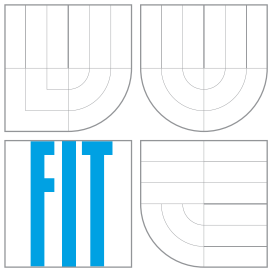
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

OBJEKTOVĚ ORIENTOVANÁ RADIOZITA

TÝMOVÝ PROJEKT DO PŘEDMĚTU PGR

AUTOŘI PRÁCE

David Bařina
Kamil Dudka
Jakub Filák
Lukáš Hefka

BRNO 2007

Objektově orientovaná radiozita

Zadání

1. Navrhněte třídy, které názorně a z hlediska OOP čistě implementují radiozitu.
2. Vytvořte demonstrační příklady - scéna může být zadána kusy kódu, které instanciují vhodné objekty (není třeba načítat ze souboru apod.)
3. Vytvořte nadstandardní dokumentaci (oproti všeobecným požadavkům), která dobře popíše (např. s pomocí grafů) a obhájí strukturu tříd.
4. Důraz je kladen na čistotu OOP návrhu a kvalitu dokumentace.

Obsah

Kapitola 1

Úvod

Radiozita je pokročilá zobrazovací metoda používaná ve fotorealistické grafice. Její výpočet je časově náročný. Tato práce se zabývá jednoduchou implementací výpočtu radiozity, která využívá akceleraci založenou na `OpenGL`.

Pomocí vzniklého programu byly vypočítány vzorové scény, které se do této dokumentace nevešly – lze stáhnout z `http://dudka.cz/rrv`. Kromě toho lze z webu stáhnout krátké video (asi 20 vteřin), které ukazuje, jak se šíří radiozita v průběhu výpočtu.

Nedílnou součástí této dokumentace je dokumentace API a diagramy tříd, které jsou dostupné rovněž na webu projektu.

1.1 Podíl jednotlivých členů týmu na projektu

David Bařina, xbarin02

- Výpočet konfiguračních faktorů pomocí `OpenGL`
- Převod primitiv (koule, válec, čajník) na polygonální reprezentaci

Kamil Dudka, xdudka00

- Objektový model
- Dělení na plošky
- Mezipaměť pro konfigurační faktory
- Interpolace barev

Jakub Filák, xfilak01

- Vstup/výstup XML, DTD
- Zpracování parametrů příkazové řádky

Lukáš Hefka, xhefka00

- Program pro vizualizaci
- Vzorová vstupní scéna

Tento dokument byl vysázen systémem LaTeX.

Kapitola 2

Teoretická část

2.1 Radiozita

Radiozita[7] je metoda globální iluminace scény (šíření světelné energie) používaná k renderování 3D scény v počítačové grafice. Radiozita jako renderovací metoda byla představena v roce 1984 výzkumníky na Cornell University. Vychází ze zákona zachování energie. Proto vyžaduje energeticky uzavřené scény. Nedokáže pracovat s průhlednými objekty, zrcadly a texturami. Scéna musí být reprezentována polygonálním modelem.

Zobrazovací rovnice vychází z dvousměrové distribuční funkce BRDF[4]. Plochy nejen odrážejí světlo, ale mohou mít i vlastní zářivost. Šíří se pouze difúzní odraz světla.

Vlastní výpočet může probíhat buď iteračně (progresivně) nebo řešením soustavy rovnic (maticové řešení). Před vlastním výpočtem je třeba polygony ve scéně rozdělit na malé plošky a spočítat *konfigurační faktory* (vliv každé plošky na každou jinou plošku ve scéně). Plošky, které na sebe nevidí mají konfigurační faktor 0. Iterační výpočet, který je použit v našem projektu, má výhodu postupného zobrazení výsledku po každé iteraci.

Radiozita (zářivost) každé plošky je definována jako:

$$B_i = E_i + R_i \sum_{j=1}^{n} B_j F_{ij} \tag{2.1}$$

kde:

- B_i je radiozita plošky i.
- E_i je vyzařovaná energie této plošky.
- R_i je odrazivost plošky.
- suma reprezentuje součet energií přicházejících na plošku i ze všech ostatních plošek.
- F_{ji} je konfigurační faktor mezi ploškami i a j (vliv plošky j na plošku i).

2.2 Výpočet konfiguračního faktoru pomocí `OpenGL`

Konfigurační faktor[1] (*form factor*) říká, kolik energie energie plošky (*patche*) i je přímo přijato ploškou j. Plošky vzdálenější mají od plošky cílové mají na tuto vliv menší než plošky bližší. Plošky viditelné z cílové plošky ze strany mají menší vliv než plošky ležící přímo před cílovou ploškou (*Lambertův kosinový zákon*). K výpočtu konfiguračního faktoru je třeba vykreslit scénu z pohledu cílové plošky (i) s rozsvícenou ploškou zdrojovou (j). Toto vykreslení (vyrenderování) scény je třeba provést pro všechny dvojice plošek ve scéně. Vykreslením všech zdrojových plošek rozdílnou barvou ke každé cílové plošce najednou lze výpočet značně urychlit. Složitost klesá z kvadratické (počet plošek na druhou) na lineární. Z výsledné vyrenderované bitmapy se pak spočítá form factor pro všechny zdrojové plošky viditelné v této bitmapě k plošce cílové (z jejíhož pohledu byla scéna renderována).

Při vlastním renderovaní[3] je potřeba vidět z pohledu cílové plošky celých 180° světa ležícího před touto ploškou. Dále je třeba aplikovat Lambertův kosinový zákon a tím snížit vliv plošek, které se nacházejí z boku cílové plošky, oproti ploškám ležících přímo naproti cílové plošky. Čím více z boku zdrojová ploška leží, tím méně světla z ní cílová ploška přijme.

K vyrenderováni 180° scény se používá vyrenderováni do tzv. *polokrychle* (hemicube). To znamená vyrenderováni scény při pohledu ze středu patche do přední stěny a do 4 bočních polostěn. Renderuje se s úhlem pohledu 90° (perspektiva). Vyrenderováni přední stěny se kamera umístí do středu cílového patche a bude se dívat ve směru normálového vektoru tohoto patche. K renderování je použito `OpenGL`, konkrétně funkce `gluLookAt()` z knihovny `GLU`. „UP“ vektor je zvolen rovnoběžně s jednou ze stran trojúhelníkového patche.

Při renderování bočních stran se kamera opět dívá ze středu patche a renderuje se opět s úhlem pohledu 90°. Směr pohledu je ovšem kolmý na normálový vektor („UP“ vektor z renderování přední stěny). Pro renderování všech 4 stran se kamera postupně otáčí po 90°. Z výsledné bitmapy je podstatná pouze část nad horizontem cílového trojúhelníku (tzn. polovina boční stěny). Všechna renderování probíhají postupně do jediného framebufferu. Oblast framebufferu, do které se bude kreslit, se nastavuje funkcí `glViewport()`. Pro další zpracování je důležité, aby na sebe pohledy do stran polokrychle správně navazovaly (správné „UP“ vektory). Po dokončení renderování je ve framebufferu zobrazena rozbalená krychle (tvoří kříž). Podstatná je pouze vnitřní část tohoto kříže (bez spodních částí bočních pohledů). V této oblasti je nyní nutné zohlednit Lambertův kosinový zákon.

Velmi jednoduše řečeno, Lambertův kosinový zákon říká, že paprsek dopadající na střed plošky kolmo má na její osvětlení vliv největší. Paprsek dopadající pod nějakým úhlem má vliv menší úměrně tomuto úhlu (kosinus tohoto úhlu). Paprsek dopadající téměř rovnoběžně s povrchem plošky má vliv téměř zcela zanedbatelný.

Před vlastním počítáním konfiguračních faktorů z vyrenderované bitmapy je nutné přenést obsah framebufferu do paměti pomocí funkce `glReadPixels()`. Dále ji bitmapa postupně procházena po pixelech. Barva každého pixelu udává, kterou zdrojovou plošku

v tomto místě cílová ploška vidí (barva pixelu je indexem zdrojové plošky). Pouhým sčítáním počtu pixelů ve scéně dané barvy by se spočítal konfigurační faktor nezohledňující Lambertův kosinový zákon (čím více pixelů ze zdrojové plošky je viděno ploškou cílovou, tím větší mají na sebe vliv). Podle pozice (x, y) v bitmapě je spočten vliv (koeficient) daného pixelu na celkový form factor k dané zdrojové plošce. Největší vliv mají pixely ležící ve středu bitmapy (středu přední stěny polokrychle), kde je tento koeficient roven 1. Nejmenší vliv (koeficient roven 0) mají pixely na vnějších hranách bočních polostěn. Tento koeficient je počítán jako součin kosinů pozice x a y, kde střed bitmapy má x i y rovno nule. Směrem ke stranám se x i y zmenšuje (doleva a nahoru) a zvětšuje (doprava a dolů) až k hodnotě $\frac{\pi}{2}$.

Kapitola 3

Návrh řešení

Výpočet radiozity je časově náročný a klade vysoké nároky na výkon stroje. Hlavními cíli návrhu tedy bylo:

- Co nejvíce urychlit výpočet – využití HW akcelerace, maximální využití dostupné paměti, ...
- Možnost pracovat s mezivýsledky – pro co nejrychlejší odhalení nedostatků ve vstupní scéně nebo zvolených parametrech výpočtu.
- Možnost navázat na dříve přerušený výpočet – v případě výpadku elektřiny, pádu systému, ...

3.1 Reprezentace scény

Výpočet radiozity pracuje s polygonálním modelem – na nejnižší úrovni jsou tedy tělesa ve scéně reprezentována množinou polygonů (trojúhelníků). Jednomu trojúhelníku odpovídá struktura `Triangle`. Na rozdíl od běžných polygonálním modelů však nezadáváme přímo barvu polygonu, kterou vidí uživatel. Tato barva je vypočítána pomocí radiozity. Zadávají se místo toho dvě složky:

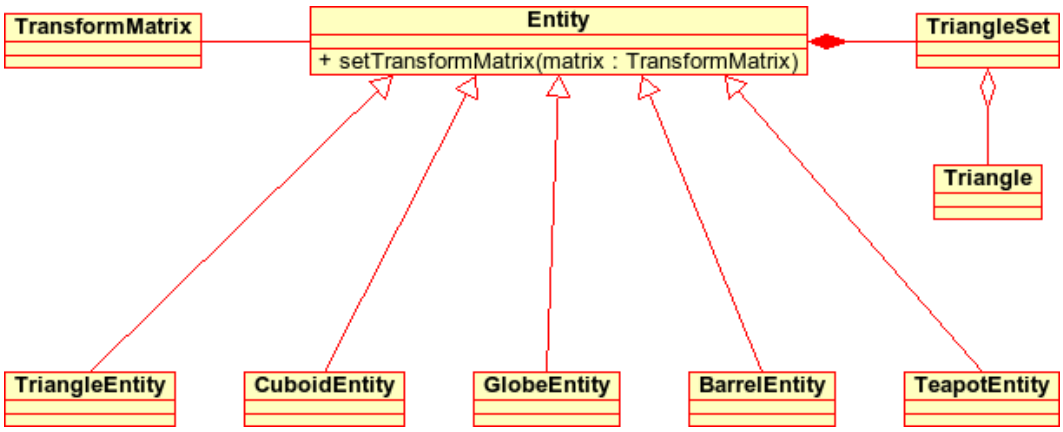
- `emission` – barva, kterou polygon vyzařuje. Používá se pouze pro zdroje světla, tělesa, která sama o sobě nezáří, mají tuto složku nulovou.
- `reflectivity` – barva, kterou polygon odráží. Jedná se tedy o barvu tělesa tak, jak ji vnímáme intuitivně.

Při zadávání scény uživatelem však nemusí být vždy práce s jednotlivými polygony zcela intuitivní. Proto definuje objektový model některá základní primitiva – krychle, koule, válec a čajník[1]. Každé primitivum je reprezentováno samostatnou třídou, která zapouzdřuje převod na polygonální model.

[1] Polygonální reprezentaci čajníku lze stáhnout z http://home.student.uu.se/yuca7825/Ass2_YuCao_YaoWang.zip

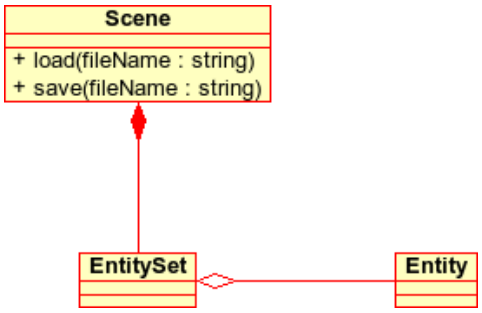
Uživatel má tedy dvě možnosti, jak reprezentovat tělesa ve scéně – polygonálně nebo pomocí primitiv. Tyto přístupy jsou z pohledu výpočtu zaměnitelné – proto je nad nimi definována abstraktní třída `Entity`. Třída `TriangleSet` reprezentuje kontejner polygonů na nízké úrovni.

Nad jednotlivými tělesy je možné provádět běžné transformace ve 3D – k tomu slouží třída `TransformMatrix` a metoda `setTransformMatrix` třídy `Entity`. Část objektového modelu reprezentující těleso je zjednodušeně znázorněna na obr. 3.1.



Obrázek 3.1: Abstraktní třída `Entity`

Scéna je tedy z pohledu uživatele tvořena oddělenými tělesy. Všechny tělesa ve scéně jsou spravovány třídou `Scene`. Tato třída, mimo jiné, umožňuje scénu jako celek načíst ze vstupního XML souboru[2] a zapsat do výstupního XML souboru. Třída `EntitySet` reprezentuje kontejner entit na nízké úrovni. Kompozice třídy `Scene` je znázorněna na obr. 3.2.

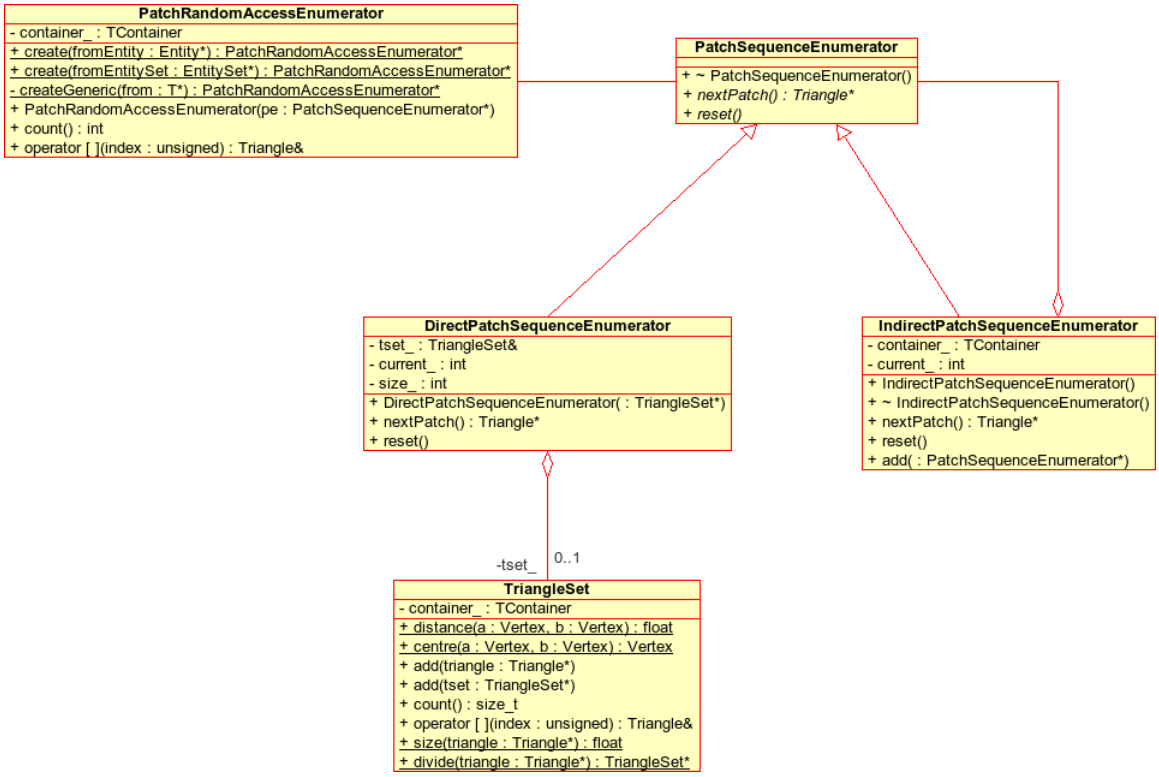


Obrázek 3.2: Třída `Scene` – reprezentace scény

Reprezentace scény pomocí oddělených těles je vhodná pro uživatele a také pro některé pomocné výpočty (např. interpolace barev). Pro výpočet radiozity však tato reprezentace

[2]DTD pro reprezentaci scény je na adrese http://dudka.cz/dtd/scene.dtd

vhodná není. Radiozita uvažuje vliv každé plošky na ostatní plošky – bez ohledu na to, kterému tělesu plošky patří. Pro tento účel byly vytvořeny enumerátory pro přístup k jednotlivým ploškám. Situaci znázorňuje obr. 3.3.



Obrázek 3.3: Enumerátory – třídy pro přístup k jednotlivým ploškám

Základem je třída `PatchSequenceEnumerator`, která implementuje sekvenční průchod přes všechny plošky ve scéně. K indexovanému přístupu k ploškám slouží třída `PatchRandomAccessEnumerator`, která se konstruuje z již existující instance sekvenčního enumerátoru. Tato třída alokuje větší množství paměti, proto je vhodné její instanci sdílet mezi všemi objekty, které vyžadují indexovaný přístup k ploškám.

3.2 Výpočet radiozity

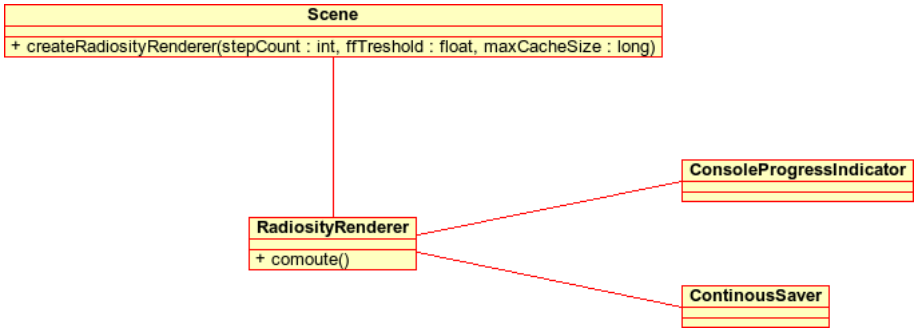
Před vlastním výpočtem radiozity je nutné rozdělit trojúhelníky z vstupní scény na plošky požadované velikosti. K tomu účelu slouží metoda `divide` třídy `Scene`, která odstartuje rekurzivní dělení trojúhelníků. Rekurzivní dělení je vždy zastaveno při dosažení požadované velikosti (obsahu) trojúhelníku. Pro výpočet obsahu trojúhelníku je použitý Heronův vzorec[6]:

$$S = \sqrt{s(s-a)(s-b)(s-c)} \tag{3.1}$$

kde

$$s = \frac{a + b + c}{2} \tag{3.2}$$

Vlastní výpočet radiozity je zajišťován třídou `RadiosityRenderer`. Instance této třídy se tvoří metodou `createRadiosityRenderer` třídy `Scene`, které se předají parametry výpočtu. Vztahy objektů pro výpočet radiozity jsou zachyceny na obr. 3.4



Obrázek 3.4: Třída `RadiosityRenderer` – reprezentace scény

Z pohledu uživatele trvá výpočet radiozity věčnost – je tedy na místě průběh výpočtu sledovat a nějak na něj reagovat. K tomuto účelu byl použit návrhový vzor *observer*, přičemž třída `RadiosityRenderer` je zde v roli sledovaného objektu. V současné verzi projektu jsou implementovány dva observery (pozorovatelé) – `ConsoleProgressIndicator` a `ContinousSaver`. Jednotlivé pozorovatele je možné za běhu připojovat a odpojovat. Třída `ConsoleProgressIndicator` během výpočtu vypisuje do konzole základní informace o průběhu. Třída `ContinousSaver` ukládá mezivýsledky do souboru s předem nastavenou frekvencí. Frekvence ukládání se v průběhu výpočtu snižuje, což odpovídá snižujícím se změnám radiozity ve scéně. Tyto mezivýsledky je možné použít například pro generování videa – viz. kapitola 5.

Jádrem výpočtu radiozity je výpočet konfiguračních faktorů – tento výpočet zajišťuje třída `FormFactorEngine`. Její implementace používá `OpenGL` pro urychlení výpočtu – viz. kapitola 2.2. Tento výpočet je však (zejména pro scény s velkým počtem plošek) náročný a je tedy nejslabším místem aplikace z hlediska výkonu.

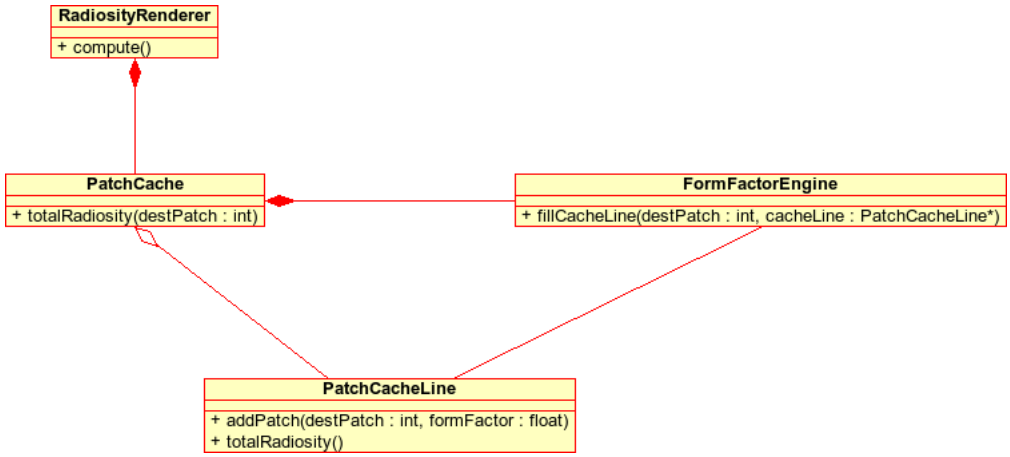
Nabízí se tedy řešení: ukládat si vypočtené konfigurační faktory do paměti mezi jednotlivými průchody. Množství potřebné paměti však roste s kvadrátem k počtu plošek ve scéně a pro netriviální scénu se všechny konfigurační faktory do paměti nevlezou. Rozumným kompromisem je uchovávat v mezipaměti tolik konfiguračních faktorů, kolik se jich tam vleze a to co možná nejefektivnějším způsobem.

Mezipaměť pro vypočtené konfigurační faktory zajišťuje třída `PatchCache`[3]. Tato mezipaměť je hierarchická – konfigurační faktory všech plošek ovlivňujících jednu konkrétní

[3]Spolu s hodnotou konfiguračního faktoru je potřeba uchovávat odkaz na plošku, ke které se konfigurační faktor vztahuje.

plošku se shlukují v objektech třídy `PatchCacheLine`. Počet potřebných objektů `PatchCacheLine` tvořících mezipaměť je zřejmě stejný jako počet plošek ve scéně.

Jednotlivé objekty `PatchCacheLine` spolu potom soutěží o uložení v mezipaměti. Kritériem je přitom počet uložených konfiguračních faktorů uvnitř objektu (a tím pádem velikost objektu v paměti). Vyšší prioritu pro uchování paměti mají objekty s nižší velikostí. Objekty s vyšší velikostí je efektivnější znovu vypočítat. Diagram tříd tvořících mezipaměť je na obr. 3.5. Prioritní fronta je implementována pomocí stejnojmenného STL kontejneru[5].



Obrázek 3.5: Třída `PatchCache` – mezipaměť pro konfigurační faktory

3.3 Vizualizace

Vizualizaci scény má na starosti třída `Visualizer`. Vizualizaci je možné provádět dvěma způsoby:

1. **Interaktivně** – uživatel může měnit pozici kamery, přepínat mezi různými způsoby zobrazení a ukládat aktuální pohled jako obrázek.

2. **Dávkově** – zobrazená scéna se uloží jako obrázek do souboru a program skončí.

Interaktivní vizualizaci zajišťuje statická metoda `visualize`, dávkovou vizualizaci zajišťuje statická metoda `takeScreenshot`. Tyto metody mají jednu zvláštnost – po jejich zavolání se už nikdy nevrátí řízení zpět[4] a program je ukončen funkcí `std::exit()`. Při návrhu je potřeba na tuto vlastnost brát ohled a zajistit správné uvolnění paměti a ostatních zdrojů.

Výsledkem radiozity jsou hodnoty barev pro jednotlivé plošky. Aby uživatel neviděl rušivé přechody barev mezi ploškami, je potřeba provést interpolaci barev. Nejprve jsou průměrováním vypočteny barvy pro jednotlivé vrcholy plošek (metoda `Entity::computeVertexColors()`), samotnou interpolaci potom provádí `OpenGL` během vykreslování scény.

[4]Tohle chování je dáno rozhraním knihovny GLUT, kterou třída používá.

Kapitola 4

Implementace

Celý projekt byl implementován v jazyce C++, využívá knihovny standardu `OpenGL` a knihovnu `GLUT`. Volně šiřitelný parser XML[2] je součástí archivu. Automatické sestavení na různých platformách zajišťuje multiplatformní make systém CMake. Jak bylo naznačeno v návrhu, sestavují se dva spustitelné soubory:

rrv-compute je program pro výpočet radiozity. Výpis parametrů je možné vypsat přepínačem `--help`, povinným parametrem je `--filein` následovaný jménem vstupního souboru. Vstupní soubor může být vytvořený uživatelem, nebo to může být mezivýsledek výpočtu, na který chceme navázat.

Dalším důležitým parametrem je `--divide`, který určuje nejvyšší přípustný obsah plošky ve scéně během rekurzivního dělení na plošky. Čím nižší číslo nastavíme, tím je výstup kvalitnější. Zároveň však roste doba výpočtu, proto je dobré najít vhodnou hodnotu experimentálně.

Výpočet je možné urychlit nastavením vhodné hodnoty parametru `--cache`. Tento parametr určuje maximální prostor vyhrazený pro mezipaměť. Konfigurační faktory, které se do mezipaměti nevlezou je nutné počítat v každém průchodu znovu, což zpomaluje výpočet. V praxi však není vhodné nastavovat velikost mezipaměti vyšší než je velikost fyzické paměti.

rrv-visualize je program pro vizualizaci scény. Pomocí tohoto programu je možné vizualizovat vstupní scénu, jakýkoliv mezivýsledek i finální scénu s vypočtenou radiozitou. Jak bylo zmíněno v návrhu, pracuje ve dvou režimech. Výchozí je interaktivní režim, dávkový režim se zapíná volbou `--screenshot on`. Povinným parametrem je opět `--filein` následovaný jménem vstupního souboru.

Pozici kamery lze v interaktivním režimu měnit myší. Levým tlačítkem se otáčí scéna, pravým tlačítkem se kamera přibližuje/oddaluje a kurzorovými šipkami se kamera posouvá. Dále je možné měnit způsob zobrazení plošek a zobrazení barev pomocí následujících kláves:

Klávesa	Funkce
1	pouze vrcholy
2	pouze hranice polygonů
3	vyplněné polygony
7	zmenšit šířku bodů/čar
8	zvětšit šířku bodů/čar
Z	barva odrazu
X	barva zdroje světla
C	barva odrazu + barva zdroje světla
V	vypočtená radiozita (bez interpolace, barva plošek)
B	vypočtená radiozita (jednoúrovňová interpolace)
N	vypočtená radiozita (jednoúrovňová interpolace, barva plošek)
M	vypočtená radiozita (dvojúrovňová interpolace)

4.1 Sestavení a instalace

Pro úspěšné sestavení je potřeba mít nainstalovanou knihovnu GLUT, knihovny OpenGL, CMake a podporovaný překladač. Vlastní sestavení a instalace jsou velmi jednoduché a přímočaré:

```
$ cmake .
$ make
$ make install
```

Sestavení bylo testováno na operačních systémech Linux, FreeBSD, Windows XP a Windows Vista. Na systémech Windows jsou podporovány překladače MinGW a Microsoft Visual Studio 2005.

Pro výpočet konfiguračního faktoru se používá akcelerátor grafické karty, přičemž číslo plošky se kóduje jako barva. Pokud provádí grafická karta vyhlazování (aplikace vyhlazování sama nezapíná), bude výsledek nesmyslný. **Proto v nastavení grafické karty nesmí být zapnutá volba vynuceného vyhlazování.**

Kapitola 5

Výsledky

Výpočet radiozity byl testován na několika vzorových scénách, které jsou součástí archivu. Vzhledem k omezením na velikost odevzdávaného souboru (a tím pádem i dokumentace) nebylo možné umístit do této dokumentace výsledky jako obrázky. Náhled scény s vypočítanou radiozitou je na obr. 5.1.

Na webu projektu `http://dudka.cz/rrv` je spousta obrázků z průběhu výpočtu. Také jsou tam ke stažení XML soubory s vypočtenou radiozitou, takže je možné si prohlédnout již spočítanou scénu. Kromě toho lze z webu stáhnout krátké video (asi 20 vteřin), které ukazuje, jak se šíří radiozita v průběhu výpočtu.

Obrázek 5.1: Scéna s vypočítanou radiozitou – originální obrázek naleznete na http://dudka.cz/rrv

Kapitola 6

Závěr

Přestože je implementace výpočtu radiozity poměrně jednoduchá, poskytuje pěkné výsledky. Navíc je díky HW akceleraci výpočet rychlý. Konkrétní doba výpočtu závisí na složitosti vstupní scény a zvolených parametrech. Při testování se doba výpočtu pohybovala v rozmezí několika minut až několika dní.

Byly splněny všechny požadavky zadání i hlavní cíle návrhu. Navíc bylo pomocí této implementace výpočtu radiozity vytvořeno zajímavé video. Hlavní výhoda projektu oproti běžně dostupným implementacím však spočívá v jeho jednoduchosti.

Literatura

[1] Ashdown, I.: *RADIOSITY - A Programmer's Perspective*. Wiley, 1994, ISBN 0-471-30488-3.

[2] Berghen, D. I. F. V.: Small, simple, cross-platform, free and fast C++ XML Parser. http://www.applied-mathematics.net/tools/xmlParser.html.

[3] Elias, H.: Radiosity. http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm, 2007.

[4] Sillion, F. X.; Puech, C.: *RADIOSITY & Global Illumination*. 1994, ISBN 1-55860-277-1.

[5] Stroustrup, B.: *The C++ Programming Language*. Addison-Wesley, special edition vydání, 1997, ISBN 0-201-88954-4.

[6] Weisstein, E. W.: Heronian Triangle. http://mathworld.wolfram.com/HeronianTriangle.html.

[7] Wikipedia: Radiosity. http://en.wikipedia.org/wiki/Radiosity, 2007.