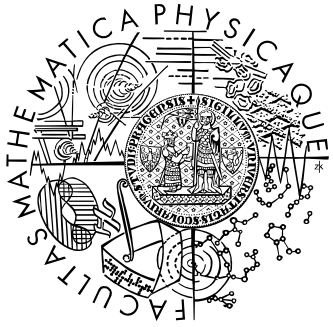
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jiří Matějka

Osvětlovací dílna

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán, KSVI

Studijní program: Informatika, programování

2008

Děkuji vedoucímu práce RNDr. Josefu Pelikánovi za rady a náměty, kterými mi velmi v přípravě práce pomohl.

Prohlašuji, že jsem svoji bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 6.srpna 2008 Jiří Matějka

Obsah

1 Úvod....6
2 Osvětlovací modely....7
2.1 Značení....7
2.2 BRDF....8
2.3 Světelné složky....8
2.4 Více světelných zdrojů....9
2.5 Rozdělení osvětlovacích modelů....10
2.6 Konkrétní osvětlovací modely....10
3 Ray-tracing....12
3.1 Ray-casting....12
3.2 Ray-tracing....13
4 Řešení....18
4.1 Požadavky....18
4.2 Řešení....18
5 Implementace....24
5.1 Požadavky implementace....24
5.2 Platforma, prostředky....24
5.3 Principy implementace....25
5.4 Bajtkód....30
5.5 Multithreading....32
6 Závěr....34
6.1 Shrnutí....34
6.2 Splnění cílů....35
6.3 Diskuze....35
6.4 Další vývoj programu....36
Literatura....37
7 Obsah CD, kompilace programu....38
7.1 Obsah CD....38
7.2 Kompilace programu....38
8 Uživatelská dokumentace....40
8.1 Požadavky....40

8.2 Instalace....40
8.1 Hlavní okno....41
8.2 Okno parametrů....42
8.3 Okno s obrazem....43
8.4 Nastavení....44
9 Syntaxe vstupního jazyka....45
9.1 Obecné informace....45
9.2 Aritmetický výraz....48
9.3 Definice scény....50

Název práce: Osvětlovací dílna

Autor: Jiří Matějka

Katedra (ústav): KSI

Vedoucí bakalářské práce: RNDr. Josef Pelikán, KSVI

E-mail vedoucího: Josef.Pelikan@mff.cuni.cz

Abstrakt: Předmětem práce je navrhnout a implementovat aplikaci pro interaktivní experimenty s lokálními osvětlovacími modely. Uživatel má možnost zadat lokální model odrazu světla pomocí matematických vzorců a posléze měnit jeho vybrané parametry. Program plynule vykresluje jednoduchou 3D scénu s měnícím se osvětlovacím modelem. Scénu je možno definovat v textovém formátu, přičemž několik scén je zahrnuto v distribuci programu.

V distribuci programu jsou zahrnuty dva známé osvětlovací modely – Straussův a Phongův.

Klíčová slova: Ray-tracing, osvětlovací model, symbolický ray-tracing.

Title: Lighting workshop

Author: Jiří Matějka

Department: KSI

Supervisor: RNDr. Josef Pelikán, KSVI

Supervisor’s email address: Josef.Pelikan@mff.cuni.cz

Abstract: The object of this work is to design and implement an application for interactive experiments with local illumination models. The user can arbitrarily define an illumination model with mathematical formulas and mark certain parameters to be changeable. While such parameters are changed, the program smoothly draws an updated 3D scene. The scene can be defined in the text format, while several scenes are bundled with the program.

Two predefined well-known illumination models are also bundled with the program – Strauss' and Phong's model.

Keywords: Ray-tracing, illumination model, symbolic ray-tracing.

1 Úvod

Světlo nám zprostředkovává informaci o okolním světě, a podle toho, jak se nám jednotlivé předměty ve světle jeví, usuzujeme na jejich vlastnosti a strukturu. Důležitým úkolem počítačové grafiky je najít principy pro modelování chování světla při průchodu scénou a při odrazech na povrchu objektů tak, aby bylo možno napodobit skutečné materiály, a výsledek přitom byl věrný realitě. Dojem realističnosti vytváří především osvětlení objektů.

Na následujících obrázcích je vidět, jak rozdílně lze vnímat tentýž objekt vykreslený pomocí různých materiálů.

Obrázek 1: Čajová konvice, zleva: sklo, bez odlesků(křída), kov.
Zdroj: Henrik Wann Jensen, http://graphics.ucsd.edu/~henrik

Realističnost zobrazení materiálu je dána použitým modelem odrazu světla (tzv. osvětlovací model). Bylo vyvinuto mnoho modelů, s různou výpočetní náročností a různou kvalitou. V grafických programech jsou však ve velké většině zadány pevně, jeden nebo omezený počet modelů.

Vytvořili jsme program podporující jakýkoli osvětlovací model, je možno jej totiž zadat symbolicky. Zvolená scéna se vykreslí pomocí daného modelu, a dále je umožněno uživateli měnit vybrané charakteristiky vzorce, a měnit tak vzhled materiálu, kde pro každý objekt ve scéně může být materiál různý. Uživatel tedy může zobrazit jakýkoli materiál, který je použitý osvětlovací model schopen vykreslit.

Druhá kapitola definuje pojem osvětlovací model, rozděluje modely a uvádí příklady některých modelů. Třetí kapitola se věnuje ray-tracingu v obecné rovině. Čtvrtá kapitola představuje požadavky na aplikace a nejdůležitější principy řešení problému, především symbolický ray-tracing. Pátá kapitola obsahuje zajímavé aspekty implementace, přehled architektury systému, definuje bajtkód jako zásadní datovou strukturu.

2 Osvětlovací modely

V této kapitole definujeme pojem osvětlovací model, modely rozdělíme podle několika kritérií a uvedeme některé známé modely.

Osvětlovací model je rovnice, která vyjadřuje, jakým způsobem povrch tělesa odráží světlo.

Lokální osvětlovací model znamená, že vyhodnocujeme odraz pouze jednoho bodu na povrchu tělesa, bez ohledu na okolí.

Pouze světelné zdroje mohou být zdrojem světla, samotná tělesa nikoli (nejsou ani sekundárními zdroji světla). Odraz samotných těles nicméně pro realistický vzhled požadujeme. Vybrali jsme ray-tracing, který toto zvládá (viz kapitolu 3).

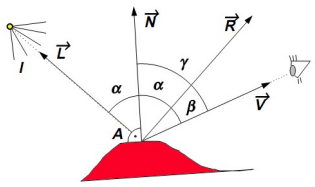
Poznámka. V této práci jsme se omezili pouze na bodové zdroje světla, protože využíváme základní variantu raytracingu.

2.1 Značení

Zde zavedeme značení, které se používá ve vzorcích osvětlovacích modelů. Uvažujeme bod tělesa A, ve kterém vypočítáme osvětlovací model. Ve vzorcích modelů je vyžadováno, aby všechny vektory byly normalizované.

Poznámka. Vektory jsou v této kapitole zapsány tučnou kurzívou.

I	Intenzita zdroje světla
L	Vektor ke světelnému zdroji
N	Normálový vektor tělesa v bodě A
R	Vektor odraženého paprsku (podle zákona odrazu)
V	Vektor k pozorovateli (kameře)
α	Úhel mezi ***L*** a ***N***, stejný jako úhel mezi ***N*** a ***R***
β	Úhel mezi ***R*** a ***V***
γ	Úhel mezi ***N*** a ***V***



Obrázek 2: Situace v bodě vyhodnocení modelu. Zdroj [2]

Značení intenzity. Barvu zdroje světla nazýváme intenzitou a značíme ***I***. Barvu povrchu tělesa značíme ***C***. Výslednou barvu světelné složky značíme ***E***.

2.2 BRDF

BRDF (*bidirectional reflectance distribution function*), neboli *dvousměrová distribuční funkce* určuje odrazivost povrchu tělesa. Říká, kolik světla přicházejícího z jednoho směru se odrazí do směru druhého. Jejími argumenty jsou tedy jednotkové vektory směrů odkud světlo přichází a odkud odchází z povrchu tělesa. Vrací skalár určující množství světla odraženého v tomto směru. Zjistíme-li tuto hodnotu pro každou barevnou složku modelu RGB, máme to, co nazýváme barva. Přesnou definici funkce zde nebudeme uvádět.

Rovnice, kterou zde nazýváme osvětlovací model, je BRDF funkcí. Nadále se však budeme držet pojmu osvětlovací model.

2.3 Světelné složky

Na odražené světlo se obvykle pohlíží jako by bylo složené ze tří složek – ambientní, difuzní, a zrcadlové. Tento přístup zjednodušuje pohled na model. Jako příklad zde uvedeme rovnice pro nejjednodušší Phongův model.

2.3.1 Ambientní složka

Ambientní složka zastupuje mnohonásobné odlesky od okolí. Intenzita složky je stejná pro všechny body scény, bez ohledu na pozici světelných zdrojů. Bývá reprezentována speciálním světelným zdrojem (*ambient light*) a je shodná pro všechny osvětlovací modely. Tak ji budeme reprezentovat i my. Proto se v rovnici modelu neuvádí.

Tvar ambientní složky je

$$E_a = k_a \cdot C_d \cdot I_a$$

kde k_a je koeficient ambientního světla, s hodnotou od 0 do 1, který udává množství okolního světla odraženého povrchem. C_d je barva tělesa. I_a je barva ambientního světla.

2.3.2 Difúzní složka

Difúzní složka (diffuse reflection) vzniká difúzním odrazem. Difuzní odraz je odraz, který světlo rozptyluje rovnoměrně do všech směrů. Dokonalý difuzní povrch

se nazývá *lambertovský* (*lambertian surface*). Takový povrch se nám jeví jako matný, bude vypadat stejně pod jakýmkoliv úhlem pohledu. Z běžných materiálů má podobný vzhled např. křída.

Tvar difuzní složky se u jednotlivých modelů liší, pro nejjednodušší Phongův model je

$$E_d = k_d \cdot I_d \cdot C_d \cdot \cos(\alpha)$$

kde k_d je koeficient difuzního odrazu (hodnoty 0 až 1), $\boldsymbol{I_d}$ je intenzita dopadajícího světla – tedy barva světelného zdroje, $\boldsymbol{C_d}$ je barva tělesa. To znamená, že intenzita odrazu je největší pro pravý úhel dopadu vzhledem k rovině tělesa.

2.3.3 Zrcadlová složka

Zrcadlové složka je tvořena zrcadlovým odrazem. U dokonalého zrcadlového odrazu (*specular reflection*) se paprsek odrazí pod stejným úhlem jako dopadl, podle zákona odrazu. Zrcadlová složka je důležitá u kovů, kde je difuzní složka méně významná. Jak spočítat odražený paprsek je uvedeno v odstavci 3.2.4.

Tvar zrcadlové složky pro Phongův model je

$$E_s = k_s \cdot I_s (R \cdot V)^h$$

kde k_s je koeficient zrcadlové složky a $\boldsymbol{I_s}$ je intenzita světelného zdroje. *h* je exponent ovlivňující velikost odlesku, může nabývat hodnoty od 5 do 500.

2.4 Více světelných zdrojů

V předchozích vztazích jsme uvažovali pouze jediný zdroj světla. Zavedeme-li ve scéně více zdrojů (jejich počet označíme *n*), musíme zavést vektory $\boldsymbol{L_i}$ a $\boldsymbol{R_i}$ pro každé ze světel, stejně tak jako úhly α_i, β_i a intenzitu $\boldsymbol{I_i}$. Tyto indexujeme od 1 do *n*. Vektory $\boldsymbol{N}$ a $\boldsymbol{V}$ se při uvažování různých světel nemění.

K *n* světlům definovaných uživatelem přidáme navíc speciální ambientní zdroj světla.

Výsledná barva bodu při osvětlení více zdroji vznikne sečtením barev pro každý ze zdrojů:

$$E = \sum_{i=1}^{n} (E_d + E_s) + k_a \cdot C_d$$

E_d a E_s jsou zadané uživatelem, druhý term zastupuje ambientní světlo a je pro všechny modely stejný.

Tato rovnice však není součástí samotného osvětlovacího modelu, ten modeluje pro jednoduchost pouze jeden světelný zdroj. Tuto rovnici musí doplnit program.

2.5 Rozdělení osvětlovacích modelů

Modely se rozdělují podle vlastností materiálu, které dokáží zohlednit:

- *Anizotropie* povrchu. Anizotropie znamená, že povrch materiálu vykazuje v různých směrech různé světelné vlastnosti (např. některé kovy, tkanina). Jinak se povrch nazývá izotropní. Modely se takto rozdělují na anizotropní (např. Kajiya) a izotropní.
- Vícevrstvý povrch. Některé modely dokáží simulovat povrch složený ze dvou a více vrstev (např. nalakované dřevo).

Dále se rozdělují podle jejich principu:

- Empirické. Takové modely nereflektují fyzikální realitu šíření světla. Mohou být ale jednoduché na výpočet, jako např. Phongův model.
- Fyzikální. Popisující odraz na základě optického šíření světla (Strauss, Torrance-Sparrow), některé i na základě šíření elektromagnetických vln (He a kol.)

2.6 Konkrétní osvětlovací modely

Náš program umí zobrazit libovolný osvětlovací model, podle symbolického zadání vzorců, které může uživatel vytvářet. Zde uvedeme ty, které jsou součástí programu.

2.6.1 Phongův model

Phongův model, izotropní empirický model, nejjednodušší z osvětlovacích modelů, byl zveřejněn už roku 1975. I když nedává příliš realistické výsledky, pro svoji jednoduchost je implementován v hardwaru grafických karet.

$$E = E_{diffuse} + E_{specular}$$
$$E_{diffuse} = k_d \cdot I \cdot C \cdot \cos(\alpha)$$
$$E_{specular} = k_s \cdot I \cdot (R \,.\, V)^h$$

Parametry byly popsány výše. Jak se vidět z rovnice, barva zrcadlového odlesku nezávisí na barvě tělesa. Cos (α) může být ekvivalentně nahrazen skalárním součinem ***R***.***V***.

2.6.2 Straussův model

Paul Strauss publikoval v roce 1990 model, který patří již mezi fyzikálně založené modely. Vzhled materiálu je možno na rozdíl od Phongova modelu ovlivnit pomocí intuitivních parametrů s hodnotami od 0 do 1:

- hladkost (smoothness, značí se *s*), kde pro 0 je materiál matný, a pro 1 dokonale lesklý.
- kovovost (metalness, *m*), pro 1 je materiál kovový
- průhlednost (transparency, *t*), pro 1 je materiál zcela průhledný

Rovnice jsou následující:

$$E = I \cdot (E_d + E_s)$$
$$E_d = (1 - m \cdot s) \cdot r_d \cdot C \cdot \cos(\alpha) \qquad r_d = (1 - s^3) \cdot (1 - t) \qquad C - \text{barva materiálu}, I - \text{barva světla}$$
$$E_s = r_s \cdot C_s \qquad r_s = r_j \cdot \cos(\beta)^h \qquad h = \frac{3}{1 - s}$$
$$r_j = \min(1, r_n + j \cdot (r_n + k_j)) \qquad r_n = 1 - t - r_d \qquad k_j = 0.1$$

$$j = F(2\alpha/\pi) \cdot G(2\alpha/\pi) \cdot G(2\gamma/\pi)$$

funkce F, G aproximují Fresnelovy vztahy

$$F(x) = \frac{(x - k_f)^{-2} - k_f^{-2}}{(1 - k_f)^{-2} - k_f^{-2}}, \quad G(x) = \frac{(x - k_g)^{-2} - k_g^{-2}}{(1 - k_g)^{-2} - k_g^{-2}}, \quad k_f = 1.12, k_g = 1.01 \text{ (konstanty)}$$

$$C_s = I + m \cdot (1 - F(2\alpha/\pi)) \cdot (C - I) \qquad C - \text{barva materiálu}, I - \text{barva světla}$$

3 Ray-tracing

Osvětlovací model nám říká, jak se světlo odráží od povrchu objektů a určuje tak barvu každého bodu ve scéně. Pokud ovšem chceme tuto trojrozměrnou scénu zobrazit na dvojrozměrné obrazovce počítače, potřebujeme metodu, která scénu promítne a pro každý pixel obrazovky určí jeho barvu. Takových metod bylo vymyšleno mnoho, lišících se kvalitou, rychlostí a spektrem optických jevů, které dokáží zobrazit. Pro naše potřeby byl zvolen *ray-tracing*, metoda zpětného sledování paprsků s rekurzí.

V této kapitole je uveden obecný popis metody, v kapitole následující je uvedena modifikace ray-tracingu pro potřeby našeho programu.

Ve fyzice se uplatňují tři pohledy na šíření světla – vlnový, paprskový a kvantový. V počítačové grafice se téměř výhradně pracuje s geometrickým – paprskovým přístupem. Paprsky se šíří od světelného zdroje do všech směrů, na lesklých objektech se odráží, na průhledných objektech se lomí a zase z objektu vystupují, a určují tak osvětlení scény. Nakonec se dostanou do oka pozorovatele (nebo kamery), pokud ovšem neskončí mimo scénu.

Ray-tracing (i ray-casting) využívají právě tuto představu. Pokud bychom chtěli simulovat výše uvedený princip šíření světla, hodně práce bychom udělali zbytečně – pro ty paprsky, které opustí prostor scény a nikdy se do kamery nedostanou. Proto obě techniky sledují paprsky opačně, směrem od pozorovatele. Proto se někdy nazývají *zpětné sledování paprsku*.

3.1 Ray-casting

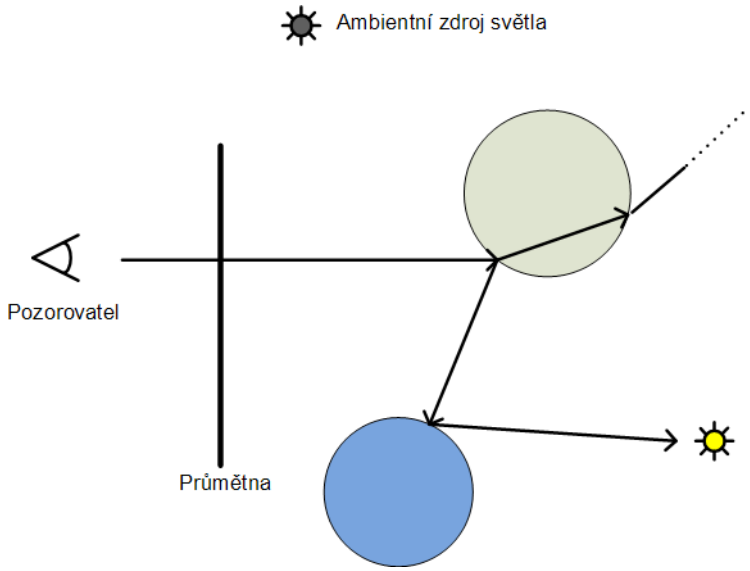
Ray-casting („vrhání paprsku“) je jednodušší varianta ray-tracingu, která vynechává odlesky a lomy. Principiálně je stejná. Zde popíšeme její princip.

Zavedeme do scény navíc kameru, která reprezentuje oko pozorovatele a *průmětnu*. Průmětna je virtuální rovina, do které se promítá výsledný obraz. V průmětně si zvolíme promítací obdélník, který představuje obrazovku. Ten si rozdělíme na pomyslné pixely. Nyní vyšleme paprsek z kamery, skrz každý pixel průmětny do scény. Na dráze paprsku najdeme nejbližší těleso, které paprsek protíná,

a jeho barvu zapíšeme do odpovídajícího pixelu. Při ray-castingu bereme v potaz pouze tento první průsečík, proto se někdy nazývá sledování paprsku prvního řádu.

3.2 Ray-tracing

Ray-tracing („sledování paprsku“) je vylepšení ray-castingu. Po nalezení prvního průsečíku s tělesem se nekončí, ale vytvoří se odražený paprsek, v případě, že těleso je lesklé. V případě, že těleso je (polo)průhledné, vytvoří se ve stejném bodě i lomený paprsek, podle Snellova zákona lomu. Tyto oba paprsky se opět vyšlou do scény a hledají se opět průsečíky s dalšími tělesy. Postup se rekurzívně opakuje. Ve výsledku na povrchu tělesa uvidíme mimo vlastní barvu povrchu i odrazy ostatních těles, u průhledných těles uvidíme také skrz ně. Tímto se ray tracing liší od ray-castingu.



3.2.1 Stíny

Chceme-li přidat podporu stínů, vyšleme z bodu, kam dopadl paprsek, navíc speciální tzv. *stínový paprsek* (*shadow ray*), směrem ke zdroji světla. Pokud na své cestě protne jiné těleso, znamená to, že původní bod leží ve stínu.

Pouze pokud bod je osvětlen zdrojem světla, je vyhodnocen osvětlovací model.

Výhodou je, že hledáme libovolný průsečík mezi bodem a světelným zdrojem, po nalezení prvního průsečíku tedy hledání zastavíme, což je rychlejší než nalezení všech průsečíků a vybrání minima.

Ambientní složku započítáme pomocí ambientního zdroje světla. Program se k němu chová odlišně než k normálnímu zdroji světla - stínový paprsek k němu dojde vždy, bez ohledu na zastínění. Je to proto, že ambientní složka je „všudypřítomná“.

3.2.2 Více světelných zdrojů

Přidání více světelných zdrojů do scény je přímočaré. Stínový paprsek vyšleme ke každému světelnému zdroji, a poté vyhodnotíme osvětlovací model pro všechny nezakryté zdroje a barvu sečteme.

3.2.3 Algoritmus ray-tracingu

Rekurzivní algoritmus zde zapíšeme v pseudokódu C.

```
Barva Sleduj(Paprsek p, int hloubka)
{
      Bod A = Najdi první průsečík (p);
      if( A == Nekonečno)
            return BarvaPozadí;
      else
      {
            Barva b = ambient;
            for(int i=0; i< počet světel; ++i)
            {
                  Paprsek stínovací = světlo_i – A;
                  Bod B = Najdi libovolný průsečík(stínovací);
                  if( B == Nekonečno )
                        b += k_l·Vyhodnoť osvětlovací model(A, světlo_i);
            }
            if( hloubka <= maximální hloubka )
            {
                  if( těleso bodu A je odrazivé )
                  {
                        Paprsek odraz = k_r·Spočti odražený paprsek(A, p);
                        b = b + Sleduj(odraz, hloubka + 1);
                  }
                  if( těleso bodu A je průhledné)
                  {
                        Paprsek lom  = k_t·Spočti lomený paprsek(A, p);
                        b = b + Sleduj(lom, hloubka + 1);
                  }
            }
            return b;
      }
}
```

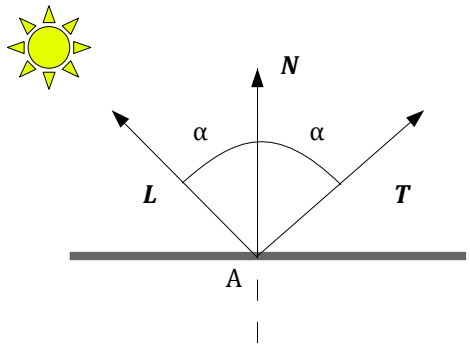
$Světlo_i$ znamená pozice i-tého světelného zdroje. k_r , k_t a jsou koeficienty váhy odrazu a lomu, k_l je intenzita světla. Všechny mají hodnoty od 0 do 1.

Protože v každém průsečíku paprsku s tělesem mohou vzniknou až dva nové paprsky, mohl by počet paprsků být při větší hloubce značný. Je proto důležité zvolit správně hloubku rekurze. Lze ji omezit tzv. dynamicky, podle podílu paprsku na výsledné barvě pixelu (tento podíl s hloubkou klesá), nebo staticky - maximální hloubku omezit konstantou. Zvolili jsme statické omezení hloubky, nastavitelné v programu. Doporučuje se 3 až 5, při větší hloubce už není nárůst kvality obrazu znatelný.

3.2.4 Výpočet odraženého paprsku

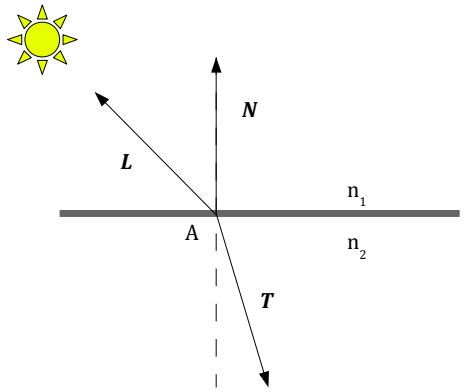
Označme místo dopadu paprsku jako A. Určíme vektor ke světlu ***L***, známe normálu povrchu ***N***. Oba vektory jsou znormalizované. Odražený paprsek ***T*** získáme rovnicí

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



Obrázek 4: Výpočet odraženého paprsku.

3.2.5 Výpočet lomeného paprsku



Paprsek se lomí na rozhraní dvou látek s indexy lomu n_1 a n_2. Podle Snellova zákona lomu, je-li úhel mezi L a N větší než mezní úhel, k lomu nedojde. Pokud je úhel menší, paprsek se zalomí.

$$T=[n_{12}(N\cdot L)-\sqrt{1-n_{12}^2(1-(N\cdot L)^2)}]\cdot N-n_{12}\cdot L$$

kde n_{12} je n_1/n_2.

3.2.6 Reprezentace scény

Základní možností je reprezentovat scénu pomocí geometrických primitiv – např. koule, kvádru, kužele. Chceme-li sestavit složitější objekty, můžeme použít *CSG – constructive solid geometry*. CSG model je sestaven z primitiv pomocí množinových operací – např. sjednocení, průnik, odečtení.

Lze přidat i implicitní a parametrické plochy, dokážeme-li spočítat jejich průsečík s paprskem. Pomocí těchto reprezentací získáme perfektně hladký obraz objektů.

Objekty lze však reprezentovat také pomocí skupiny trojúhelníků (*triangle mesh*), lze tak aproximovat libovolně složité plochy a objekty.

3.2.7 Vlastnosti ray-tracingu

Ray-tracing zvládá velmi dobře odrazy, i několikanásobné, a lomy, a poskytuje tak poměrně kvalitní výsledky.

Ve své základní podobně podporuje pouze bodové zdroje, což znamená, že všechny stíny mají ostrý okraj, jinými slovy, nepodporuje *měkké stíny*. Tuto základní variantu ray-tracingu jsme použili v této práci.

Problémem ray-tracingu je rychlost – počítání průsečíku paprsku se scénou je výpočetně velmi náročné. Zabírá více než 90% času běhu algoritmu u základní, neoptimalizované varianty. Tímto problémem, pro omezenou skupinu změn scény, jsme se zabývali v této práci.

3.2.8 Globální osvětlovací model

Lokální osvětlovací model je takový model, který vypočte odraz v místě dopadu paprsku pouze podle světelného zdroje, bez toho, aniž by uvažoval vliv dalších objektů.

Zavedením odrazů a lomů jsme zavedli globální osvětlovací model, protože na barvě bodu se podílí barvy objektů získaných pomocí odraženého paprsku a lomeného paprsku. Barvu bodu (tělesa, které je lesklé i poloprůhledné) v globálním osv. modelu získáme jako

$$E = E_l + k_r \cdot E_r + k_t \cdot E_t \qquad (1)$$

kde E_l je barva bodu získaná pouze pomocí lokálního osvětlovacího modelu, E_r je barva bodu, získaná vyhodnocením modelu v místě dopadu odraženého paprsku (*reflected*). E_t je barva bodu v místě dopadu lomeného paprsku (*transmitted*, nebo také *refracted*), je-li těleso průhledné. k_r je koeficient váhy odrazu, k_t je koeficient váhy lomu, oba s hodnotami od 0 do 1.

Rovnice (1) je ve své podstatě rekurzivní, protože pokud paprsek odrazu dopadl opět na lesklé těleso, je v tom místě opět vypočten globální osvětlovací model pomocí této rovnice. Rozepíšeme-li si rovnici, v případě plného lomu a odrazu do hloubky dvě, dostaneme

$$E = E_l^a + k_r \cdot \left(E_l^b + k_r \cdot E_r^b + k_t \cdot E_t^b\right) + k_t \cdot \left(E_l^c + k_r \cdot E_r^c + k_t \cdot E_t^c\right)$$

Horními indexy jsou značeny body, ve kterých zjišťujeme osvětlení.

Jak je vidět z rovnice, váha odrazů (a lomů) klesá s hloubkou tak, že váha odrazu v *i*-té hloubce je k_r^i. Dále je na tomto příkladu vidět, že složitost vyhodnocení osvětlení jediného bodu může být v globálním osv. modelu mnohem výpočetně náročnější než v lokálním modelu.

4 Řešení

Našim cílem je implementovat aplikaci pro interaktivní testování osvětlovacích modelů podporující jakýkoli osvětlovací model.

V této části popíšeme požadavky kladené na aplikaci a výsledné řešení těchto požadavků, které bylo implementováno v naší aplikaci.

Uvedeme zde hlavní principy řešení a vynecháme technické detaily, které uvedeme v kapitole následující.

4.1 Požadavky

Zásadním požadavkem je možnost měnit osvětlovací model libovolně, nikoli z pevně dané množiny modelů. To znamená zadat model symbolicky, pokud možno podobně formální definici modelu, jako je uvedeno v kapitole 2.

Pokud bychom vykreslili osvětlovací model staticky, scéna bude mít vzhled a barvy určeny pevně, bez možnosti je změnit. Pokud však chceme určitý osvětlovací model prozkoumat, zjistit jeho možnosti, je namístě mít možnost interaktivně měnit parametry materiálu a pozorovat, jak se vzhled materiálu mění. Dalším požadavkem je tedy interaktivní změna parametrů modelu s rychlým zobrazením výsledku.

Pro realistický vzhled požadujeme také odrazy a odlesky.

Požadavek pramenící z rovnic osvětlovacích modelů je, že v našich rovnicích potřebujeme podporovat vektorovou aritmetiku.

4.2 Řešení

Jako zobrazovací metodu jsme použili ray-tracing. Sám o sobě nám zajistí přechod od lokálního osvětlovacího modelu ke globálnímu. Je založen na geometrické optice a principy odrazu a lomu světla jsou v něm již obsaženy. To nám v programu umožní vyžadovat osvětlovací modely od uživatele pouze ve své lokální podobě. Principy odrazů a lomů jsou u osv. modelů neměnné, takže matematický popis modelu nebudeme zatěžovat popisem odrazů a lomů a tyto budou implicitně vypočítány.

Požadavky jsme vyřešili úpravou ray-tracingu, která staví na těchto principech:

- symbolický ray-tracing – paprsek vrací nikoli barvu pixelu, nýbrž vzorec modelu pro tento pixel
- oddělení ray-tracingu od vykreslení – vzorce vrácené ray-tracingem uložíme a při následné změně parametrů už je pouze vyhodnocujeme, ray-tracing tedy probíhá pouze jednou.

Úpravu nyní vysvětlíme podrobně.

4.2.1 Terminologie – parametry a proměnné

Proměnnými budeme nazývat hodnoty důležité pro výpočet osvětlovacího modelu v bodě dopadu paprsku. Je to například normálový vektor tělesa ***N*** v tomto bodě, vektor k pozorovateli ***V***, vektor ke světlu ***L*** (pro každé světlo jiný). Podrobný výpis je v odstavci 9.2.3.

Proměnná je obecně v každém bodě tělesa různá, a je určena tělesy ve scéně a jejich rozměry. Uživatel nemůže hodnotu ovlivnit přímo.

Parametr je naproti tomu pojmenovaná hodnota (skalár či vektor) ve vzorci, které chceme nechat uživatelem libovolně měnit. Je to typicky barva tělesa, světla, nebo různé koeficienty modelu, které mění vzhled materiálu.

Uvedeme zde pro ilustraci opět Phongův model.

$$E = E_{diffuse} + E_{specular}$$
$$E_{diffuse} = k_d \cdot I \cdot C \cdot \cos(\alpha)$$
$$E_{specular} = k_s \cdot I \cdot (R \,.\, V)^h$$

Zde k_d, k_s, ***I***, h jsou parametry – jejich hodnoty jsou volitelné a mění vzhled objektu. ***R***, ***V***, α jsou proměnné - pevně dané (ale v každém bodě tělesa různé) hodnoty.

4.2.2 Princip symbolického ray-tracingu

Ray-tracing je výpočetně velice náročný, a nejvíce času algoritmu (až 90%) zabírá výpočet průsečíků s tělesy. Pokud bychom však se scénou nějakou dobu nehýbali, stačí vypočíst průsečíky jen jednou na začátku. Při statické scéně nám ještě zbývá jedna skupina změn ve scéně – změna materiálu objektů. Při takové změně se mění jen barva, tělesa zůstávají na svých místech. Do doby, než se změní pozice objektů ve scéně, můžeme využívat informace o průsečících zjištěné předem, a obejít

tak časově náročnou část. Kreslení pak bude mnohem rychlejší. Tohoto principu využijeme, nicméně z předchozího vyplývá, že animace je nám zapovězena.

Ray-tracing upravíme tak, že když vystřelíme paprsek, nevrátí se barva, jak je obvyklé. Vrátí se kompletní vzorec osvětlení podle aktuálního osvětlovacího modelu, včetně odrazů a lomů. To nazýváme *symbolický ray-tracing*. Vzorce získáme pro každý pixel obrazu a uložíme do paměti pro pozdější vyhodnocení. Jeden vzorec v podstatě uchovává kompletní informaci o průchodu jednoho paprsku scénou.

Vyhodnocování vzorce na barvu je tak odloženo, neděje se při hned průchodu scénou paprskem, jako je tomu u normálního ray-tracingu.

Parametry jsou jediné měnitelné elementy vzorce. Když uživatel změní hodnotu nějakého parametru, stačí vzorec vyhodnotit, s aktuálními hodnotami parametrů, a dostaneme barvu pixelu. Provedeme-li to pro každý pixel obrazu, máme výsledný obraz. Ray-tracing je tak oddělen od vykreslování.

Vzorec je tak parametrizován aktuálními hodnotami parametrů. Zadáme-li konkrétně všechny hodnoty parametrů, je tím jednoznačně dán obraz scény.

Pokud se změní pozice tělesa ve scéně, nebo se změní rozměr nějakého tělesa, je samozřejmě nutné znovu provést ray-tracing – hledání průsečíků, jinak by obraz nereflektoval provedené změny.

4.2.3 Vzorec osvětlení

Zde popíšeme, jak ze zápisu vzorce osvětlovacího modelu získáme vzorec osvětlení v daném bodě, zahrnujíce více světel, odrazy a lomy. Předvedeme opět na Phongově modelu.

$$E_{local} = k_d \cdot I \cdot C \cdot \cos(\alpha) + k_s \cdot I \cdot (R.V)^h$$

Zde vidíme vzorec osvětlení, tak, jak je zadán uživatelem. Nazveme ho *základní vzorec*. Jak je vidět, nereflektuje více světel, ani odrazy a lomy. Zadání vzorce tak sice zůstane jednoduché a intuitivní, ale vše ostatní musí do vzorce přidat program.

Instanciace proměnných

Vzorec zadaný uživatelem obsahuje výskyty proměnných, které však jsou pouze symbolickými označeními. Konkrétní hodnoty jsou doplněny při ray-tracingu, podle místa, kam paprsek dopadl. Dále budeme konkrétní hodnoty značit horními indexy podle místa, kam paprsek dopadl. Např. V^A je vektor k pozorovateli v průsečíku A.

Výsledný vzorec

Takto vypadá výsledná rovnice globálního osvětlovacího modelu s libovolným počtem světel. Využíváme zde teorie z odstavců 2.4 a 3.2.8.

$$E^A = k_l \cdot \left(\sum_1^n E^A_{local} + k_a \cdot C_d \cdot I_a \right) + k_r \cdot E^B + k_t \cdot E^C \tag{2}$$

pro každé světlo z *n* světel vyhodnotíme lokální osvětlovací model a sečteme je, a přičteme ambientní složku (generovanou programem automaticky). Dále přičteme vyhodnocení na objektech, kam dopadl paprsek lomu a odrazu. Pro ilustraci rozepíšeme term uvnitř sumy:

$$E^A = \left[k_l \cdot \sum_{i=1}^n \left[k_d \cdot I_i^A \cdot C^A \cdot \cos(\alpha^A) + k_s \cdot I_i^A \cdot (R_i^A . V^A)^h \right] + k_a \cdot C_d \cdot I_a \right] + k_r \cdot E^B + k_t \cdot E^C$$

Dolním indexem *i* je značen index světla, horním indexem bod, ve kterém se vyhodnocuje. C^A znamená barvu tělesa, na kterém je bod A.

4.2.4 Parametry

Ve dvou předcházejících kapitolách jsme se dopustili zjednodušení, které nyní doplníme. Jelikož parametry určují kvalitu materiálu, a každé těleso má obecně svůj vlastní materiál, musí i parametry existovat nezávisle pro každé těleso zvlášť. Takové dále (a také v programu) nazýváme *tělesové parametry* (*body specific parameters*). Jsou však i takové parametry, které jsou společné pro všechna tělesa. Je to třeba barva světla, nebo koeficient difuzního osvětlení. Takové dále nazýváme parametry globální (*global*), popř. parametry scény. Dále je zřejmé, že budeme potřebovat i *světelné parametry* (*light specific parameters*), abychom mohli mít pro každé světlo vlastní barvu.

Ve vzorcích výše by tedy parametry měly být značeny podle těles, ke kterým náleží (jsou-li tělesové). Parametry k_s, k_d zůstávají globální, k_r^A, k_t^A tělesové (je vlastností každého tělesa, jak moc odráží a jak moc láme světlo), k_l je světelný parametr značící intenzitu světla.

Další dělení parametrů spočívá v jejich datovém typu. Parametr může být buď skalární hodnota, nebo vektorová – složená ze tří čísel, což je třeba barva. Podle toho dělíme parametry na skalární a vektorové.

Nastiňme implementaci parametrů. V programu jsou parametry ovládány posuvníky. Pro každé těleso (a zvlášť pro scénu) je naalokováno pole parametrů, do kterých se zapisuje při pohybu posuvníku. Ve vzorcích osvětlení jsou uloženy ukazatele na hodnoty parametrů, a při vyhodnocení vzorce jsou tyto hodnoty čteny.

4.2.5 Přednastavené parametry

Uživatel zadává vzorec ve tvaru E_{local}, což je vzorec pouze lokálního modelu osvětlení. Jak bylo popsáno výše, program z něho vytvoří vzorec globálního modelu (rovnice 2). Přitom jsou do vzorce přidány parametry, které musíme „exportovat“ uživateli, aby je mohl ovládat. Jsou to následující tělesové parametry

- k_t – koeficient lomu (skalár, hodnoty 0 - 1)
- k_r – koeficient odrazu (skalár, hodnoty 0 - 1)
- C_d – barva tělesa (vektor, hodnoty 0 - 1)

a tyto světelné parametry

- k_l – koeficient intenzity světla(skalár, hodnoty 0 - 1)

Tyto parametry budou vytvořeny vždy, bez ohledu na vzorec, jaký bude zadán uživatelem.

4.2.6 Vektorová aritmetika

Vzorec osvětlovacího modelu vrací jako výsledek barvu, vektor tří složek RGB. Je to tedy vektorová rovnice.

$$E_{local} = k_d \cdot I \cdot C \cdot \cos(\alpha) + k_s \cdot I \cdot (R.V)^h$$

V uvedené rovnici Phongova modelu se vyskytuje násobení více druhů, podle typů operandů. Například ks·I je násobení skaláru s vektorem, I·C je násobení vektoru s vektorem. V zájmu toho aby uživatel zadával vzorec co nejvíce podobný formálnímu zápisu modelu musíme implementovat mechanismus přetěžování operátorů, který rozhodne o správné variantě operátoru. Přetížené operátory se musí lišit v počtu operandů nebo jejich typu (skalár/vektor). Implementace je uvedena v odstavci 5.3.2.

4.2.7 Invarianty

Některé koeficienty některých osvětlovacích modelů nemohou být zadávány zcela libovolně, musí mezi nimi být zachován jistý vztah.

Ve Phongově modelu např. musí platit

$$k_s + k_d + k_t = 1$$

protože množství světla, které povrch tělesa opouští nemůže být větší než množství, které na povrch dopadá. Proto jsme implementovali jednoduché invarianty, které zajišťují, že určené parametry mají součet roven jedné.

5 Implementace

V této části uvedeme principy implementace, architekturu projektu, a důležité datové struktury. Snažili jsme se vybrat zejména zajímavé aspekty implementace.

5.1 Požadavky implementace

Zde jsou uvedeny další požadavky na implementaci vedle požadavků na funkčnost programu uvedených v 4.1.

Protože máme vzorec pro každý pixel obrazu, musíme dbát na paměťové nároky programu. Bylo třeba navrhnout úspornou reprezentaci vzorce, která byla nazvána *bajtkód*.

Ray-tracing je úloha, kterou je možno dobře rozdělit na podúkoly. Dalším požadavkem je podpora *multithreadingu*, tedy zajištění spolupráce více vláken programu pro běh na vícejádrových procesorech/víceprocesorových systémech.

5.2 Platforma, prostředky

Jako programovací jazyk jsme zvolili C++. Je to z důvodu nároků na rychlost programu při vyhodnocování bajtkódu.

Při volbě knihoven pro tvorbu GUI aplikací pro C++ jsme se rozhodovali mezi Qt a wxWidgets, které jsou obě multiplatformní. Rozhodli jsme se pro wxWidgets, protože neomezuje licenční politiku výsledného programu, přičemž jsou bezplatným produktem. Použili jsme nejnovější verzi 2.8. Aplikace dále dosti využívá standardních knihoven C++.

Program byl primárně vyvíjen pro Windows, nicméně díky wxWidgets by bylo možno jej snadno portovat pro ostatní rozšířené OS (Mac OS, Linux).

Vykreslování výsledného 2D obrazu je kvůli rychlosti řešeno pomocí OpenGl a je popsáno v odstavci 5.3.6.

5.3 Principy implementace

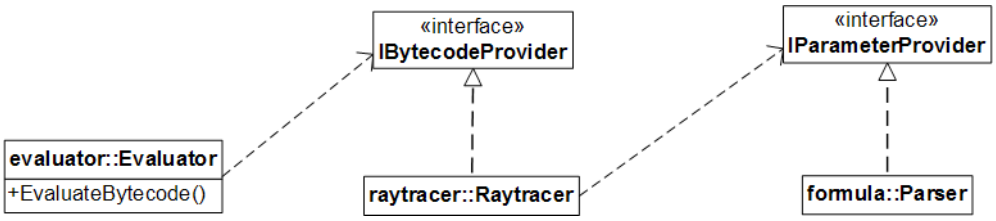
Tato podkapitola se zabývá základními principy implementace symbolického ray-tracingu, vysvětluje cestu vzorce od textového zadání až k výslednému obrazu.

Poznámka. Objekty frameworku wxWidgets mají předponu *wx*, např. wxString. Pod pojmem vector uvažujeme STL kontejner *std::vector*.

5.3.1 Přehled jmenných prostorů

V kódu dosti využíváme jmenných prostorů, abychom seskupili logicky související třídy. Jmenné prostory pojmenováváme malými počátečními písmeny, zatímco třídy velkými.

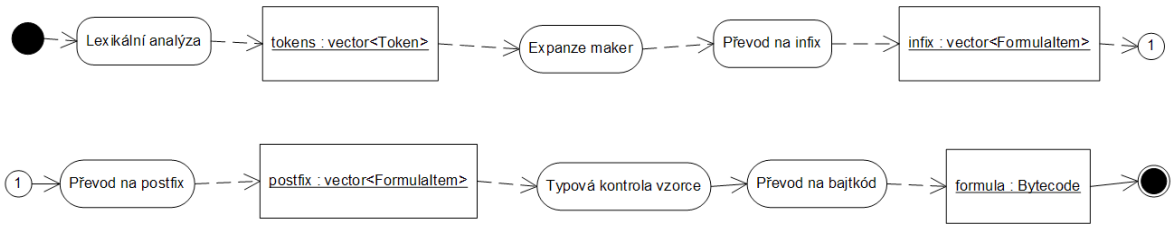
- config. Obsahuje třídu Config pro ukládání/načítání nastavení aplikace.
- evaluator. Třída Evaluator pro vyhodnocení bajtkódu a MasterThread a WorkerThread pro vyhodnocení ve více vláknech.
- exceptions. Vlastní definice výjimek, které aplikace používá.
- formula. Zásadní jmenný prostor pro zpracování vzorce a definice scény. Obsahuje třídu Parser a SceneParser pro parsování vstupu, třídu Bytecode a definice instrukcí bajtkódu v souboru Bytecode.h.
- geometry. Třídy pro výpočty ve 3D, včetně přetížených operátorů.
- gui. Třídy reprezentující okna aplikace, které dědí od wxFrame – obecného okna wxWidgets.
- interfaces. Rozhraní pro snížení závislostí mezi třídami.



- raytracer. Třída Raytracer pro ray-tracing a MasterThread a WorkerThread pro vícevláknový ray-tracing.
- scene. Definice těles, kamery, světla.
- Třída App. Není ve jmenném prostoru, reprezentuje aplikaci.

5.3.2 Zpracování vzorce

Zpracování vzorce odpovídá transformaci ze vstupního textu do bajtkódu. Provádí ho třída formula::Parser.



Obrázek 7: Diagram aktivit transformace vzorce. Obdélník značí objekt, zakulacený obdélník úkol.

Následuje popis operací nad vzorcem, v pořadí v jakém jsou prováděny, podle obr. 7. Uvedené metody a datové členy jsou patří do třídy *formula::Parser.*

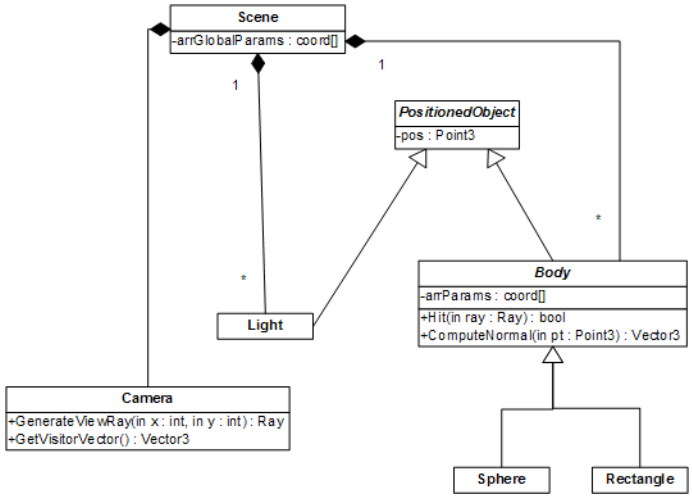
- Lexikální analýza. Lexikální analýzu provádní funkce *Scanner*. Rozpozná identifikátory, symboly(např. operátory) a čísla.
- Expanze maker. Jak je řečeno v příloze C, makra jsou čistě textová, bezparametrická. Jejich expanze spočívá pouze v nahrazení identifikátoru makra v hlavním vzorci. Podporujeme nicméně zanořování maker. Expanzi maker provádí funkce *ExpandMacros*.
- Převod na infix. Provádí se syntaktická a sémantická analýza parametrů (*ParseParameters*) a syntaktická analýza vzorce (*Tokens2InfixFormula*).
  - Sémantická analýza parametrů spočívá v naplnění kontejnerů *globalParameters* a *bodyParameters* typu *std::map<Parameter>*.
  - Syntaktickou analýzou vzorce se určí, jakého typu je identifikátor – parametr, proměnná, funkce, operátor. Pro určení se hledá v tabulkách funkcí, proměnných a parametrů. Určení správné funkce je ztíženo přetěžováním funkcí (a operátorů). Přetížené funkce se musí lišit počtem nebo typy argumentů. V této fázi určena pouze signatura funkce (např. * - násobení), a určení přesné varianty (např. $*_{(skalár, vektor)}$) proběhne při typovém vyhodnocení vzorce.
- Převod na postfix. Ten zajišťuje funkce *Infix2PostfixFormula*. Je použit Dijsktrův algoritmus „shunting yard“, který používá zásobník na operátory. Tento algoritmus pracuje s prioritami operátorů. Protože zde však ještě nemáme určené přesné varianty přetížených operátorů, musíme zajistit, aby všechny varianty měly vždy stejnou prioritu. Tento předpoklad je však přirozeně splněn (např. násobení).

- Typová kontrola vzorce. Je zajištěna metodou *TypeEvaluate*. Ta simuluje vyhodnocení vzorce v postfixu na zásobníku, ale vyhodnocuje pouze typy (skalár/vektor), nikoli čísla. Tak odhalíme veškeré chyby vzorce, a při vyhodnocování už nic nekontrolujeme. Především zde však rozlišíme varianty přetížených funkcí – zjistíme zde typy argumentů.
- Převod na bajtkód. Popis bajtkódu je rozsáhlejší, je proto uveden v podkapitole 5.4. Převod je přímočarý, protože vstupní i cílová reprezentace je v postfixu. Zajišťuje ho funkce *Postfix2Bytecode*.

Po rozparsování vzorce je v každém tělese naalokováno pole arrParams (viz obr.8), kde jsou hodnoty všech tělesových parametrů. Do těchto hodnot je zapisováno při pohybu posuvníku a ukazatele v bajtkódu směřují do tohoto pole. Ve scéně je naalokováno pole arrGlobalParams s globálními parametry.

5.3.3 Scéna

Při parsování definice scény třídou SceneParser je vytvořena instance třídy Scene a v ní jsou vytvořeny objekty podle definice scény. Situace je vidět na obr. 8. Scéna obsahuje instance kamery, světel a těles.



5.3.4 Implementace ray-tracingu

Použili jsme nejjednodušší variantu ray-tracingu bez optimalizace hledání průsečíků, podporujeme pouze bodová světla.

Úkolem naší implementace ray-tracingu je vytvořit na základě obecného vzorce lokálního modelu zadaného uživatelem – tzv. *základní vzorec* – vytvořit pro každý

pixel obrazu konkrétní vzorec který odpovídá dráze paprsku při putování po scéně. Výsledný vzorec je tvaru vzorce (2) v odstavci 4.2.3, tj. globální osvětlovací model. Výsledné instance vzorce se liší *pouze* hodnotami proměnných, podmíněných průsečíky toho kterého paprsku, a ukazateli na parametry, které závisí na tělese, kam paprsky dopadly.

Základní vzorec máme při ray-tracingu k dispozici v bajtkódu.

Implementovaný ray-tracing principiálně odpovídá algoritmu ray-tracingu uvedenému v odstavci 3.2.3.

Ray-tracing je implementován v rekurzivní metodě *Raytracer::RaytraceRay*. Metoda sleduje primární paprsek, který směřuje z kamery přes průmětnu do scény. Tento paprsek vygeneruje raytraceru kamera scény metodou *GenerateViewRay*. *Multisampling* (posílání více paprsků skrz jeden pixel) jsme neimplementovali.

Paprsek je sledován pro průsečíky na tělesech scény. Zda paprsek protíná těleso (abstraktní třída Body), zjistíme voláním abstraktní metody *Hit* na tomto tělese. Když najdeme těleso, které paprsek protíná, a nejbližší takové, v místě průsečíku vypočteme hodnoty proměnných (mezi jinými normálový vektor, který získáme voláním *Body::ComputeNormal* na tělese, jenž jsme protnuli). To zajišťuje metoda ComputeVariables.

Následující postup převádí bajtkód I.typu na bajtkód II.typu, pro definici viz odstavec 5.4.2.

Do bajtkódu okopírujeme základní vzorec, najdeme instrukce VAR_* (např. VAR_N_V, VAR_R_V) a nahradíme je aktuálními hodnotami proměnných v tomto bodě (instrukce SCALAR_VAL či VECTOR_VAL). Dále najdeme instrukce pro parametry (PARAM_*, např PARAM_G_S) a nahradíme je instrukcemi SCALAR_PTR, popř. VECTOR_PTR spolu s ukazateli na hodnoty příslušných parametrů (*Body::arrParams* či *arrGlobalParams* ve třídě Scene). Tak je bajtkód napojen na hodnoty parametrů, které pouze čte. Hodnoty jsou měněny jako důsledek tažení posuvníku pro změnu parametru. Tento postup samozřejmě předpokládá, že paměť, do které ukazatele ukazují, zůstane alokovaná po celou dobu, kdy může k vyhodnocení dojít. Toto zajišťuje metoda *WriteVariables*.

Tyto parametry jsou jediné hodnoty, které jsou získávány dynamicky při vyhodnocování bajtkódu. Ostatní instrukce jsou buď pevné hodnoty (proměnné) nebo funkce.

Právě popsané nahrazování v bajtkódu jsme nemohli provést dříve, neboť jsme nevěděli, v jakém bodě tělesa, a vůbec v jakém tělese se nacházíme, proměnné jsme tedy nemohli vyhodnotit. Podobně parametry – k jejich napojení musíme znát těleso, do jehož pole *arrParams* ukážeme ukazatelem.

Těmito úpravami se bajtkód stane nezávislý na osvětlovacím modelu, stane se z něj zcela obecná rovnice, která vrací vektor tří čísel.

Tímto byla popsáno vytvoření vzorce lokálního modelu, do globálního modelu ještě zbývají odrazy a lomy a více světel.

Více světel implementujeme přímočaře podle vzorce (2). Základní vzorec nakopírujeme *n*×, kde *n* je počet světel, a v každé kopii změníme hodnotu L – vektoru ke světlu. Poté přidáme *n-1* instrukcí vektorového sčítání. Tak vytvoříme validní bajtkód pro všechny světla.

Odraz a lom je implementován pomocí rekurze – zavoláme funkci RaytraceRay pro získání odrazu a lomu pomocí paprsků odrazu a lomu, které jsme vypočítali ze stávajícího paprsku. Funkce vrací bajtkód, a my ho přilepíme za již stávající hotový bajtkód a přenásobíme koeficienty k_r, k_l.

Využíváme zde vlastnosti postfixu, sice že nepotřebuje závorky. Stávající vzorec můžeme považovat za číslo, které vznikne jeho vyhodnocením na zásobníku. Při přidávání na konec stávajícího kódu vložíme jak operátory, tak operandy.

5.3.5 Vyhodnocení bajtkódu

Vyhodnocení vykonává rutina *Evaluator::EvaluateBytecode*, která vzorec vyhodnocuje na zásobníku. Jsou vyhodnoceny všechny vzorce na základě aktuálních hodnot parametrů, tak jak byly nastaveny uživatelem.

Rutina vyhodnotí vzorec na jeden průchod zleva doprava. Přejde-li na instrukci uložení hodnoty (např. SCALAR_VAL), zkopíruje hodnotu na vrchol zásobníku. Přejde-li na instrukci uložení hodnoty ukazatele (např. SCALAR_PTR), ukazatel dereferencuje a uloží na vrchol zásobníku. Přejde-li rutina na instrukci funkce, vezme ze zásobníku tolik čísel, jakou má funkce aritu, vyhodnotí funkci, a výsledek umístí na zásobník. Na konci vzorce zůstane na zásobníku vektor tří čísel, který je vrácen.

Takto získáme pro každý bod obrazu barvu, která mu přísluší.

5.3.6 Vykreslování obrazu

Pokud chceme dosáhnout realtimové rychlosti aktualizace obrazu, musíme zvolit rychlou metodu vykreslování. Kreslení pomocí GDI ve Windows je pro malou rychlost nepoužitelné. Po otestování dostupných metod jsme použili následující metodu vykreslování pomocí OpenGl.

Funkce *EvaluateBytecode* vyhodnocuje vzorec, a po vyhodnocení výslednou barvu zapíše do pole (*Evaluator::RenderContext::colorArray*), které je naalokováno tak, aby se dalo přímo použít jako textura OpenGl.

V režimu OpenGl nastavíme pouze ortogonální projekci, protože chceme vykreslit pouze 2D obraz. Vykreslíme jediný čtverec přes celou obrazovku – v režimu GL_QUADS. Jako texturu tohoto čtverce aplikujeme vyrenderovaný obraz. Protože rozměry obrazu jsou libovolné – nastavitelné uživatelem, musíme použít funkci *glTexSubImage2D*, která dokáže vykreslit texturu, která nemá rozměry mocnin dvou.

5.4 Bajtkód

Bajtkód je reprezentace konkrétního vzorce osvětlovacího modelu. Byl navržen tak, aby byl maximálně paměťově úsporný, a jeho vyhodnocování bylo přitom rychlé.

Je to aritmetický výraz v postfixu, kde operandy i operátory jsou definovány instrukcemi. Je vyhodnocen zásobníkovým automatem (viz níže).

Rutina, která bajtkód vyhodnocuje, je zcela nezávislá na sémantice vzorce, pouze vzorec vyhodnotí a dostane vektor tří složek, který je chápán jako barva bodu.

Programově je reprezentován jednoduše jako pole typu BYTE (unsigned char), toto pole je pro snadnou manipulaci obaleno třídou Bytecode, která obsahuje metody pro přidávání instrukcí.

5.4.1 Instrukce

Instrukce jsou jsou čísla, ve zdrojovém kódu jsou reprezentovány jako konstanty. Jejich definice je v souboru Bytecode.h. Pro všechny funkce a operandy nám postačí méně než 70 instrukcí, jedna instrukce zabírá tedy pouze jeden bajt. Za ní následují parametry instrukce, jsou-li nějaké.

Instrukce jsme rozdělili do pěti základních typů:

- Uložení hodnoty

    - skaláru SCALAR_VAR, následuje `sizeof(float)` bajtů na uložení hodnoty typu float
    - vektoru VECTOR_VAL, následuje `3·sizeof(float)` bajtů na uložení tří složek vektoru jako float
- Uložení ukazatele
    - na skalár SCALAR_PTR, následuje `sizeof(float*)` bajtů na uložení ukazatele na float
    - na vektor VECTOR_PTR, následuje `sizeof(float*)` bajtů na uložení ukazatele na první prvek pole
- Funkce a operátory (v postfixu mezi funkcí a operátorem není rozdíl). Tyto instrukce nemají parametry, nýbrž operandy. Každá funkce a operátor uvedený v 9.2.2 má svoji instrukci, s toutéž aritou. Operandy jsou uvedeny v postfixu před instrukcí. Jméno konstanty je tvaru Jméno_*TypOperandu*$_1$*TypOperandu*$_2$*...TypOperandu*$_n$*_TypVýsledku.*
- Proměnné. Jsou to instrukce tvaru VAR_*JménoProměnné_Typ*, které zastupují všechny proměnné uvedené v odstavci 9.2.3. Tyto instrukce jsou pouze symbolické výskyty proměnných, nikoli jejich hodnoty. Vyskytují se pouze v bajtkódu I.typu (viz níže).
- Parametry. Instrukce PARAM_G_*Typ* pro globální, PARAM_B_*Typ* pro tělesové. Za instrukcí následuje pořadové číslo parametru, nikoli aktuální hodnota parametru. Tyto instrukce se vyskytují pouze v bajtkódu I.typu (viz níže).

5.4.2 Dva typy bajtkódu

V průběhu zpracování vzorce a ray-tracingu se pracuje se různými formami bajtkódu, které jsme pojmenovali typ I a typ II. Liší se v tom, jak nakládají s parametry a proměnnými, jinak jsou stejné.

Bajtkód I. typu vznikne při zpracování vzorce. Obsahuje pouze symbolické instrukce pro proměnné (VAR_...) a parametry (PARAM_G_S, ...). Vzorec má v této fázi pouze jedinou instanci, a konkrétní hodnoty proměnných nejsou definovány. Podobně je tomu s parametry.

Při ray-tracingu je bajtkód I. typu převáděn na typ II. Převod je uveden v odstavci 5.3.4. Typ II vzniká v mnoha instancích a každá z nich se liší konkrétními

hodnotami proměnných. Symbolické instrukce pro proměnné jsou nahrazeny konkrétními čísly (SCALAR_VAR) zjištěnými v konkrétním bodě tělesa. Symbolické instrukce parametrů jsou nahrazeny ukazateli (SCALAR_PTR) do pole hodnot konkrétního tělesa.

5.4.3 Příklad

Předvedeme příklad bajtkódu II.typu na difuzní složce Phongova modelu.

Vzorec v infixu je

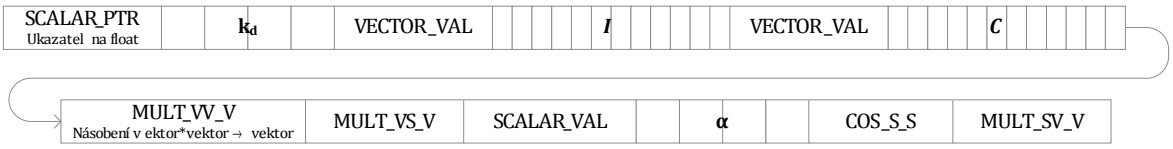
$$k_d \cdot I \cdot C \cdot \cos(\alpha)$$

V postfixu je např. ve tvaru

$$kd \quad I \quad C \quad \cdot_{(v,v)} \quad \cdot_{(v,s)} \quad \alpha\alpha \quad \cos \quad \cdot_{(s,v)}$$

kde dolními indexy jsou značeny typy operandů.

Výsledný bajtkód vypadá následovně, za předpokladu, že `sizeof(float)` = 4.



5.5 Multithreading

V předchozím výkladu jsme pro přehlednost neuváděli aspekty multithreadingu. Nyní vysvětlíme principy využití paralelismu v naší aplikaci.

wxWidgets obsahují podporu pro multithreading, pomocí tohoto frameworku je tedy možné psát vícevláknové aplikace multiplatformně.

Paralelně jsme implementovali ray-tracing a vyhodnocování. Parsování vzorce je velmi rychlé, takže není třeba jej rozdělovat.

Ray-tracing obecně je úloha velmi vhodná pro rozdělení na více vláken, protože sledování paprsků ve scéně je na sobě nezávislé. Úloha se tak přirozeně rozdělí na podúkoly.

V naší aplikaci využíváme paralelního modelu Master-Worker. Jedno vlákno, označené jako master, rozděluje úkoly pracovníkům, workerům. Když worker skončí s aktuálním úkolem, master mu přidělí další část.

Ray-tracing i vyhodnocování mají stejnou elementární jednotku práce – zpracování jednoho pixelu. Jsou implementovány podobně, proto popíšeme společné aspekty implementace.

Používáme větší jednotku, abychom zabránili velké režii přepínání vláken. Jednotkou je skupina po sobě jdoucích horizontálních linek obrazu, reprezentovaná strukturou *Task*.

Nyní popíšeme podrobně princip modelu Master-worker.

Úkoly se hromadí ve frontě TaskQueue<Task>, která obsahuje rutiny pro výběr a vložení úkolu. Tato třída je synchronizována pomocí kritické sekce.

Pro synchronizaci úkolů využíváme semafor (třída wxSemaphore). Metoda Wait se pokusí semafor dekrementovat (a blokuje pokud je roven nule), metoda Post semafor inkrementuje.

Pro synchronizaci masteru a workerů máme dva semafory – semAvail („available“) a semFinished.

Master při startu vytvoří *m* workerů – vláken, které budou zpracovávat podúkoly, a spustí je. Počet úloh označíme *n*.

Při každém vložení úlohy do fronty master jednou inkrementuje semAvail, tím signalizuje, že úloha byla vytvořena. Celkem tedy zavolá *n*× `semAvail.Post()`. Poté zavolá také *n*× `semFinished.Wait()`. Tím počká na dokončení *n* úloh.

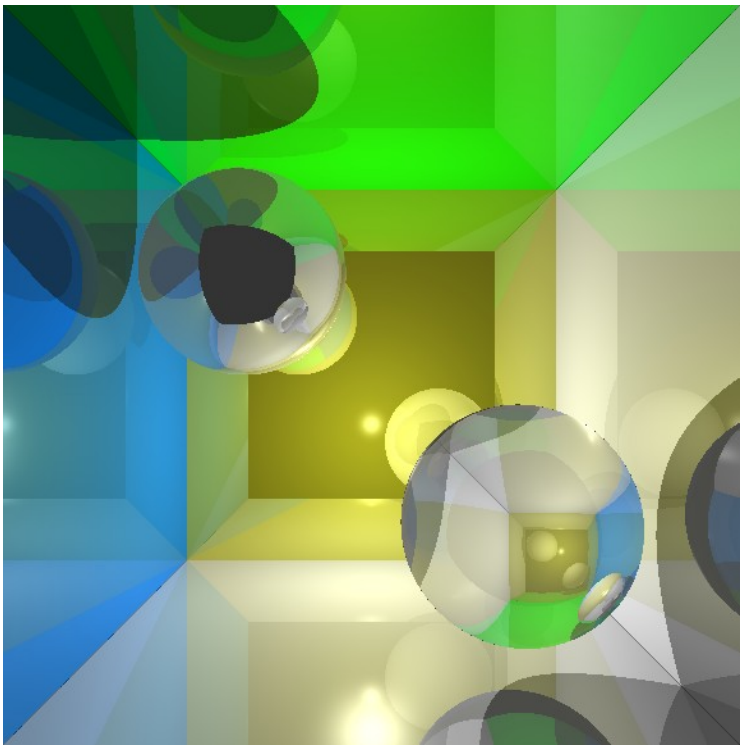
Worker při svém spuštění dostane instance semAvail a semFinished od mastera. Než se ve frontě objeví úkol, čeká na volání `semAvail.Wait()`. Když se úkol objeví, semafor se hned dekrementuje a worker je odblokován. Úkol vyzvedne z fronty, zpracuje ho, a zavolá `semFinished.Post()`, čímž signalizuje, že jeden úkol je hotov.

Protože master čeká na příkazu *n*× `semFinished.Wait()`, bude odblokován, až všech *n* úloh bude hotových.

Aby byl maximálně využit výkon procesoru(ů), vytvoří program právě tolik workerů – pracovních vláken, kolik procesorů/jader je v systému (přesněji kolik hlásí operační systém). Při menším počtu vláken by některá jádra procesoru neměla nic na práci.

6 Závěr

Implementovali jsme systém, který dokáže aplikovat libovolný osvětlovací model na danou scénu a parametry modelu nechat libovolně měnit uživatelem.



6.1 Shrnutí

Druhá a třetí kapitola byly teoretické, s obecnými tématy týkající se naší práce. Ve druhé kapitole jsme definovali osvětlovací model a uvedli dva známé modely. Ve třetí kapitole jsme prozkoumali ray-tracing v obecné rovině a také jeho vztah k osvětlovacím modelům. Čtvrtá kapitola představila požadavky na aplikaci a základní principy řešení těchto požadavků, především princip *symbolického ray-tracingu*, prozatím však v teoretické rovině, bez většího zřetele na implementaci. V páté kapitole jsme uvedli syntaxi vstupního jazyka vzorce a scény a v další kapitole byl představen implementovaný systém pomocí uživatelské dokumentace. Sedmá kapitola uvedla přehled architektury a principy implementace systému, především strukturu *bajtkód*.

6.2 Splnění cílů

Podařilo se nám vytvořit fungující implementaci, která podporuje symbolické zadání vzorce.

Snažili jsme se především o univerzálnost programu, proto počet vytvořených modelů je dosti malý, stejně jako počet těles, která lze ve scéně vytvořit. Nebylo by však těžké toto napravit.

6.2.1 Rychlost programu

Rychlost ray-tracingu i vyhodnocování velmi záleží na velikosti obrazu, složitosti scény a především na vzorci – vzorce se dost liší velikostí. Když je obraz malý a vzorec jednoduchý (Phong), rychlost vyhodnocení skutečně probíhá v reálném čase. U složitější scény a složitějšího vzorce však rychlost dosti klesá. Pro porovnání, ve scéně bez odrazů je velikost Phongova vzorce pro jeden pixel přibližně 120 bajtů a scéna se vykresluje velmi rychle. U složitější scény, kde byl použit Straussův model, je velikost vzorce průměrně 1,5kB, místy i 3kB, a to pro jeden pixel! Nevýhoda je v tom, že každý odraz a lom v podstatě zdvojnásobuje velikost vzorce. Velikost by se dala omezit označením některých objektů za neprůhledné popř. neodrazivé. Nyní to není možné, protože nemůžeme dopředu vědět, zda na tomto objektu bude požadován odraz/lom. Proto musíme vytvořit všechny potenciálně možné odrazy/lomy.

6.3 Diskuze

Práce na projektu byla velmi zajímavá a poučná, naučil jsem se mnoho nových zajímavých programovacích technik, zejména principy paralelního programování. Dále jsem se poučil v tématech počítačové grafiky, zejména osvětlovacích modelů a ray-tracingu.

Použitý framework wxWidgets mě poněkud zklamal, při programování v něm jsem narazil na několik záludných „vlastností“. Vytváření GUI programu je dosti těžkopádné, chybí nějaký opravdu pohodlný editor. Dokumentace toho frameworku by zasloužila rozšířit. Na druhou stranu, framework svůj účel splnil docela dobře, a také bych měl vyzdvihnout jeho rychlost, kvůli tomu, že vstupní (na OS nezávislý) kód

je překládán do nativního API cílového operační systému pomocí maker preprocesoru; nepoužívá se žádná mezivrstva.

6.4 Další vývoj programu

Projektu bych se rád dále věnoval, možných zlepšení je velmi mnoho.

Bylo by dobré program zrychlit, vyhodnocování především pomocí optimalizace bajtkódu (zbytečné výrazy typu `<0,0,0> · cokoli`). Bylo by také namístě přepsat parser tak, aby pracoval metodou rekurzivního sestupu, současný je napsán nestandardně a poněkud těžkopádně. Pro teoretickou exaktnost by bylo vhodné sjednotit pojmenování a sémantiku parametrů, zvláště těch přednastavených.

Rád bych vytvořil více osvětlovacích modelů, a více těles, především triangle mesh s načítáním z nějakého známého formátu. Pro realistické odlesky by bylo zajímavé přidat (HDR) environment mapping. A velkou výzvou by bylo program převést na GPU, kde by se dalo velmi dobře využít paralelismu.

Literatura

[1] Jiří Žára a kol.: Moderní počítačová grafika, Computer Press, Brno, 2004

[2] RNDr. Josef Pelikán: materiály k přednášce Počítačová grafika II, http://cgg.ms.mff.cuni.cz/~pepca/lectures/pgr004.html

[3] Martin Krahulík: Reprezentace BRDF v počítačové grafice, diplomová práce, ČVUT FEL, Praha 2005

[4] William Shoaff: Color, Illumination models, and Shading, http://www.cs.fit.edu/~wds/classes/graphics/Illuminate/illuminate/

[5] Peter Shirley, R. Keith Morley: Realistic ray-tracing, second edition, A. K. Peters, Ltd., USA, 2003

[6] Paul S. Strauss, A realistic lighting model for computer animators,IEEE Computer Graphics and Applications ,Volume 10, Issue 6, November 1990, IEEE Computer Society Press, USA

[7] Bruce Eckel, Chuck Allison: Myslíme v jazyku C++ 2. díl, Grada, Praha, 2000

[8] www.wxwidgets.org, dokumentace knihoven wxWidgets

[9] nehe.gamedev.net, tutoriály knihovny OpenGL

[10] www.opengl.org/sdk/docs/man, reference knihovny OpenGL

[11] Shunting yard algorithm, Wikipedia, http://en.wikipedia.org/wiki/Shunting_yard_algorithm

7 Obsah CD, kompilace programu

7.1 Obsah CD

Zde popíšeme adresářovou strukturu CD.

- Bc/Bc.pdf – bakalářská práce ve formátu PDF
- Illuminator.zip – distribuční archiv projektu
- Illuminator
  - project – soubory .sln a .vcproj Visual Studia
  - src – zdrojové texty
  - preset – přednastavené vzorce osvětlení (/formula), scény (/scene) a materiály (/material)
  - session – uložené relace
  - doc – uživatelská dokumentace (user_doc.htm), programátorská (tech_doc.htm) dokumentace a dokumentace vygenerovaná ze zdrojových kódů pomocí programu Doxygen (adresář generated)
  - include, lib – knihovny a hlavičkové soubory potřebné pro kompilaci

7.2 Kompilace programu

Zde je popsána kompilace programu v produktu Microsoft® Visual Studio® 2005/2008 pro Windows.

Nejprve je třeba přeložit knihovnu wxWidgets.

- Stáhněte verzi 2.8.8 knihoven wxWidgets na www.wxwidgets.org/downloads a rozbalte/nainstalujte do libovolného adresáře.
- Ujistěte se, že systémová proměnná WXWIN je nastavena na tento adresář.
- V souboru include/wx/msw/setup.h v tomto adresáři vyhledejte a změňte tyto makra:
  - wxUSE_GLCANVAS 1 (potřebujeme OpenGL, což není v implicitním nastavení zahrnuto)
  - wxUSE_IOSTREAMH 0
  - wxUSE_STD_IOSTREAM 0

- Otevřete projekt build/msw/wx.dsw a popř. nechte zkonvertovat na novější typ projektu.
- U všech projektů v tomto solution je třeba nastavit typ runtime knihoven. V Solution Exploreru vyberte všechny projekty, a pomocí kontextové nabídky zvolte Properties. Dále zvolte C++ » Code Generation » Runtime Library , nastavte na *Multithreaded Debug* resp. *Multithreaded* pro konfiguraci Debug resp. Release. Náš projekt i knihovny musí být totiž přeloženy se stejným nastavením knihoven.
- Zkompilujte solution v Debug a v Release konfiguraci.

Teď je potřeba zapsat cestu k Windows SDK do proměnné *PlatformSDKDir*, která není standardně vytvořena při instalaci těchto knihoven. Součástí instalace Visual Studia edice vyšší než Express tyto knihovny jsou, popř. stáhněte „Windows SDK for Windows Server 2008 and .NET Framework 3.5“ pro Visual Studio 2008 a “Windows® Server 2003 SP1 Platform SDK“ pro Visual Studio 2005.

Nyní by se měl náš projekt zkompilovat bez dalších úprav. Soubor .sln se nachází na CD ve složce project\VisualStudio08 (VisualStudio05). Jsou připraveny konfigurace Debug a Release.

8 Uživatelská dokumentace

V této příloze popíšeme požadavky programu na cílový systém, instalaci, prostředí programu a jeho chování. Popis syntaxe vzorce je uveden v následující příloze.

8.1 Požadavky

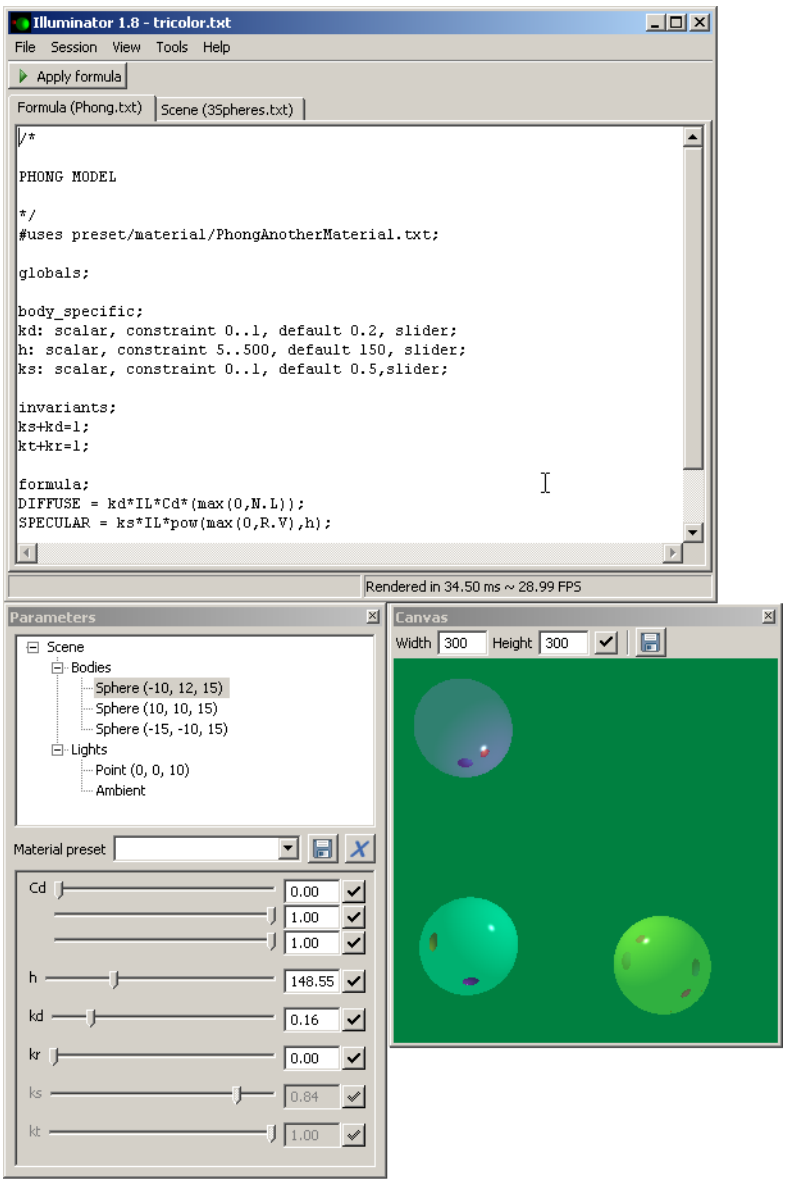
Softwarové požadavky. Cílový operační systém je Microsoft® Windows® 2000, XP nebo Vista. Dalším požadavkem je OpenGl verze 1.1 a vyšší.

Hardwarové požadavky. Minimální konfigurace není stanovena, program by měl fungovat na téměř jakékoli konfiguraci, ovšem s grafickou kartou, která podporuje OpenGl. Doporučená konfigurace:

- procesor 1 Ghz
- operační paměť 256 MB
- dvoujádrový procesor

8.2 Instalace

Program není třeba instalovat, stačí do libovolné složky rozbalit distribuční ZIP soubor. Spustitelný soubor je *Illuminator.exe*.



Obrázek 11: Prostředí programu

Po spuštění program zobrazí tři okna – hlavní okno, okno parametrů a okno pro vykreslování. V závislosti na nastavení aplikace se zobrazí okno logu.

8.1 Hlavní okno

Hlavní okno slouží především pro zadávání definice vzorce a scény.

Obsahuje dvě záložky, jednu pro vzorec (*Formula*) a druhou pro scénu (*Scene*). V každé záložce je otevřen jeden souboru, jehož jméno obsahuje v titulku. Souborové

operace *New*, *Open*, *Save*, *Save as* najdeme v nabídce *File*. Operace se provádí na aktuálním souboru.

Nabídka *View* obsahuje volbu *Show log window*, která přepíná viditelnost okna logu (viz níže).

8.1.1 Okno logu

Vypisuje informace o běhu programu. Zajímavá mohou být měření času výpočtu – doba běhu ray-tracingu, a zejména doba vyhodnocení vzorců a vykreslení obrazu při pohybu posuvníku. Vypisuje se doba v milisekundách a FPS (*frames per second*) – počet snímků za sekundu, které program dokáže vykreslit. Tato informace je průměr času měření, které proběhly při táhnutí posuvníku myší.

8.1.2 Spuštění zpracování vzorce

Tlačítkem *Apply formula* na panelu nástrojů se spustí proces zpracování vzorce a scény. Proběhne kontrola chyb, které jsou hlášeny spolu s pozicí. Nenajdou-li se chyby, je vzorec zpracován, je naplněno okno parametrů parametry podle zadaného vzorce, je naplněn seznam těles ve scéně a poté vykreslen obraz. Pak může uživatel měnit hodnoty parametrů a pozorovat, jak se obraz mění.

Pokud jsou v definici scény či vzorce provedeny úpravy, nejsou reflektovány do okamžiku, než uživatel opět stiskne tlačítko *Apply formula* .

8.1.3 Relace (*sessions*)

Pomocí relace(*session*) je umožněno uložit stav programu, pod tím rozumíme otevřené soubory vzorce a scény, hodnoty všech parametrů a rozměr okna obrazu. Relace se ukládá do textového souboru pomocí nabídky *Session » Open*, a ukládá se pomocí *Session » Save (as)*. Při načtení je obnoveno výše uvedené.

8.2 Okno parametrů

Okno parametrů slouží pro experimentování s parametry.

V horní části okna jsou vypsány tělesa a světla ve scéně, v dolní části parametry s posuvníky. Kliknutím na těleso (příp. na *Scene*) se zobrazí parametry tomuto objektu příslušné. Dbá se přitom na rozdělení parametrů na tělesové, světelné a globální (viz

4.2.4), při vybrání *Scene* se zobrazí globální parametry, při vybrání objektu konkrétní tělesové/světelné.

U každého parametru je zobrazen posuvník na změnu hodnoty. Skalární parametry je možno ovládat jedním posuvníkem, vektorové třemi posuvníky pro každou složku, složky jsou seřazeny vzestupně.

Rozsah hodnot posuvníků je dán definovaným rozmezím pro hodnotu parametru (např. *constraint 1..100*), aktuální hodnota je zobrazena v textovém poli, kde je možno ji také změnit - zadáním hodnoty a stisknutím tlačítka.

Pokud je daný parametr součástí invariantu, může být jeho posuvník zablokovaný. Je to tím, že jeho hodnota je jednoznačně určena hodnotami zbývajících parametrů v invariantu.

8.2.1 Materiály

Program umožňuje uložit sadu tělesových parametrů, nazývanou *materiál*. Sada parametrů zřejmě závisí na osvětlovacím modelu. Každý model může obsahovat na začátku nepovinnou deklaraci `#uses <jméno souboru>;`, která zajistí použití tohoto souboru jako úložiště pro materiály, které uživatel vytvoří. Načítání a ukládání sady materiálů z/do tohoto souboru funguje zcela automaticky.

Lišta s ovládacími prvky pro práci s materiály je zobrazena nad posuvníky s parametry. Lišta je zamknuta, není-li vybráno těleso.

Při úpravách hodnot parametrů nějakého tělesa uložíme materiál stiskem tlačítka *Save material as...* a zadáním jména materiálu. Stejně tak můžeme materiál smazat sousedním tlačítkem. Konečně vybereme-li jméno materiálu z nabídky, program nastaví všechny tělesové parametry na hodnoty materiálu.

8.3 Okno s obrazem

V tomto okně (nazvaném „*Canvas*“) se zobrazuje výsledný obraz.

Je možné libovolně měnit velikost okna, po změně se ihned provede zpracování vzorce a nové vykreslení obrazu. Velikost okna je možné zadat i numericky v textových polích.

Poznámka. Pokud je ve Windows nastaveno zobrazování obsahu okna během přetahování, posílají Windows oknu zprávu na překreslení nejen při přesunu, ale i při

změně velikosti. V tom případě nebude změna velikosti okna obrazu fungovat. Toto nastavení se vypne ve vlastnostech Plochy » karta Appearance » tlačítko Effects » *Show window content while dragging* (české názvy analogicky).

Tlačítko *Save as image* ukládá obraz na disk. Jsou podporovány formáty BMP a PNG.

8.4 Nastavení

Nastavení je přístupné z hlavního okna volbou *Tools » Options*.

Okno obsahuje dvě záložky – *General*, *Default values*.

8.4.1 Obecné (*General*)

- Ray-tracing
  - *Enable shadow ray* – přepíná použití stínovacího paprsku při ray-tracingu
  - *Maximum depth* – hloubka rekurze ray-tracingu. Rozmezí hodnot je 1 až 9. Při hodnotě 1 nejsou zobrazeny odrazy a lomy, při hodnotě 2 jsou zobrazeny odrazy a lomy první úrovně, atd.
  - *Background color* – barva pozadí

8.4.2 Implicitní hodnoty (*Default values*)

- *Parameters* – implicitní hodnoty přednastavených parametrů, viz odstavec 4.2.5
- *Scene*
  - *Default refractive index value* – implicitní hodnota indexu lomu pro tělesa, u kterých nebyl index lomu specifikován v deklaraci
- *Startup session* – jméno relace, která se automaticky načte při startu

9 Syntaxe vstupního jazyka

Zde popíšeme syntaxi jazyka, ve kterém je zadáván vzorec osvětlovacího modelu do programu. Podobný jazyk je použit i pro definici scény, objektů v ní obsažených. Definice vzorce i scény se ukládá do textového souboru, několik jich je obsaženo v distribuci programu.

Jazyk byl navržen tak, aby byl podobný matematickému zápisu vzorce osvětlovacího modelu.

9.1 Obecné informace

Jazyk je citlivý na velikost písmen. Jako oddělovač desetinných míst je používána desetinná tečka. Jsou podporovány komentáře jazyka C, tedy víceřádkové komentáře typu `/* ... */`, bez vnořování.

Jazyk se složen z návěští a příkazů, oddělených středníky. Mezery, konce řádků a tabulátory jsou považovány za bílé znaky a ignorovány.

Protože rovnice osvětlovacích modelů jsou vektorové rovnice, bylo nutno implementovat podporu vektorové aritmetiky. Rovnice modelu je pouze

9.1.1 Syntaktické diagramy

Syntaktické diagramy dále uvedené vznikly z rozšířené Backus-Naurovy formy naší gramatiky. Velkými písmeny značíme neterminál, malými terminál.

Používáme nedefinované neterminály se zřejmým významem:

- **NUMBER** je racionální číslo, s možným unárním mínusem
- **IDENTIFIER** je neprázdná posloupnost písmen anglické abecedy, číslic a podtržítek, kde číslice není na prvním místě.
- C**ONST_VECTOR** je vektor tří čísel tvaru `<NUMBER,NUMBER,NUMBER>`
- **EXPRESSION**, **VECTOR_EXPRESSION** – viz níže

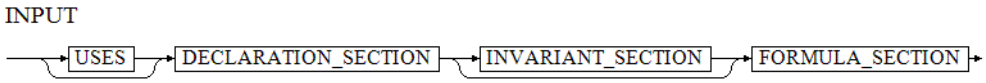
9.1.2 Syntaxe vzorce osvětlovacího modelu

Pro ilustraci uvedeme příklad vzorce Phongova modelu.

```
#uses preset/material/PhongMaterial.txt;

body_specific;
```

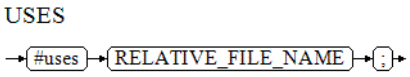
```
kd: scalar, constraint 0..1, default 0.2, slider;
h: scalar, constraint 5..500, default 150, slider;
ks: scalar, constraint 0..1, default 0.5,slider;

invariants;
ks+kd=1;
kr+kt=1;

formula;
DIFFUSE = kd*IL*Cd*(max(0,N.L));
SPECULAR = ks*IL*pow(max(0,R.V),h);

=DIFFUSE + SPECULAR;
```

Zde uvedeme syntaktické diagramy spolu s vysvětlením.



Vstupní text je složen z deklarační sekce a sekce vzorce, v tomto pořadí.

9.1.3 Klauzule uses

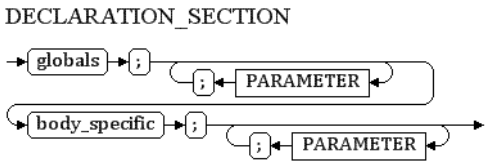


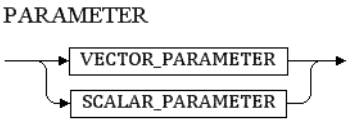
Tato klauzule specifikuje jméno souboru s databází přednastavených materiálů. Program soubor načítá a ukládá zcela automaticky.

Jméno souboru může být relativní, ve tvaru např „preset/material/Phong.txt“, od adresáře ze kterého byl program spuštěn. Cesta může být také absolutní.

9.1.4 Deklarační sekce parametrů

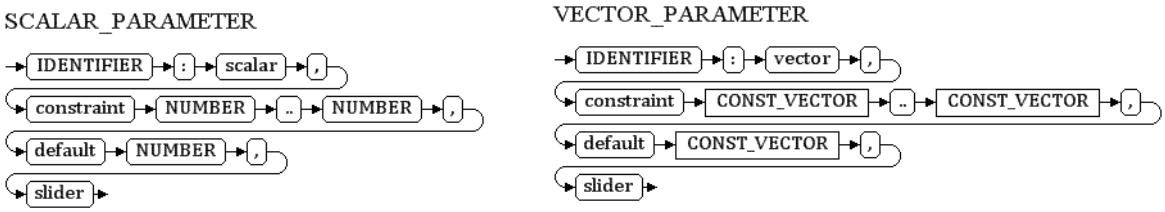
Zde jsou uvedeny deklarace všech parametrů, nejprve globálních, poté tělesových. Deklarace jednotlivých parametrů jsou odděleny středníky.



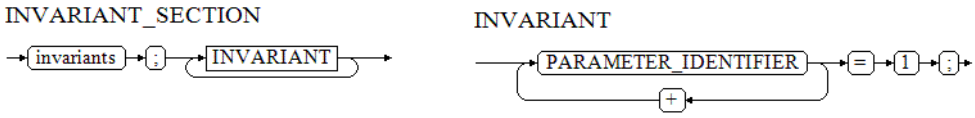


Parametr je dán svým jménem, typem (skalár/vektor), povoleným rozsahem hodnot – tvaru minimum..maximum. Pro vektorové parametry je toto omezení aplikováno na každou složku. Parametr má svoji implicitní hodnotu. Jméno parametru

nesmí kolidovat se jmény proměnných, maker (viz níže), dalších parametrů či funkcí. Jinak je nahlášena chyba.



9.1.5 Sekce invariantů

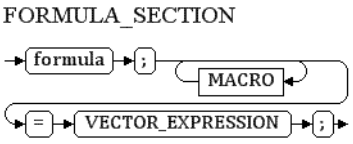


Program podporuje libovolné množství invariantů, pouze však jediného typu – invariant součtu rovného jedna. Důvod zavedení invariantů uvádíme v odstavci 4.2.7.

Jména parametrů musí být definována a parametry musí být skalárního typu s rozsahem (*constraint*) 0 až 1.

9.1.6 Sekce aritmetického výrazu

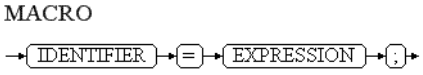
V této sekci je definován aritmetický výraz osvětlovacího modelu.



Na začátku je libovolný počet maker, pak následuje *základní vzorec*.

Makra

Makro je podvzorec označený jménem, zavedený z důvodu zpřehlednění zápisu výrazu a maximální podobnosti s formální definicí modelu.



Makro je dáno svým identifikátorem, který nesmí kolidovat se jmény parametrů, proměnných, funkcí či ostatních maker.

Výskyt identifikátoru makra v hlavním vzorci způsobí vložení textu makra na místo identifikátoru. Proto vektorová i skalární makra nerozlišujeme.

Obsahem makra musí být platný aritmetický výraz; viz níže.

Základní vzorec

Základní vzorec začíná pouze rovnítkem. Výraz za ním následující musí být vektorový, tzn. výsledkem jeho vyhodnocení musí být vektor. Výsledek je chápán jako barva, vektor tří složek RGB s hodnotami od 0 do 1. Syntaxe výrazu viz níže.

9.2 Aritmetický výraz

Výraz představuje neterminál EXPRESSION, a VECTOR_EXPRESSION v syntaktických diagramech výše.

Výrazem je obvyklý infixový výraz využívající operátory, funkce, závorky, a také proměnné, parametry a makra. Výraz se rozlišuje na skalární (vyhodnocením dostaneme skalár) a vektorový.

U každého identifikátoru program určí, jakého typu je, při duplicitě či chybějící deklaraci parametru je hlášena chyba spolu s pozicí znaku, kde byla detekována.

Následuje reference operátorů, funkcí a proměnných.

9.2.1 Priorita operátorů

V následující tabulce jsou operátory, seřazené sestupně podle priority. Jejich reference viz níže.

( )	Závorky; operátor volání funkce
< >	Operátor vytvoření vektoru
-	Unární mínus
^	Umocňování
* / .	Všechny varianty násobení, dělení; skalární součin
+ -	Binární plus, mínus

9.2.2 Funkce a operátory

Zde jsou podrobně popsány podporované funkce a operátory, které je možno je vzorci použít. Funkce jsou zapsány včetně typů, které vrací (skalár/vektor), počtu argumentů a jejich typů.

Funkce se volá s povinnými závorkami a tak, jak je obvyklé – ve tvaru `identifikátor(argument₁,argument₂,...,argumentₙ)`. Operátor se aplikuje tak, jak je v matematice obvyklé, není-li řečeno jinak.

Skaláry v dalším značíme S, vektory V, argumenty funkcí značíme po řadě x, y.

Vrací	Zápis	Argumenty				Popis
		1	2	3	4	
S	-	S				Unární mínus
S	sqrt	S				$\sqrt{x}$
S	floor	S				$\lfloor x \rfloor$
S	getx	V				Získání 1. složky vektoru
S	gety	V				Získání 2. složky vektoru
S	getz	V				Získání 3.složky vektoru
S	pow	S	S			Umocňování; $\mathrm{pow}(x,y) \equiv x^y$
S	^	S	S			Umocňování; x^y
S	exp	S				e^x
S	log	S				$\log_e(x)$
S	sin cos tan	S	S			Goniometrické funkce; úhel se zadává v radiánech
S	* / + -	S	S			Skalární varianty operací
S	.	V	V			Skalání součin; $x \cdot y$
V	* /	V	S			Operace s vektorem zleva a skalárem zprava
V	*	S	V			Násobení se skalárem zleva a vektorem zprava
V	+ -	V	V			Vektorové varianty operací
V	*	V	V			Násobení vektorů po složkách
S	min max	S	S			Minimum, maximum
S	min max	S	S	S		Minimum, maximum
V	<>	S	S	S		Operátor vytvoření vektoru ze tří složek `<x,y,z>`
S	perlin	V	S	S	S	Perlinova šumová funkce tvaru `perlin(pt,a,b,n)` Argumenty pt – bod v prostoru a – alpha – změna amplitudy b – beta - persistence n – počet šumových průběhů, které se sčítají

9.2.3 Proměnné

Proměnné jsou programem vypočítané hodnoty pro každý bod, kde je vyhodnocen model osvětlení. Rovnice může tyto hodnoty libovolně použít pro vyhodnocení barvy. Všechny vektory jsou normalizované.

Zápis v textu	Typ	Zápis v programu	Popis
N	V	`N`	Normálový vektor plochy v bodě průsečíku
L	V	`L`	Vektor ke světlu
V	V	`V`	Vektor k pozorovateli
R	V	`R`	Vektor otočený podle normály, tj. vektor odraženého (*reflected*) paprsku
α	S	`alpha`	Úhel mezi L a N; `cos(alpha)=L.N`
β	S	`beta`	Úhel mezi R a V; `cos(beta)=R.V`
λ	S	`gamma`	Úhel mezi N a V; `cos(gamma)=N.V`
A, B, C, ...	V	`point`	Pozice aktuálního průsečíku v prostoru

9.3 Definice scény

Scéna obsahuje tělesa, při jejichž zobrazení je aplikován daný osvětlovací model, světla a kameru.

Definice scény se ukládá to textového souboru, který program otevře.

Soubor definice scény je zcela oddělen od souboru vzorce – obě definice jsou na sobě zcela nezávislé. Je to v souladu se stavem věci – materiál je nezávislý na tvaru a pozici tělesa, které je z něj utvořeno.

Je důležité si uvědomit, že změna scény nutně vede k opětovnému spuštění ray-tracingu, protože je třeba vytvořit nové vzorce na základě změněných průsečíků paprsků s tělesy.

Modelování těles není předmětem této práce, proto je množina těles poměrně malá.

Všechna tělesa mají tentýž osv. model, přičemž se předpokládá, že jedno těleso má vždy jeden materiál.

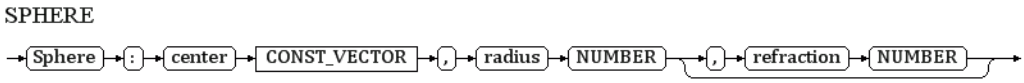
Všechny souřadnice v definici scény jsou chápány jako světové souřadnice (*world coordinates*).

9.3.1 Syntaxe definice scény

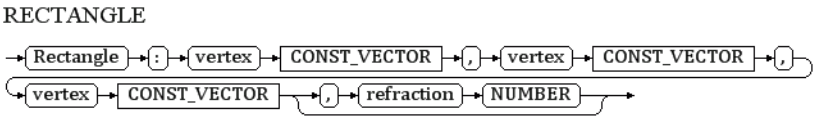
Definice je tvořena deklarací těles, světel a kamery, v tomto pořadí.

Program podporuje dvě tělesa – kouli a obdélník.

Koule je dána středem a poloměrem.

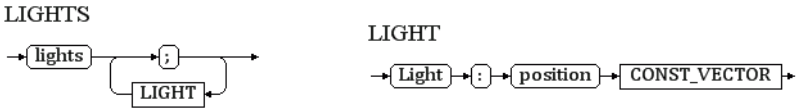


Obdélník je dán třemi svými vrcholy. Vrcholy musí být zadány za sebou po směru hodinových ručiček, při pohledu zepředu (tj. odtud, kam bude směřovat normálový vektor).



Index lomu. Obě tělesa mají nepovinný atribut – index lomu. Není-li zadán, použije se implicitní hodnota z nastavení programu. Index lomu nemůže být parametr, jelikož mění průchod lomených paprsků tělesem. Změna indexu lomu tedy nutně vede k opětovnému ray-tracingu.

Definice světelných zdrojů je přímočará, protože program podporuje pouze bodové všesměrové zdroje.



Kamera je určena (v tomto pořadí) svojí pozicí, vektorem pohledu (*gaze*), vektorem „nahoru“ (*up*), a úhlem pohledu kamery (*field of view*), zadaným ve stupních.