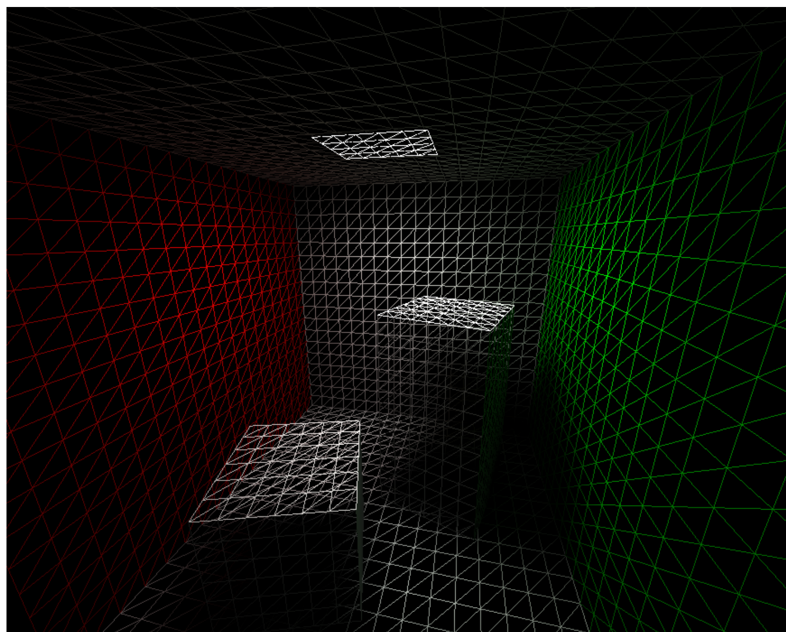
Dalším potomkem je třída `LoadingModel`, sloužící jako pomocný model pro načítání dříve uložené scény. Jelikož soubor uložené scény obsahuje pouze data plošek, je nutné použít tuto třídu přijímající v konstruktoru pole plošek, které pak opět sama vrací. Princip ukládání a načítání scény bude popsán později v kapitole 4.5.

Poslední realizací třídy `Model` je `WaveFrontModel`, tedy model načtený ze souboru ve formátu *WaveFront* s příponou `obj`. Tato implementace je ovšem pouze experimentální a v současné podobě neumožňuje plnohodnotné načtení externího modelu v tomto formátu. Ukazuje však jakým způsobem je možné přidávat další libovolné formáty souborů s modely.

4.2.3 Ploška

Ploška je základním stavebním kamenem modelů. Ať už je původní reprezentace modelu jakákoliv, pro potřeby radiositní metody je nutné, aby byl aproximován rovinnými ploškami. Plošky jsou čtyřúhelníkové a pro správnou funkci metody by měly mít všechny plošky ve scéně ideálně stejný obsah. Stejně tak by měly být plošky nejlépe čtvercové anebo alespoň s poměrem stran co nejblíže jedné. Objekt zná souřadnice svých vrcholů, svoji barvu, odrazivost, své energie a má ukazatele na své nejbližší sousedy. (V diagramu tříd jsou opět zmíněny jen nejpodstatnější vlastnosti.)

Samotná ploška se také umí rozdělit na několik dalších. Právě zde na nejnižší úrovni tedy probíhá dříve zmiňované pokrytí scény sítí plošek. Parametrem k této operaci je nejvyšší povolený obsah každé nově vzniklé plošky. Podle cílené velikosti a znalosti svých rozměrů ploška určí na kolik dílů se bude ve kterém směru dělit a vrcholy nově vzniklých plošek získá bilineární interpolací. Současně novým ploškám ukládá informace o sousedních ploškách. Ta bude užitečná v závěrečné fázi, kdy bude třeba interpolací barev mezi sousedícími ploškami scénu jimi rozdělenou opět vizuálně spojit.



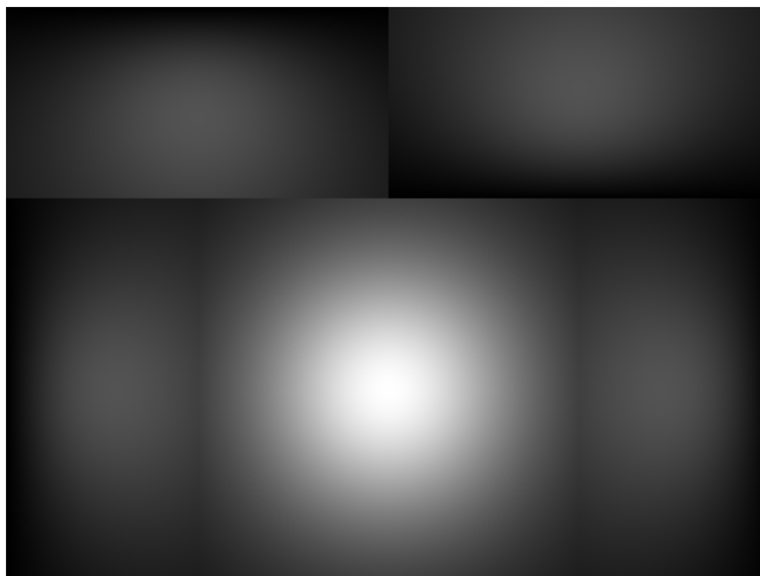
Obrázek 4.2: Drátěný model již osvětlené scény názorně ukazuje síť plošek

4.3 Příprava scény

Při úvodní jednorázové inicializaci se připravují všechny později používané objekty, včetně pomocných. Dynamická alokace většího množství paměti totiž způsobuje v kreslicí smyčce znatelné zpomalení, a proto všechna probíhá již před začátkem vykreslování.

4.3.1 Maska konfiguračních faktorů

Aplikace používá k výpočtu konfiguračních faktorů promítání na polokrychli, které již bylo dříve popsáno. Stejně tak bylo řečeno, že lze delta faktory jednotlivých plošek polokrychle předpočítat a uložit za využití souměrnosti krychle. Přesně takto aplikace postupuje s výjimkou toho, že nevyužívá souměrnosti. Nejdříve předpočítá konfigurační faktory čelní a jedné boční stěny a poté tyto hodnoty překopíruje do textury o stejném rozlišení, jako bude mít kamera umisťovaná do zářících plošek. Aplikace totiž vykreslí všech pět (čelní + 4 boční) pohledů do jediné textury a z této poté odečítá zasažené plošky. Jednotlivé pohledy jsou uspořádány tak, že čelní pohled je uprostřed, boční jsou po jeho stranách a pohledy nahoru a dolů jsou u horního okraje textury. Předpočítané konfigurační faktory používají totožné rozložení a vypočtenou hodnotu reprezentují v textuře barvou v odstínu šedi. Díky tomu bude výpočet výsledného faktoru v pohledu obnášet pouze nahlédnutí do textury a přičtení nalezené hodnoty. Ve srovnání s přístupem, který využívá symetrii polokrychle, je tento mírně paměťově náročnější. Jelikož má ale předpočítaná textura malé rozlišení, je zvýšení paměťové náročnosti prakticky neznatelné. Samotný výpočet faktorů pro plošky v pohledu pak bude mnohem rychlejší, protože nebude třeba pomocí podmínek hledat v textuře bod symetricky odpovídající vyšetřovanému.



Obrázek 4.3: Předpočítané konfigurační faktory uložené v textuře

4.3.2 Rozlišení plošek

Aby mohla distribuce energie probíhat, je nutné umět odlišit každou zasaženou plošku, tedy plošku na kterou zářící plocha vidí a může ji osvítit. Pro tento účel se využívá toho, že *kontejner scény* vytvořil z jednotlivých modelů pole všech plošek ve scéně. A právě index plošky v tomto poli aplikace používá jako její unikátní *ID*.

Pro vykreslení tohoto ID, které je číselné, je potřeba jej transformovat na barvu. To zajišťuje statická třída `Colors`. Nejdříve je třeba nastavit jí parametry, a sice počet bitů pro každou barevnou složku. Rozsahy barev totiž nemusejí být vždy stejné a záleží pouze na konfiguraci framebufferu a textury, do které vykresluje. Dalším parametrem je minimální počet unikátních barev, které aplikace potřebuje pro vykreslení všech plošek ve scéně. Třída podle tohoto čísla a počtů bitů vygeneruje odpovídající masky a offsety pro vytvoření barvy a její zpětný převod. ID převedené na barvu pak odpovídá 32bitovému číslu ve formátu odpovídajícímu některému z vnitřních reprezentací barev OpenGL. Konkrétní formát využívaný aplikací je `GL_UNSIGNED_INT_2_10_10_10_REV`, tedy 10 bitů pro reprezentaci každé z barev.

Převod ID $\rightarrow$ barva a zpětný barva $\rightarrow$ ID pak zajišťuje dvojice velmi rychlých funkcí. Ty díky vhodně vytvořeným maskám a offsetům při vytváření barev používají nejdříve nejvyšší bity a až později přecházejí k nižším (je-li to vůbec potřeba). Scéna je pak při vykreslení unikátních barev plošek dostatečně světlá a plošky nemají tendenci barevně splývat. Tento postup je ovšem doplňující a je zřejmé, že by rozlišení jednotlivých ploch fungovalo i v případě, že by se jejich barvy lišily jen minimálně, pro lidské oko neznatelně. Současný postup ale vytváří názornější odlišení plošek, které se může hodit například při prezentaci či praktickém studiu této práce a implementace radiositní metody.

Při zpětném převodu barvy vrácené grafickou knihovnou se místy objevily drobné nepřesnosti. Ty jsou způsobeny vnitřní reprezentací barev datovým typem `float`, tedy číslem s plovoucí řádovou čárkou. Při převodu zpět do celého čísla je před následným převodem barvy zpět na ID potřeba přičíst korekci. Tato korekce je stejně jako masky a offsety předpočítaná a provádí zaokrouhlení výsledného čísla na očekávanou hodnotu.

4.3.3 OpenGL objekty

Pro veškeré vykreslování aplikace používá již zmiňované Vertex Buffer Objects (VBO) a Vertex Array Objects (VAO), viz kap. 3.1.2. První dva VBO jsou triviální a fungují jako úložiště vrcholů a úložiště vazeb mezi nimi. Další VBO obsahuje unikátní barvy pro jednotlivá ID plošek. K jeho naplnění používá již popsané metody třídy `Colors` a obsahuje vždy pouze jeden interval barev. Toto je důležité pro případy, kdy je ve scéně více plošek než kolik unikátních barev je schopná třída vytvořit. Barvy se z důvodů úspory paměti a zejména přenášených dat neopakují. Správné rozlišení většího množství plošek je řešeno až při samotném kreslení a bude popsáno později.

Následující dva VBO slouží k uložení radiativních a iluminativních energií plošek. Radiativní energie je v této terminologii energie, kterou ploška vyzáří do scény a tím ji osvětlí, zatímco iluminativní energie je osvětlenost plošky samotné, tedy barva plošky při výsledném vykreslení. Tyto energie jsou před započetím samotného výpočtu nulové s výjimkou objektů označených jako světla. Ty již na počátku mají svoji vlastní energii, kterou budou vyzařovat. Pro potřeby radiositní metody je buffer pro radiativní energie nepotřebný. Je ale užitečný opět při zkoumání podstaty metody samotné, kdy je možné výpočet pozastavit a díky tomuto bufferu zobrazit zbývající množství energie ve scéně, které se ještě vyzáří, a sledovat jejich úbytky.

Poslední pomocný VBO obsahuje data pro náhledový kříž. V něm je možné zobrazit pohled vykreslený z plošky která právě vyzařuje svoji energii. Opět jde o funkci vhodnou pouze pro lepší pochopení metody a pro samotný výpočet postačuje s tímto pohledem pracovat vnitřně a není nutné jej vykreslovat uživateli.

Tyto buffery se poté použijí ve Vertex Array objektech, které budou definovat jejich vazby na jednotlivé vstupní atributy shaderů. Hlavní VAO používá geometrii scény (tedy buffer vrcholů a buffer vazeb) a dále připojuje buffer s iluminativními energiemi plošek. Tento VAO tedy reprezentuje běžný pohled do scény, který bude vykreslován uživateli a v případě požadavku na zobrazení radiativních energií bude nahrazen buffer iluminativních energií bufferem druhým.

Pro vykreslení pohledu z plošky slouží pole Vertex Array objektů, kde každý VAO definuje právě jeden interval plošek, kdy velikost tohoto intervalu odpovídá počtu dostupných unikátních barev. Tento VAO totiž namísto bufferu s energiemi vytváří vazbu na buffer naplněnými jedním rozsahem unikátních barev. Je tedy nutné, aby obsahoval i jemu odpovídající rozsah plošek. Je důležité říci, že pracuje se stále stejnými VBO definujícími geometrii scény a pouze vhodně nastavuje ukazatel na první kreslený prvek. Díky tomuto rozdělení do intervalů nebude grafická paměť obsahovat duplicitní data a bude možné kreslit jednotlivé intervaly odděleně.

Nakonec je definován jeden VAO pro vykreslení náhledového kříže. Zde jsou pouze použity dříve definované buffery s geometrií tohoto objektu. Definice v odděleném VAO umožní jeho snadné samostatné vykreslení.

Jsou také definovány a zkompilovány shadery. Jmenovitě shader pro vykreslení uživatelského pohledu do scény, shader pro vykreslení pohledu z plošky a nakonec shader pro vykreslení náhledového kříže. První dva zmíněné neobsahují žádnou pokročilou logiku. Poslední z nich pouze transformuje body z textury v dříve uvedeném rozložení (obrázek 4.7) do podoby kříže neboli rozbalené polokrychle.

V závěru inicializace OpenGL objektů je vytvořen framebuffer, tedy struktura již dříve popsaná, do které lze vykreslovat. V tomto případě je ale framebuffer skrytý (kreslení pro-

bíhá mimo obrazovku) a jeho výsledný obraz je zapisován do textury. Jde o texturu, kterou uživateli může vykreslovat shader náhledového kříže a jejíž body odpovídají konfiguračním faktorům v předpočtené textuře o stené velikosti a rozložení.

4.3.4 OpenCL objekty

Příprava OpenCL objektů obnáší mimo jiné vytvoření kontextu, který vychází z kontextu OpenGL. Díky tomu bude možné sdílet s OpenGL některé objekty, například texturu, do níž se vykreslil pohled z plošky. Dále následuje inicializace bufferů, tedy pamětí, se kterými bude ať už vstupně či výstupně pracovat kernel. Jeho vstupy budou předpočítané konfigurační faktory a objekt textury, o jejíž inicializaci a naplnění se stará OpenGL. Výstupem budou dvě pole – pole ID plošek a pole součtů faktorů pro každou z plošek které v pohledu jsou. ID a součet faktorů budou vždy na shodných indexech. Třetím výstupem bude číselná hodnota značící délky těchto dvou polí. Dále je do kernelu přidáno makro pro převod barvy na ID plošky s pomocí masek a offsetů vypočtených třídou `Colors`. Nakonec je kernel zkompilován, a tedy připraven k použití.

4.4 Vykreslovací smyčka

Vykreslovací smyčka je hlavní částí programu a je velmi žádoucí, provést její cyklus co nejrychleji. V klidovém stavu, tedy pokud neprobíhá výpočet osvětlení, pouze vykresluje uživatelský pohled do scény, případně i náhledový kříž. Uživatel může scénou volně procházet a právě plynulost výsledné prezentace se odvíjí od rychlosti kreslicí smyčky. Tato rychlost je měřená v zobrazených snímcích za sekundu (*FPS*) a je zobrazována v záhlaví okna. Při výpočtu radiosity doba potřebná k provedení jednoho průchodu stoupá, čímž odezva a plynulost aplikace naopak klesá.

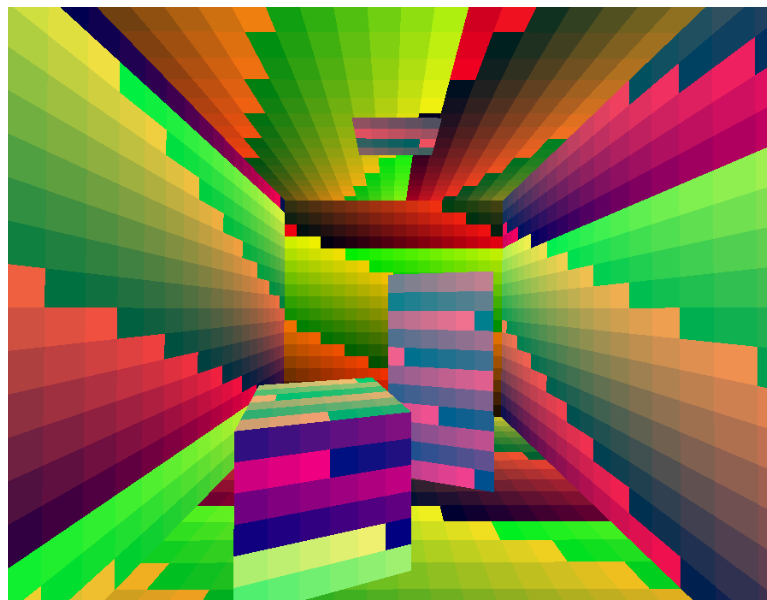
4.4.1 Vykreslení do pomocného framebufferu

Jestliže je spuštěn výpočet, před samotným vykreslením uživatelského pohledu probíhá několik operací. První z nich je vykreslení do pomocného framebufferu. Ten je svázán s texturou, v níž bude výsledný obraz uložen.

Program si nejdříve od *kontejneru scény* vyžádá ID plošky s nejvyšší radiativní energií. Následně vyčistí framebuffer (kde mohl zůstat předchozí obraz) a umístí kameru do středu vybrané plošky, směřující pohledem souběžně s její normálou. Poté správně nastaví oblast ve framebufferu, resp. v textuře, do které se bude kreslit a také ořez části pohledu, který je pro metodu nepodstatný. Tyto oblasti jsou předpočítány a jejich použitím vzniká textura o rozložení, jaké bylo již popsáno v souvislosti s obrázkem 4.7. Samotné vykreslení používá shader pro pohled z plošky a VAO, odpovídající právě kreslenému intervalu plošek. Následuje zpracování pomocí OpenCL a přenos energií. Tento postup od vyčištění framebufferu pro vyzáření energie se pak opakuje pro každý interval plošek ve scéně. Teprve po zpracování všech intervalů se znovu hledá ploška s největší energií. Takto je možné počítat osvětlení i v rozlehlých scénách.

4.4.2 Zpracování na grafickém čipu a distribuce energie

Jakmile je dokončeno vykreslení pohledu do textury, je třeba data zpracovat. Vykreslená textura zůstává v paměti grafické karty, ale OpenGL dočasně předává její správu do rukou OpenCL. Poté je spuštěn kernel, který texturu paralelně zpracovává.



Obrázek 4.4: Pohled do scény ze zářící plošky. Každá ploška ve scéně má unikátní barvu.

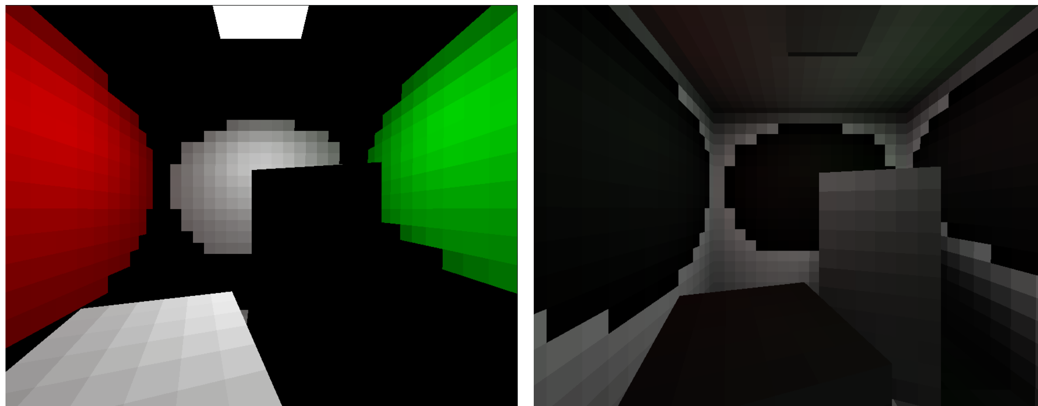
Jak již bylo řečeno dříve, jednotlivé instance kernelu tvoří mřížku. V tomto případě zpracovává každý řádek textury jedna instance, přičemž horizontální dělení určuje délka „úsečky" v textuře. Jestliže bude tato úsečka tvořit třetinu šířky textury, bude počet instancí $3 * vyskatextury$. Každá instance poté zná rozměry textury, délku úsečky připadající každé z nich a také souřadnici v textuře, na které její výpočet začíná.

Každá instance postupuje horizontálně po svém řádku, dokud nenarazí na konec úsečky či pravý okraj textury. Každou barvu, na kterou narazí si zapamatuje a dokud je v následujících bodech barva stejná, přičítá pro každý bod odpovídající konfigurační faktor. Ten dohledá v druhé textuře (vytvořené při inicializaci aplikace) na přesně stejných souřadnicích. Jakmile narazí na bod s odlišnou barvou, vyžádá si index, pod kterým zapíše do výstupního pole součet konfiguračních faktorů a ID původní barvy a proces opakuje opět pro novou barvu.

Synchronizaci instancí zajišťuje právě index pro zápis do dvou výstupních polí. Tato číselná hodnota, stejně jako obě pole i obě vstupní textury, je sdílená všemi instancemi kernelu. Pokud některá z instancí došla na poslední bod některé barvy, provede atomickou inkrementaci tohoto indexu a pod tímto indexem do polí zapíše data. Atomičnost zde zajišťuje, že na stejný index nebude zapisovat žádné jiné vlákno, a jelikož jsou výstupní pole v této fázi využívána pouze pro zápis, není nutné řešit synchronizaci nijak detailněji.

Poté co proběhne výpočet nad texturou se řízení vrací do hlavního programu, který získá ukazatele na pole ID a energií a číslo posledního obsazeného indexu. Použije pomocné pole o velikosti počtu plošek ve scéně, do nějž sečte části energií nalezené jednotlivými instancemi pro každou viditelnou plošku. Je totiž velmi pravděpodobné, že každá ploška byla v textuře vykreslená přes více než jeden řádek. Jakmile jsou energie sečteny, projde program celé pole a pro plošky na které směřuje nenulové množství energie přidá jejich objektům (instance třídy `Patch`) radiativní energii. Tato energie odpovídá energii vyzařované zdrojovou ploškou, snížené úhlem dopadu světla (vliv konfiguračních faktorů) a odrazivostí plošky a ovlivněná také barvou povrchu zdrojové plošky. Právě díky tomu dochází k přenosu barvy, typickému

pro difúzní odraz světla, také nazývanému *color bleeding*. Nakonec je u zdrojové plošky odebrána veškerá radiativní energie a je přičtena k iluminativní. Představuje to stav, kdy se ploška vyzářila (již nemá energii, kterou by mohla předat), v důsledku čehož se zvýšila její osvětlenost při pohledu kamerou, resp. lidským okem.



Obrázek 4.5: Scéna bez vyhlazení zobrazující iluminativní (vlevo) a radiativní energie.

Jelikož při testování mnohdy převyšovala režie spuštění a obsluhy kernelu samotný čas nutný pro výpočet, implementoval jsem rozšíření, které vyzáří několik plošek najednou. Namísto jedné plošky s největší energie se hledá N plošek a pohled z nich se pak vykreslí do jediné textury. Grafický čip poté dostává více dat, a přestože doba zpracování jedné textury lehce vzrůstá, efektivita díky více pohledům roste.

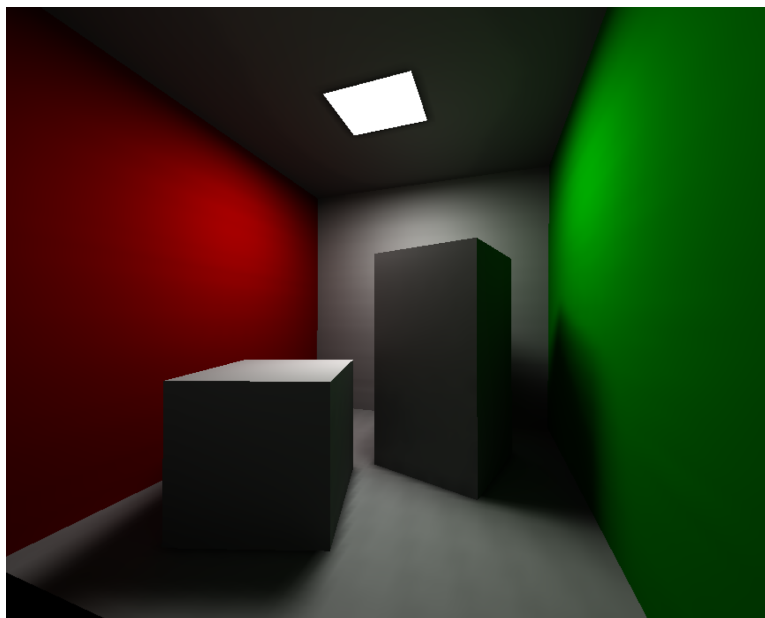
4.4.3 Obnovení bufferů a vyhlazení

Výše popsaný postup od vyhledání plošky s největší energií, až po předání energie všem zasaženým plochám, se v rámci jednoho průchodu kreslicí smyčky v cyklu několikrát opakuje. Počet takovýchto „výstřelů" je parametrizovatelný a zásadně ovlivňuje rychlost výpočtu. Při velkém množství vyzářených plošek během jednoho průchodu kreslicí smyčkou sice dočasně klesne odezva aplikace, ale výpočet se může urychlit až ke zlomkům sekundy.

Po obnovení hodnot v instancích plošek je totiž nutné promítnout tyto hodnoty i do bufferů OpenGL, které budou následně vykresleny. Tato operace, současně s operací aktivace a deaktivace framebufferu, jsou totiž velmi náročné. Proto pokud se provede pouze jediné vyzáření, doba k dokončení výpočtu se značně prodlouží, přičemž po celou dobu výpočtu bude aplikace vytížená, což se projeví snížením její odezvy a plynulosti.

Při samotném vykreslení scény, ať už jde o kteroukoliv fázi výpočtu, je žádoucí vizuálně skrýt rozdělení scény do plošek. K tomuto účelu má každá ploška pole ukazatelů na své sousedy, tedy své osmiokolí. V bufferu barev (tedy iluminativních energií) odpovídá počet prvků počtu vrcholů ve scéně. Každá ploška má tedy čtyři vrcholy, kde každý může mít odlišnou barvu. V případě kreslení plošek v barvách jejich ID jsou všechny čtyři hodnoty shodné. Stejný postup by zde sice vytvořil správně osvětlenou scénu, ale rozdělení by bylo stále znatelné. Proto se poté, co se buffer naplní novými barvami, tyto barvy vyhladí. To probíhá jednoduchým způsobem, kdy barvu vrcholu udává průměr barev všech čtyř plošek, které spojuje. Jelikož se hodnoty iluminativních energií uchovávají v instancích plošek a v bufferu jsou pouze vyextrahované kopie, neovlivní toto vyhlazení další průběh

cyklu výpočtu.



Obrázek 4.6: Výsledná scéna

4.5 Ukládání a načítání

Možnost uložit scénu v průběhu výpočtu či naopak načíst již hotový výsledek přináší zřejmé výhody, přičemž procesy potřebné k uložení a obnovení scény jsou velmi jednoduché.

Scéna se ukládá v podobě plošek, které ji reprezentují a ze kterých je možné zpětně scénu sestavit včetně vypočteného osvětlení. Do binárního souboru se tedy ukládá pole plošek tak, jak je uloženo v *kontejneru*. Jediný problém nastává u sousedností, kde každá ploška obsahuje ukazatele na plošky okolní. Tyto ukazatele se samozřejmě ukončením aplikace stávají neplatnými, a proto jsou před uložením nahrazeny relativními ukazateli. Relativní ukazatel je číslo, které identifikuje plošku v rámci scény, tedy již dříve používané ID.

Při načítání scény je nutné tyto ukazatele opět převést na klasické absolutní. Jakmile je toto hotovo, scéna se vyprázdní, z plošek se vytvoří pomocný model, tedy instance třídy `LoadingModel`, který je pak do scény vložen. Poté opět proběhne inicializace všech objektů a struktur, stejně jako by byla aplikace právě spuštěna.

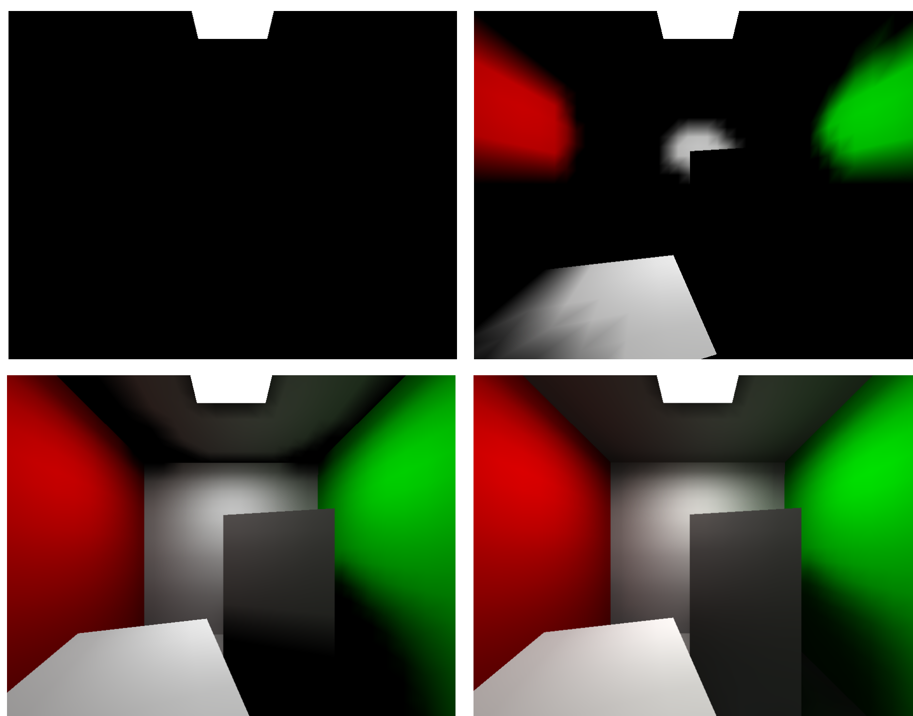
4.6 Možná rozšíření

Přestože vzniklá aplikace v rámci možností naplňuje vytyčené cíle, není ani zdaleka dokonalá. Zde navrhuji několik možných rozšíření:

Dokončit podporu souborů ve formátu WaveFront a přidat podporu i pro modely, resp. scény v jiných formátech. Přestože je tato změna spíše praktická a samotný výpočet osvětlení neovlivní, značně by rozšířila možnosti využití aplikace. V současné podobě lze bez úpravy zdrojových kódů vykreslovat pouze ukázkovou scénu. Současně by bylo vhodné doplnit mechanismus pro načítání nové scény bez nutnosti znovu spouštět aplikaci. Dále by

mohlo být vhodné přidat plošce (třídě `Patch`) informaci o normále. Ta je nyní vypočítána automaticky. Jak jsem ale zjistil při experimentování s modely ve formátu WaveFront, tato normála nemusí být vždy vypočtena správně (může mít opačný směr). V takovém případě selhává jak radiositní metoda, tak i běžné vykreslení pomocí OpenGL, které skrývá polygony s normálou odvrácenou od kamery. Řešením by bylo načítat spolu s vrcholy modelu i jeho normály přímo ze souboru a předpokládat, že modelovací software definuje normály správně.

Znatelné urychlení by mělo přinést využití již dříve popsaného adaptivního dělení plošek. Počet plošek které je třeba vyzářit by klesl, s čímž i výpočetní čas. Kvalita výsledku by ovšem byla srovnatelná, jelikož tato modifikace používá jemné dělení (a tím i kvalitnější výsledek) právě v místech, kde vznikají stíny a přechody v osvětlení.



Obrázek 4.7: Radiosita postupující scénou

Kapitola 5

Závěr

Cílem této práce bylo implementovat jednoduchou aplikaci demonstrující radiositní metodu globálního osvětlení scény. Aplikaci jsem úspěšně navrhl a implementoval. Obsahuje předdefinovanou ukázkovou scénu, tzv. *Cornell Box*, kde je názorně vidět výsledek difúzního osvětlení. Stíny jsou měkké a dochází k přenosu barvy mezi blízkými osvětlenými plochami. Scénou je také možné libovolně procházet.

Výpočet radiositního osvětlení lze ručně spouštět a pozastavovat. Je též možné jej parametrizovat, a to konkrétně počtem vyzářených plošek za jeden průchod kreslicí smyčky, počtem plošek které najednou zpracovává grafický čip, hustotou sítě plošek a rozlišením polokrychle, na kterou se pohled z vyzařující plošky promítá. Změnou těchto parametrů lze lze plynule přecházet od velmi názorného, pomalého a kvalitního výpočtu, až k výpočtu velmi rychlému, prakticky okamžitému. Rychlost výpočtu je samozřejmě vykoupená jeho kvalitou. To, jak moc bude nutné kvalitu omezit pro získání okamžitého výpočtu, záleží vždy zejména na výkonu procesoru a grafického čipu. Při nižších nárocích na kvalitu výsledku ale výpočet probíhá velmi rychle, a je proto možné použít aplikaci pro výpočet osvětlení dynamické scény v reálném čase.

Scénu lze včetně již vypočteného osvětlení uložit a opět načíst. U složitých scén, jejichž osvětlení trvá delší dobu, by bylo možné výpočet pozastavit a pokračovat později. V případě ukázkové scény je ale tato technika smysluplná pouze v případě zvolení velmi vysoké kvality. I tak ale výpočet trvá v nejhorším případě jednotky minut.

Experimentoval jsem také s načítáním scény v otevřeném textovém formátu WaveFront (přípona `obj`). Zde jsem ale narazil na problém s trojúhelníkovými ploškami, které jsou častým výstpem modelovacího softwaru, ale pro aplikaci jsou nevhodné. Současně se také projevil nedostatek v tom, že aplikace sama počítá normály plošek a u některých ploch z externích modelů určuje v opačném směru, než bylo zamýšleno. Řešení tohoto problému jsem navrhl v předchozí kapitole.

Přínosem práce je potvrzení domněnky, že zvyšování výkonu grafických karet velmi výrazně ovlivňujě rychlost i tak složitých osvětlovacích metod, jako je radiosita. Důležité je především zjištění, že i taková v jistých směrech neoptimalizovaná implementace, jakou jsem představil, umožňuje pro jednoduchou scénu vypočíst osvětlení ve zlomku sekundy. Jsem proto přesvědčen, že s neustále rostoucím výkonem bude stoupat i kvalita výsledků, které bude tato aplikace v reálném čase podávat. Obzvláště pokud by se tato práce dále rozvíjela a některé její rysy byly vylepšeny tak, jak jsem popsal v kapitole o možných rozšířeních či v kapitole o modifikacích radiositní metody.

Literatura

[1] Carlson, W. *A Critical History of Computer Graphics and Animation* [online]. 2003 [cit. 2.5.2011]. Dostupné na: <http://design.osu.edu/carlson/history/lesson4.html>.

[2] Žára, J. et al. *Moderní počítačová grafika.* 2. vyd. [b.m.]: Computer Press, 2005. ISBN 80-251-0454-0.

[3] Nechvíle, K. a Pelikán, J. *Úvod do globálního osvětlování: Zobrazovací rovnice* [online]. 2003 [cit. 2.5.2011]. Dostupné na: <http://www.fi.muni.cz/ ptx/PA010/Slides/PA010_12.pdf>.

[4] Phong, B. T. Illumination for computer generated pictures. *Communications of the ACM.* červen 1975, roč. 18, č. 6. ISSN 0001-0782.

[5] Kajiya, J. T. The rendering equation. In *SIGGRAPH '86 Proceedings of the 13th annual conference on Computer graphics and interactive techniques.* 1986. S. 143–150. ISBN 0-89791-196-2.

[6] Goral, C. M. et al. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques.* 1984. S. 213–222. ISBN 0-89791-138-5.

[7] Cohen, M. F., Wallace, J. a Hanrahan, P. *Radiosity and realistic image synthesis.* [b.m.]: Academic Press Professional, Inc., 1993. ISBN 0-12-178270-0.

[8] Owen, G. S. *Overview of radiosity (SIGGRAPH 1993 Education Slide Set)* [online]. 3.8.1998 [cit. 3.5.2011]. Dostupné na: <http://www.siggraph.org/education/materials/HyperGraph/radiosity/radiosity.htm>.

[9] Cohen, M. F. et al. A progressive refinement approach to fast radiosity image generation. In *SIGGRAPH '88 Proceedings of the 15th annual conference on Computer graphics and interactive techniques.* 1988. S. 75–84. ISBN 0-89791-275-6.

[10] Šindlar, L. *Paralelní algoritmy počítačové grafiky.* Praha: MFF UK, 1995. Diplomová práce.

[11] Hrbek Štěpán. *Radiosita v dynamických scénách.* Praha: MFF UK, 2000. Diplomová práce.

[12] Whitted, T. An improved illumination model for shaded display. *Communications of the ACM.* červen 1980, roč. 23, č. 6. ISSN 0001-0782.

[13] *The OpenGL Utility Toolkit* [online]. [cit. 5.5.2011]. Dostupné na: <http://www.opengl.org/resources/libraries/glut/>.

[14] *OpenGL 3.3 Core Profile Specification* [online]. 11.3.2010 [cit. 5.5.2011]. Dostupné na: <http://www.opengl.org/registry/doc/glspec33.core.20100311.pdf>.

[15] *OpenCL 1.0 Specification* [online]. 6.10.2009 [cit. 5.5.2011]. Dostupné na: <http://www.khronos.org/registry/cl/specs/opencl-1.0.pdf>.

[16] *The Cornell Box* [online]. 2.1.1998 [cit. 7.5.2011]. Dostupné na: <http://www.graphics.cornell.edu/online/box/>.