VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

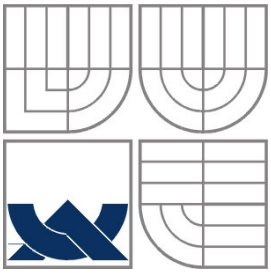
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM SLEDOVÁNÍ PAPRSKU

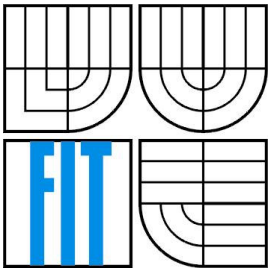
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Miroslav Luňák
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ INTRO 64KB S POUŽITÍM SLEDOVÁNÍ PAPRSKU

GRAPHICS INTRO 64KB USING RAY TRACING

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

MIROSLAV LUŇÁK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. RADOVAN JOŠTH

BRNO 2008

Abstrakt

Tato práce se zabývá popisem tvorby grafického intra 64kB s použitím sledování paprsku.
V dokumentu jsou popisovány problémy a principy související s danou tématikou práce. Dokument dále popisuje vlastní realizaci aplikace a dosažené výsledky práce. Závěr pak obsahuje zhodnocení a možné cesty pro pokračování práce na projektu.

Klíčová slova

intro, sledování paprsku, pixel, globální osvětlování, primární paprsek, sekundárních paprsek, odražený paprsek, průsečík, odraz světla, algoritmus, objekt, světelný zdroj, bilineární interpolace, adaptivní vyhlazování, spekulární složka, difúzní složka, ambientní složka, materiál, rekurze, vektor.

Abstract

This thesis is concerning the description of creation of graphic intro 64kB using ray tracing. The main focus is the problems and principles connected with the topic of the thesis. Further on, the thesis describes the actual realization of the application and the achieved results. The conclusion covers the evaluation of thesis and possible ways for further improvement of the project.

Keywords

intro, ray-tracing, pixel, global lighting, primary ray, secondary ray, reflected ray, intersection, light reflection, algorithm, object, light source, billinear interpolation, adaptive sub-sampling, specular, difuse, ambient, material, recursion, vector

Citace

Luňák Miroslav: Grafické intro 64kB s použitím sledování paprsku. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Grafické intro 64kB s použitím sledování paprsku

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením p. Ing. Radovana Joštha.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Miroslav Luňák
15.5.2008

Poděkování

Chtěl bych tímto poděkovat svému vedoucímu p. Ing. Radovanu Jošthovi, který mi udělil cenné rady při vytváření této práce.

Obsah

1 Úvod

Tato práce se zabývá tvorbou grafického intra s použitím sledování paprsku. Tvorba intra je náročnou záležitostí. Jedná se o grafickou aplikaci zobrazovanou v reálném čase. Na autora aplikace jsou kladeny vysoké nároky, neboť si musí sám navrhnout algoritmus sledování paprsku, pro který neexistuje podpora v žádné grafické knihovně. Následně je algoritmus potřeba aplikovat na scénu, kterou si autor také vytvoří sám.

Nadcházející řádky popisují obsah této bakalářské práce a postupy, které jsem zvolil při jejím řešení.

První kapitola je úvod, který čtenáře seznámí se zevrubným obsahem tohoto dokumentu.

Druhá kapitola pojednává o teorii potřebné pro tvorbu aplikace. Zabývám se v ní popisem algoritmu sledování paprsku, jeho výhod a nedostatků. Jsou zde také popsány akcelerační techniky pro metodu sledování paprsku a model osvětlení.

Ve třetí kapitole popisuji vlastní implementaci řešeného problému. Jsou zde popsány datové typy a funkce použité v programu.

Poslední čtvrtá kapitola je závěr, který pojednává o dosažených výsledcích a nastiňuje další možné pokračování práce.

2 Teorie

2.1 Úvod

V této kapitole se zabývám popisem metod počítačové grafiky, které souvisí s řešením daného úkolu a které jsem při tvorbě grafického intra použil. Popisuji zde metodu sledování paprsku a rozšiřující algoritmy, které byly použity při tvorbě aplikace. Jedna část kapitoly se také věnuje fenoménu grafického intra, popisu jeho vzniku a současným trendům.

2.2 Fenomén grafického intra

Tvorba grafických inter je jednou z nejzajímavějších fenoménů digitálního umění. Tato kultura má počátky v 80 letech 20. století, kdy se lidem do rukou dostávají první osobní počítače. V té době také vznikají první počítačové hry a začíná se rozvíjet počítačové pirátství. Crackerům, kterým se podařilo u hry prolomit ochranu proti kopírování před ní přidávali vlastnoručně vytvořené grafické prezentace. Ty měly představovat jakýsi digitální podpis, podle kterého měli všichni poznat, kdo ochranu prolomil. Grafické intro mělo předvádět zručnost jeho tvůrců a byl u nich kladen velký důraz na umělecké zpracování. Zajímavé je, že intra měla často lepší úroveň grafického zpracování, než samotná hra, před kterou bylo umístěno.

Dnes se intra před hry již neumisťují, ale jsou volně ke stažení na internetu. Prezentují se jako samostatná díla na festivalech nových médií a často jsou k vidění jako součást VJ vystoupení apod. Díky malé velikosti inter je jejich šíření velmi jednoduché. Velkost se pohybuje od 4kB výše. Člověk si říká, že animace, která má velikost pouze 4kB nemůže vypadat hezky. Opak je ale pravdou. I malá intra dokáží zaujmout a sdělit nějakou zajímavou myšlenku.

2.3 Sledování paprsku

Nejprve se budu věnovat popisu metody sledování paprsku, neboť je základním kamenem pro zobrazení scény. Sledování paprsku označuje skupinu metod globálního světlování, které ze 3D scény vytvářejí 2D obraz. V literatuře se pojmem sledování paprsku často míní zpětné sledování paprsku, proto se v následujícím textu budu držet této konvence a pokud to nebude explicitně uvedeno, budu pojmem sledování paprsku označovat zpětné sledování paprsku. Podrobné informace o této metodě, její optimalizaci a programování algoritmu lze nalézt v literatuře [1].

2.3.1 Základy

Sledování paprsku je metoda globálního osvětlování scény. Při jejím návrhu byl kladen důraz na kvalitu výsledného obrazu. Jako základ pro tento algoritmus posloužil reálný svět. V reálném světě jsou paprsky světla vysílány od zdroje rovnoměrně všemi směry do okolního světa, odráží se od předmětů, postupně ztrácí svou intenzitu až úplně zaniknou. Některé paprsky se však odrazí směrem k našemu oku a skončí svou cestu na oční sítnici. Barva každého bodu na sítnici, jež pak tvoří obraz, který vidíme, je součtem barevných příspěvků jednotlivých paprsků, dopadajících na tento bod.

Kdybychom ale měli vytvořit algoritmus, který by pracoval na stejném principu, bylo by zapotřebí výpočetní síly superpočítačů pro vygenerování jednoho snímku. Počet paprsků, které by došly až do lidského oka, by byl jen minimální a většinu času by zabral výpočet nepotřebných paprsků, které by danému obrazu ničím nepřispěla. Algoritmus který takto pracuje se nazývá dopředné sledování paprsku (forward ray tracing). Mezi jeho nevýhody patří právě malý počet paprsků, které se dostanou až k oku pozorovatele a problematický šum ve výsledném obraze.

Tyto nevýhody eliminuje algoritmus zpětného sledování paprsku. Paprsky jsou zde vysílány z oka pozorovatele, a proto se počítá pouze množství nezbytné k vytvoření obrazu.

2.3.2 Princip

Princip vychází z metody vrhání paprsku, kterou obohacuje o možnost rekurze sekundárních paprsků. Před pozorovatele je umístěna průmětna, která je rozdělena na pixely. Jejich počet je dán rozlišením obrazu. Každým tímto pixelem je z oka pozorovatele do scény vystřelován paprsek. Jeho cesta a je sledována skrz scénu. Když se paprsek střetne s nějakým objektem, je v místě střetu vyhodnocena barva a osvětlení. Výsledek je přiřazen pixelu, skrz který byl do scény paprsek vržen. Výsledkem tohoto postupu je dvourozměrný obraz zobrazující trojrozměrnou scénu.

Metoda sledování paprsku tento postup rozšiřuje. Nejprve jsou vrženy paprsky skrz průmětnu a pokud dojde ke střetu s nějakým tělesem, je vyhodnoceno osvětlení pomocí stínových paprsků. Dále je počítán ještě odražený nebo lomený paprsek. Tyto paprsky jsou počítány rekurzivním voláním algoritmu. Paprsek vržený skrz průmětnu je nazýván primární a odražené paprsky jsou nazývány sekundární. Výsledná barva pixelu je určena součtem barevných příspěvků primárního a sekundárních paprsků. Základní algoritmus je popsán takto:

Sleduj_Paprsek(paprsek R, H - hloubka rekurze)

1. Nalezni průsečík P paprsku R s nejbližším tělesem ve scéně
2. Pokud průsečík P neexistuje, přiřaď Paprsku R barvu pozadí a skonči
3. Ke každému světelnému zdroji vyšli z bodu P stínový paprsek a pokud k němu paprsek dorazí, označ světelný zdroj jako nezakrytý

4. Vyhodnoť příspěvky osvětlení v bodě P od všech nezakrytých světelných zdrojů
5. Pokud hloubka rekurze H nepřekročila maximální hloubku sledování , vyšli
    a. odražený paprsek Rr voláním SledujPaprsek(Rr, H+1)
    b. lomený paprsek Rt voláním SledujPaprsek(Rt, H+1)
6. Paprsku R přiřaď výslednou barvu jako součet příspěvků osvětlení, barvy odraženého paprsku Rr a barvy lomeného paprsku Rt

2.3.3 Vlastnosti

Sledování paprsku je metoda, která je schopná generovat obrazy ve vysoké, téměř foto-realistické kvalitě. Je to dáno tím, že se pro každý pixel obrazu barva získává samostatně pomocí primárního paprsku a výsledná barva je případně doplněna barvou sekundárních paprsků. Ve výsledném obrazu jsou tedy zachyceny všechny viditelné objekty scény, včetně jejich odrazů a to velice detailně. Sledování paprsku generuje stíny a odrazy jako přirozený výsledek svého algoritmu, čímž má oproti ostatním renderovacím metodám velkou výhodu. Výhodou je také to, že algoritmus je relativně jednoduchý na implementaci a přesto dává pozoruhodné výsledky. Jednotlivé paprsky jsou na sobě výpočetně nezávislé a proto je algoritmus snadné upravit pro paralelní zpracování.

Základní algoritmus má však také svá omezení. Kvůli použití bodových zdrojů světla není možné v obraze zachytit měkké stíny. Důvod je ten, že pokud na průsečík paprsku a tělesa nedopadá světlo, je mu přiřazena černá barva. Na vedlejší bod však světlo už dopadat může. Tímto v obraze vznikají ostré přechody světla a stínu. Metoda také nedokáže vykreslit difrakci světla na hranách objektů. Je to důsledek toho, že v algoritmu jsou zanedbány vlnové vlastnosti světla. I přes tato omezení je však kvalita výsledného obrazu výborná. Největší nevýhodou této metody však zůstává její časová náročnost. Achillovou patou algoritmu je počítání průsečíku s objekty ve scéně. Pro každý paprsek, jak primární, tak sekundární je nutné počítat průsečík se všemi objekty scény a pak vyhodnotit nejbližší z nich. Pro stínové paprsky to platí také, jen není nutné vyhodnocovat vzdálenost. Tyto výpočty zaberou přes 95% celkového času běhu algoritmu. Pro představu o náročnosti výpočtů uvedu příklad: Mějme scénu s rozlišením 1024x768 pixelů. To znamená 768 432 primárních paprsků. Když vezmeme v úvahu hloubku rekurze 5, dostaneme 3 842 160 primárních i sekundárních paprsků. Ve scéně která bude obsahovat objekt složený z 1000 trojúhelníků, bude potřeba vypočítat téměř čtyři miliardy průsečíků, pokud nepočítáme stínové paprsky. Vykreslení dané scény nám tedy podle její složitosti může trvat několik hodin až dní.

Další nevýhodou této metody je neexistující podpora u grafických procesorů. To znamená, že všechny výpočty jsou prováděny procesorem počítače a grafická karta se stará pouze o vykreslení výsledného obrazu. Aby bylo možné použít sledování paprsků pro grafické intro, je nutné použití akceleračních technik, aby zobrazení obrazů probíhalo v reálném čase.

Obrázek 1 – ukázka metody sledování paprsku: scéna renderovaná programem POV-RAY

2.3.4 Akcelerační techniky

Jak jsem již nastínil v předchozí kapitole, metoda sledování paprsku generuje vysoce kvalitní obrazy, ale je výpočetně velmi náročná. Tento její problém je už dlouhou dobu v popředí zájmu špičkových počítačových odborníků a vývojářů. Vývojové týmy na celém světě věnují akceleračním technikám obrovské úsilí, aby bylo dosaženo co nejmenšího počtu počítaných průsečíků s objekty ve scéně, při zachování výborné kvality obrazu. Současné techniky dokáží urychlit výpočet scény až o 3 řády, což znamená obrovské zrychlení oproti neoptimalizovanému algoritmu.

Rychlost zobrazování je také stěžejní částí aplikace, kterou jsem vyvíjel. Snažil jsem se dosáhnout takové rychlosti zobrazení, aby výsledná animace byla částečně plynulá. K tomu je zapotřebí alespoň 15 snímků za sekundu. Proto v mém návrhu hrály akcelerační techniky velmi významnou roli.

2.3.4.1 Rozdělení akceleračních technik

Akcelerační techniky se dělí do dvou hlavních skupin. První skupinou jsou techniky, které urychlují výpočet průsečíků s objekty ve scéně. Do druhé skupiny patří ty, které se zaměřují na snižování počtu počítaných paprsků. Jejich základní přehled naleznete v následující tabulce.

Akcelerační techniky metody sledování paprsku		
Rychlejší výpočet průsečíků		**Snížení počtu počítaných paprsků**
Rychlejší výpočet průsečíků s objekty	**Snížený počet průsečíků**	
Jednoduché obálky objektů	Hierarchie obálek	Adaptivní řízená rekurze
Speciální výpočet průsečíků pro každý objekt	Dělení prostoru	Adaptivní vyhlazování
	Směrové techniky	

2.3.4.2 Techniky rychlejšího výpočtu průsečíků

Techniky rychlejšího výpočtu průsečíků se dělí na dvě podskupiny.

První podskupinou jsou techniky pro rychlejší výpočet průsečíků s objekty. Pro každý typ objektu je navrhnuta speciální optimalizovaná funkce, která minimalizuje čas potřebný ke zjištění, jestli byl objekt zasažen či nikoliv. Například pro výpočet průsečíku s koulí existuje tzv. geometrické řešení. To má se standardním algebraickým řešením přibližně stejný počet operací, ale geometrické řešení ještě před samotným výpočtem kvadratické rovnice pomocí několika násobení a porovnání zjistí, jestli paprsek kouli mine. Tímto se minimalizuje počet operací potřebných pro zjištění průsečíku a dojde ke značnému zrychlení funkce.

Dále sem patří jednoduché obálky objektů. Komplexní objekty, které mají složité funkce pro zjištění průsečíku se obalí jednoduchými, nejčastěji kulovými obálkami. Tímto se sice do scény přidá další objekt, ale následné testování probíhá rychleji. Pokud paprsek mine tuto obálku, mine i v ní ukrytý objekt, takže není nutné zbytečně počítat složitou funkci. Pokud je obálka zasažena paprskem, je zavolána i funkce pro objekt jí obalený.

Druhou podskupinou jsou techniky, které snižují počet průsečíků.

Patří sem technika hierarchie obálek. Princip je takový, že se každé těleso ve scéně obalí jednoduchou obálkou. Více obálek se znovu obalí novou rodičovskou obálkou. Takto se pokračuje, dokud počet obálek nedosáhne mezní hodnoty, kdy se už nevyplatí přidávat další rodičovské obálky. Při hledání průsečíků se nejdříve otestují rodičovské obálky a pokud je nějaká zasažena, pokračuje testování na její dceřinné obálky. Při zkoumání sekundárních paprsků se toto obrátí a obálky se testují od listu stromu obálek. Tímto přístupem se počet testovaných objektů logaritmicky sníží a pro komplexní scény je uváděno až desetinásobné zrychlení.

Další technikou je dělení prostoru. Ta objekty ve scéně uzavře do co nejmenšího kvádru. Ten je následně rozdělen na menší části, kterým je pak přiřazeno těleso nacházející se v dané části. Při hledání průsečíku se dále vyhodnotí, ve které části scény se paprsek nachází a podle toho jsou

otestovány objekty, jež se v dané části vyskytují. V rozsáhlých a komplikovaných scénách, kde se nalézají např. složité polygonální modely, tato technika přináší také více než 10 násobné urychlení.

Posledními a nejnovějšími přístupy jsou směrové techniky. Ty ve svých rozhodovacích algoritmech berou v úvahu směr vektoru sledovaného paprsku. Používá se zde směrové krychle, která má těžiště umístěné do počátku souřadnicového systému a je zarovnaná s osami. Prostorové vektory jednotlivých paprsků se pak přiřadí pravoúhlým polím na povrchu krychle. Jednotlivým polím jsou poté přiděleny objekty, které leží daným směrem. Mezi tyto techniky patří například světelný buffer, který tyto objekty používá k urychlení výpočtu stínového paprsku.

Výše popsané techniky se používají pro rozsáhlé scény, kdy nám při výpočtech přinesou kýžené urychlení. Pro potřeby grafického intra, které obsahuje pouze omezené množství objektů ve scéně ovšem moc vhodné nejsou. Urychlení, které by tyto techniky poskytly, by bylo minimální a není tedy výhodné je implementovat.

2.3.4.3 Techniky snižující počet počítaných paprsků

Techniky snižující počet počítaných paprsků jsou druhou skupinou akceleračních metod.

Jako první uvedu techniku adaptivního řízení počtu rekurzí. Její princip spočívá ve vyhodnocení příspěvku sekundárního paprsku k paprsku primárnímu. Pokud intenzita barvy přenášená sekundárním paprskem klesne pod určitou stanovenou hranici, rekurze se zastaví a další sekundární paprsky již nejsou sledovány, i když by se ještě mohly dále odrážet od okolních objektů. Aplikováním tohoto přístupu dojde k malému zkreslení výsledného obrazu, je však jen stěží postřehnutelné. Tato metoda také nepřináší algoritmu sledování paprsku příliš velké zrychlení, ale je velmi jednoduché ji do algoritmu začlenit.

Druhou technikou je adaptivní vyhlazování. Tato metoda rozdělí průmětnu na čtverce o zadané velikosti. Následně je pak na každý roh čtverce aplikován algoritmus sledování paprsku. Ten pro každý roh zjistí výslednou barvu a uloží další informace pro rozhodovací metodu. Tyto dodatečné informace jsou například příspěvky osvětlení jednotlivých sekundárních paprsků, informace o objektech zasažených jednotlivými paprsky, nebo stavy světel ve scéně. Na každý čtverec je poté nasazena rozhodovací metoda, která určí, jestli se čtverec bude dále dělit na menší nebo jestli jeho barva bude určena z jeho rohů. Pokud jsou všechny rohy stejné, mohou se zbylé pixely vyhodnotit určenou interpolační metodou. Pokud se alespoň jeden liší, je nutné čtverec dále rozdělit.

Technika adaptivního vyhlazování účinně snižuje počet počítaných paprsků, při zachování vysoké kvality výstupu. Nevýhodou je ztráta některých detailů na objektech samotných, pokud na nich nedochází k odrazům nebo lomům.

2.3.4.4 Shrnutí

Výše zmíněné techniky slouží k optimalizaci algoritmu. Další cesta ke zrychlení výpočtu je použití specializovaného hardware v podobě přídavných karet nebo implementace algoritmu pro počítání na více procesorových systémech.

Pro svůj projekt jsem zvolil techniky adaptivní řízené rekurze a adaptivního vyhlazování. Jsou to jediné vhodné metody z těch, co jsem tu představil. Použití metod pro snižování počtu průsečíků se nevyplatí pro scény s malým počtem objektů a nepřineslo by požadované urychlení.

2.3.5 Světelný model

Osvětlení jednotlivých bodů scény je nejčastěji realizováno Phongovým modelem osvětlení. Tento model navrhl v roce 1977 Bui-Thong Phong. Důvodem pro používání tohoto modelu je jeho snadná implementace a rychlost, při zachování dobré kvality obrazu. Proto jsem se tento model rozhodl použít při návrhu mého programu. Phongův osvětlovací model rozlišuje tři druhy odrazu světla od povrchu tělesa. Odraz je rozdělen na zrcadlový, difusní a ambientní.

Zrcadlová složka je vyjádřena jako

$$I_s = I_L * r_s (\vec{v} \cdot \vec{r})^h \tag{1}$$

kde I_L představuje barevné složení dopadajícího paprsku, $\vec{v}$ představuje normalizovaný vektor pohledu a vektor $\vec{r}$ vyjadřuje směr ideálního zrcadlového odrazu. Koeficient zrcadlového odrazu r_s určuje míru zastoupení zrcadlového odrazu v celkovém odraženém světle, neboli vyjadřuje procentuální zastoupení zrcadlové složky ve světelném modelu. Koeficient h udává ostrost zrcadlového odrazu a může nabývat hodnot v rozmezí $\langle 1 - \infty \rangle$.

Difusní složku obrazu definuje vztah

$$I_d = I_L \cdot r_d (\vec{l} \cdot \vec{n}) \tag{2}$$

kde r_d je koeficient difusního odrazu, $\vec{l}$ je směrový vektor stínového paprsku a $\vec{n}$ je normálový vektor v bodě průsečíku primárního paprsku s objektem scény. Difusní složka je v podstatě barva tělesa. Je to vektor, který udává míru zastoupení jednotlivých složek. Tímto vektorem je vynásobena barva tělesa a tím získáme skutečnou barvu v daném bodě. Příspěvek této barvy je tím větší, čím menší je úhel mezi normálou v místě dopadu a směrovým vektorem stínového paprsku. Je to vlastně interpretace Lambertova zákonu

$$I_d = I \cdot \cos\alpha \tag{3}$$

Ambientní složka je definována vztahem

$$. \qquad I_a = I_A \cdot r_a \tag{4}$$

Tato složka vyjadřuje okolní světlo, které není přímo vyzařováno ze žádných světelných zdrojů. Toto světlo vzniká tak, že se světlo z nějakého zdroje několikrát odrazí (například od stěn v místnosti), a není tedy jasně patrný směr, odkud přichází.

Výsledná barva bodu je pak dána součtem jednotlivých složek Phongova osvětlovacího modelu

$$I_V = I_s + I_d + I_a \tag{5}$$

Metoda sledování paprsku pak výsledné barvě ještě přidává příspěvek odraženého nebo lomeného sekundárního paprsku.

3 Implementace

3.1 Úvod

V této kapitole budu popisovat návrh a implementaci programu.Vlastní implementace byla nejtěžší částí bakalářské práce. Autor se snaží převést své myšlenky a nápady do návrhu programu, ale při vlastní implementaci zjistí, že se v původním návrhu nachází chyba a proto nelze původně „výborný“ nápad vůbec realizovat. Implementaci algoritmu sledování paprsku jsem také musel prakticky od základu předělat, protože až při implementaci člověku dojdou některé souvislosti, které mu při návrhu unikly a zjistí, že k některým problémům musí přistupovat jiným způsobem, než na začátku předpokládal. Je tedy velice důležité věnovat návrhu aplikace velkou pozornost, abychom předešli podobným problémům.

Po počátečních nesnázích se mi implementaci algoritmu podařilo dotáhnout do zdárného konce. Popisuji zde postupy, které jsem použil při návrhu programu tak, aby program odpovídal zadaným specifikacím. V dalších kapitolách na tuto kapitolu naváži a popíšu implementaci samotnou.

3.2 Návrh aplikace

Při návrhu aplikace je nutné si nejprve shrnout požadavky kladené na program, analyzovat je a vytvořit si specifikaci, podle které budeme program navrhovat a implementovat. Často zde uplatňujeme metodu návrhu shora dolů. Je nutné postupovat důsledně, protože dobře promyšlený návrh dokáže při implementaci ušetřit velké množství času, naopak návrh obsahující logické chyby může vést k tomu, že bude nutné velkou část projektu předělat.

3.2.1 Požadavky

Požadavky vyplývají ze zadání této bakalářské práce. Mám vytvořit grafické intro do velikosti 64kB pomocí sledování paprsku. Zadání je komplexní a poměrně složité.

Základní kritéria jsou:

- Program má přehrát uživatelem neměnitelnou animaci, tzv. intro
- Vykreslení scény musí být provedeno metodou sledování paprsku
- Výsledná velikost spouštěcího souboru nesmí přesáhnout 64kB

3.2.2 Specifikace

Program má přehrát neměnitelnou animaci. Uživatel nemůže zasahovat do vzhledu scény, tudíž není nutné implementovat grafické rozraní a metody pro ošetření uživatelských vstupů. Jediná věc, kterou

jsem implementoval je**,** že uživatel se může rozhodnout, chce-li intro přehrát na celé obrazovce nebo v okně. Celoobrazovkový režim se spustí klávesou F, zpět do okna klávesou W. Uživatel také může předčasně ukončit běh programu klávesou Esc.

Jelikož má intro demonstrovat umění svého tvůrce a bývá volně přístupné přes internet, bude koncovým uživatelem člověk s osobním počítačem. To je třeba brát v potaz při nastavování rozlišení výsledného obrazu. Je nutné zajistit, aby se intro vykreslovalo na osobních počítačích rozumnou rychlostí. Pokud by tedy bylo rozlišení zvoleno nevhodně, byla by rychlost vykreslování příliš malá. Jednou z hlavních priorit mého návrhu tedy bylo vytvořit grafické intro běžící na mém osobním počítači s procesorem AMD Athlon 3000+ plynule, vykreslované alespoň 15 snímky za sekundu.

Zřejmě nejdůležitějším bodem specifikace je použití metody sledování paprsku. Jak jsem již popsal výše, je to metoda generující kvalitní výstup charakteristická svými vysokými nároky na výpočetní výkon počítače. Bude tedy zapotřebí optimalizovat algoritmus akceleračními metodami a vykreslovaná scéna musí být poměrně jednoduchá. Ve svém programu používám akcelerační techniku adaptivní řízené rekurze, která výsledný program příliš nezrychlí, ale je nesložitá a nenáročná na implementaci. Druhá technika optimalizace, kterou jsem použil je adaptivní vyhlazování. Tato technika, pokud je dobře implementovaná, dokáže zrychlit základní algoritmus téměř o jeden řád.

Posledním bodem požadavků je výsledná velikost programu nepřesahující 64kB, které lze dosáhnout co nejmenším používáním standardních knihoven jazyka a nastavením kompilátoru tak, aby do spouštěcího souboru neukládal zbytečné informace. Osobně jsem použil program Upx, který ze spustitelného souboru nadbytečné informace odstraňuje. Z původní velikosti souboru, která byla 180kB dokázal soubor zmenšit na velikost kolem 50kB.

Jako implementační jazyk jsem zvolil jazyk C. Jazyk asembler by byl zajímavou alternativou, ale bylo by těžké výsledný kód optimalizovat tak, aby byl stejně rychlý jako kód vygenerovaný překladačem jazyka C.

3.3 Vlastní implementace

V této části popíšu vlastní implementaci hlavních algoritmů programu a přístup, který jsem zvolil při jejich implementaci.

3.3.1 Datové typy

Při navrhování datových typů jsem uplatnil přístup zdola nahoru. Začal jsem od těch nejjednodušších a z nich postupně skládal datové struktury pro použité algoritmy. Na začátku bylo třeba zjistit, jaké datové typy budou v aplikaci nezbytné. Základem celého programu je algoritmus sledování paprsku, který počítá průsečíky paprsků s objekty ve scéně a následně v tomto místě vyhodnotí barvu. Primárním datovým typem jsem tedy zvolil strukturu skládající se ze třech položek typu float

pojmenovanou Vector3. Stejné datové typy jsou také typy pro uložení bodu - Point a typ pro uložení barvy - Color. Tato jednoduchá struktura se ukázala být hlavním stavebním kamenem všech ostatních datových typů. Ostatní datové typy budou popsány vždy u příslušné metody, ke které patří.

3.3.2 Práce s vektory

Vektorové operace jsou nejčastějším výpočetním krokem jak tohoto programu tak metody sledování paprsků obecně. V programu jsem implementoval všechny obvyklé vektorové operace: sčítání a odčítání vektorů, skalární i vektorový součin, normalizace vektoru, velikost vektoru a násobení vektoru číslem. Všechny tyto operace jsou definovány jako funkce se vstupními parametry typu vektor a požadovanou návratovou hodnotou. Jako rozšiřující operace mám definované porovnávání dvou vektorů, které vrací hodnotu 1, pokud jsou vektory stejné a hodnotu 0, pokud jsou rozdílné a násobení vektorů, které vynásobí odpovídající si složky dvou vektorů a vrátí je zpět znovu jako vektor.

Všechny tyto operace používám v programu i pro datové typy Point a Color

3.3.3 Algoritmus sledování paprsku

3.3.3.1 Datové typy

Datové typy, které tento algoritmus používá je struktura Ray, která představuje paprsek. Skládá se z počátečního bodu - Origin a ze směrového vektoru - Direction. Získání paprsku pro daný bod na průmětně zajistí funkce getRay, která podle nastavení kamery získá potřebný směrový vektor a vrátí paprsek pro tento bod.

3.3.3.2 Implementace

Vstupním bodem algoritmu je funkce tracePoint, která jako vstupní parametry obsahuje ukazatel na scénu a souřadnice bodu na průmětně, pro který potřebujeme získat barvu. Funkce nejprve získá paprsek pomocí funkce getRay a následně volá funkci rayTrace.

Funkce rayTrace představuje algoritmus sledování paprsku. Jejím základem jsou funkce hledající průsečík objektu s paprskem. Pro každý typ objektu je používána samostatná funkce, u koule je to funkce getHitSphereDistance, u plochy se tato funkce nazývá getHitPlaneDistance. Vstupními parametry těchto funkcí jsou paprsek - ray, ukazatel na testovaný objekt a ukazatel na typ float – a_distance, která uchovává největší vzdálenost průsečíku paprsku s nějakým objektem ve scéně. Pokud paprsek protíná daný objekt a vzdálenost tohoto průsečíku od počátku paprsku je menší než aktuální vzdálenost a_distance, je a_distance nastavena na tuto hodnotu a je vrácena hodnota HIT. Pokud došlo k zasažení objektu zevnitř, je vrácena hodnota INSIDE_HIT. V případě, že nedošlo k zásahu, vrací funkce hodnotu MISS.

Funkce rayTrace má jako vstupní parametry paprsek ray (ten, který chceme sledovat), pole ukazatelů na objekty scény - objects, ukazatel na typ SquarePoint, který je důležitý pro algoritmus adaptivního vyhlazování a bude popsán v následující kapitole, dále počet objektů ve scéně obj_count, aktuální hloubku rekurze a_depth, aktuální příspěvek barvy paprsku a_color_index a ukazatel na barvu bodu a_pix_color. Funkce postupuje podle algoritmu sledování paprsku popsaném v kapitole 2.3.2. Nejprve testuje průsečíky s objekty ve scéně. Pokud nebyl žádný objekt zasažen, do proměnné a_pix_color je uložena barva pozadí a funkce je ukončena, je-li zasažen zdroj světla, proměnná a_pix_color je nastavena na barvu tohoto světelného zdroje a funkce také skončí. Při zasažení nějakého objektu paprskem jsou vyslány stínové paprsky pro všechny světelné zdroje, jestliže není zkoumaný bod pro daný světelný zdroj ve stínu, nastává vyhodnocení osvětlení podle Phongova světelného modelu popsaného v kapitole 2.3.5.

Jednotlivé hodnoty pro Phongův model jsou uloženy u každého objektu ve struktuře Surface. Ta obsahuje barvu objektu – color a povrchové vlastnosti materiálu. Mezi povrchové vlastnosti materiálu patří koeficient odrazivosti tělesa, koeficient velikosti odlesku, koeficient pro difusní odraz světla na tělese, koeficient lomu světla a index lomu tělesa a příznak.

Předchozím postupem byla vyhodnocena barva primárního paprsku. Sekundární paprsky se generují, umožňují-li to vlastnosti materiálu zasaženého tělesa. To znamená, že odrazivost tělesa nebo intenzita lomeného světla je větší než nula. Další kriteria pro vznik sekundárního paprsku jsou hodnota aktuálního příspěvku paprsku a aktuální hloubka rekurze. Aktuální příspěvek musí být větší než požadované minimum a hloubka rekurze nesmí přesáhnout nastavenou maximální hodnotu. Pro každý sekundární paprsek je znovu volána funkce rayTrace. Výsledná barva pixelu průmětny je pak dána součtem příspěvků primárního a všech sekundárních paprsků.

Funkce rayTrace také ukládá do proměnné typu SquarePoint ukazatele na všechny objekty zasažené primárním nebo sekundárními paprsky a stavy všech světel pro primární paprsek. Nalezené hodnoty pak slouží jako rozhodovací kriteria pro metodu adaptivního vyhlazování.

3.3.4 Algoritmus adaptivního vyhlazování

Adaptivní vyhlazování je stěžejní část celé aplikace. Sledování paprsku by bez implementace tohoto algoritmu nebylo použitelné pro potřeby intra. Proto jsem věnoval jeho návrhu největší pozornost a snažil se ho naprogramovat co nejefektivněji. Základní myšlenka je popsána v kapitole 2.3.4

3.3.4.1 Datové typy

Základem pro tento algoritmus je struktura SquarePoint, která představuje jeden bod průmětny. Struktura obsahuje informaci o barvě bodu, která je uchována typem Color, dále pole typu int – light_state, které zaznamenává stavy jednotlivých světel pro objekt zasažený primárním paprskem a poslední částí je pole ukazatelů na objekty – obj, ve kterém se ukládají ukazatele na všechny objekty zasažené primárními a sekundárními paprsky. Důležitým hodnotou je základní

velikost čtverce pro rozdělení průmětny - SQUARE_SIZE. Průmětnu představuje struktura ImagePlane, která je složena z dvourozměrného pole a má velikost rozlišení obrazu. Každý bod je představován typem SquarePoint.

3.3.4.2 Implementace

Vstupním bodem pro vykreslení obrazu a pro metodu adaptivního vyhlazování je funkce renderScene. Ta má jako vstupní parametr ukazatel na typ Scene. RenderScene rozdělí průmětnu na čtverce o základní velikosti. Na začátku se vždy vykreslí první pod scény a poté se cyklicky procházejí všechny čtverce scény a pro vykreslení každého z nich se volá funkce getSquare.

Funkce getSquare má jako vstupní parametry pozici levého dolního rohu čtverce, velikost čtverce a dva příznaky. Vykresluje nám vždy celý čtverec najednou. Pokud je aktuální velikost čtverce rovná původní velikosti SQUARE_SIZE, jsou vykresleny jen vrcholy čtverce. Při menší velikosti jsou vykresleny i body, které rozdělují čtverec na čtyři menší čtverce. Poté následuje porovnávání jednotlivých čtverců, které zajišťuje funkce checkSquare. Její výsledek rozhodne o nutnosti rozdělení čtverce na čtyři menší. To je provedeno rekurzivním voláním funkce getSquare, které se zadá levý dolní roh menšího čtverce a velikost čtverce je nestavena na poloviční. Toto dělení probíhá tak dlouho, dokud je možno dále dělit, tedy do velikosti strany čtverce rovné jedné. Může se stát, pokud bude scéna složitá, že bude sledováním paprsku vykreslen celý obraz.

Funkce checkSquare má vstupní parametry souřadnice levého dolního rohu čtverce a velikost čtverce. Slouží k rozhodnutí o nutnosti rozdělení čtverce na menší. Funkce nejprve porovná barvu všech vrcholů čtverce. Pokud je barva ve všech místech shodná, je zavolána funkce fillSquare, která vyplní celý čtverec stejnou zadanou barvou. Tento postup je vhodný pro oblasti scény, kde nejsou žádné objekty a paprsky získají jen barvu pozadí. Zde je poté možné celý čtverec touto barvou vyplnit, bez toho, aby bylo nutné použít bilineární filtrování. Pokud není barva stejná, porovnávají se odkazy na objekty zasažené primárním a sekundárními paprsky. Tyto odkazy zapisuje funkce traceRay. Jsou-li shodné i odkazy na objekty, porovnávají se stavy jednotlivých světel. Funkce traceRay nastaví stavy podle toho, jestli daný bod světlo osvětluje či nikoliv. Jednotlivé stavy pro každé světlo scény jsou:

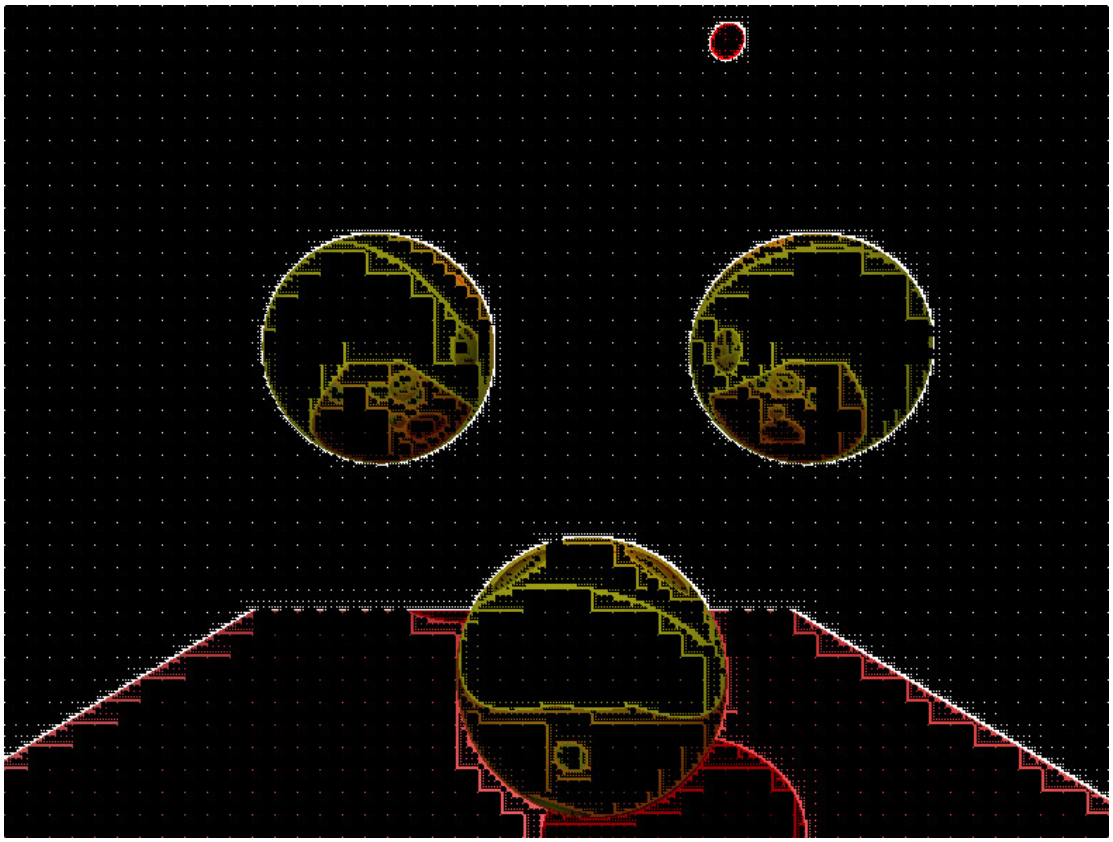
- 0 - pokud je bod ve stínu a není osvětlen daným světlem
- 1 - když je bod osvětlen
- 2 - když je bod osvětlen a zároveň dochází ke spekulárnímu odrazu světla

Pokud se shodují i tyto stavy, je na daný čtverec použito bilineární filtrováni, které interpoluje barvy mezi jednotlivými rohy čtverce.

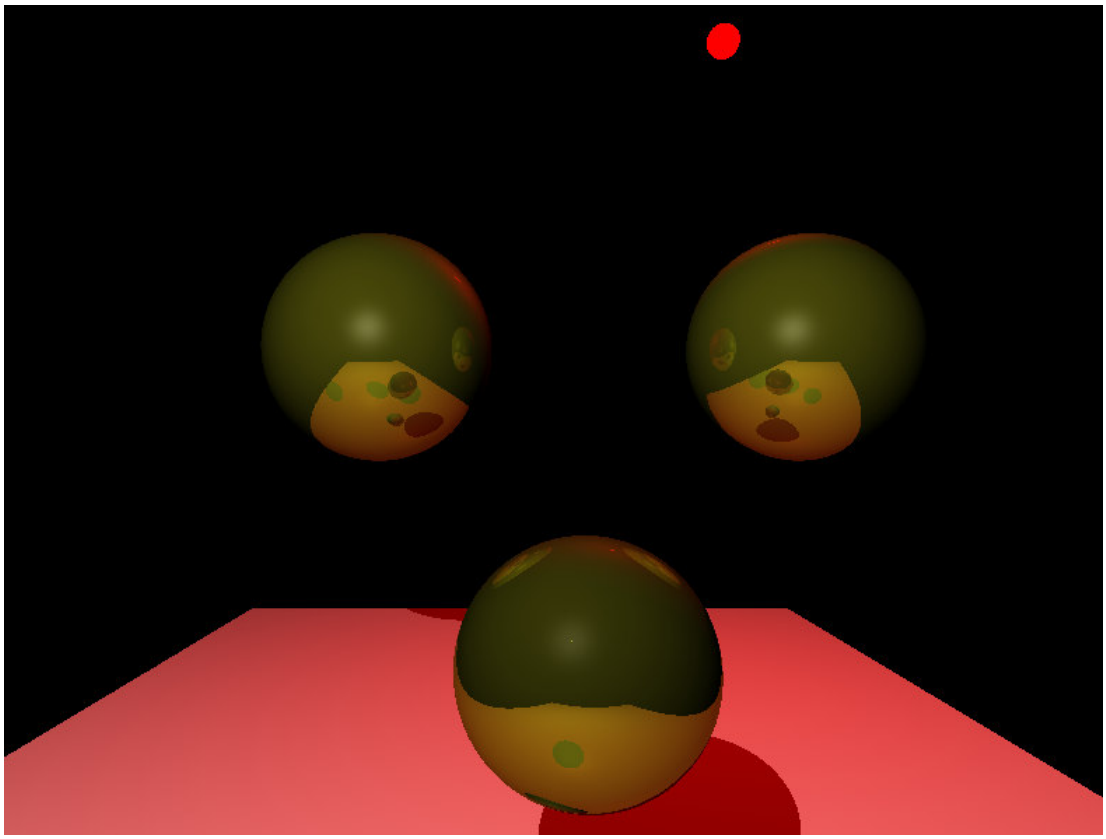
Funkce vrací hodnotu 0, podařilo-li se daný čtverec vyplnit barvou. A vrací hodnotu 1, bude-li nutné daný čtverec ještě rozdělit.

Čtverce se na průmětně se vzájemně překrývají. Díky této optimalizaci se mi podařilo dosáhnout minimálního počtu bodů, které je potřeba zobrazit metodou sledování paprsku. Velikost

vykreslovaného čtverce je v základu nastavena na hodnotu 8. Experimenty se mi podařilo zjistit, že zvětšení čtverce na velikost 17 již výrazné zrychlení nepřinese a jako nežádoucí efekt se objevila ztráta některých detailů a deformace objektů. Velikost čtverce 8 se jeví jako optimální z hlediska kvality i rychlosti algoritmu. Optimalizovaný algoritmus je asi 9x rychlejší, něž kdybychom sledováním paprsku zobrazili celou scénu. Uvedu zde jeden příklad. Při rozlišení průmětny 1024x768 zobrazil optimalizovaný algoritmus 10,5 snímků za sekundu, zatímco algoritmus bez optimalizace pouze 1,2 snímku za sekundu. Test probíhal na počítači s procesorem AMD Athlon64 3000+ a generovaná scéna obsahovala 4 koule, 1 plocha a 2 zdroje světla. Závěrem uvádím dva obrázky modelové scény. První ukazuje pouze body, které v optimalizovaném algoritmu byly zobrazeny sledováním paprsku, druhý ukazuje obrázek kompletně vykreslený optimalizovaným algoritmem.



Obrázek 2- body zobrazené metodou sledování paprsku při použití techniky adaptivního vyhlazování



Obrázek 3 –modelová scéna vykreslená algoritmem sledování paprsku

3.3.5 Scéna

Scéna je realizována datovým typem Scene. Obsahuje pole ukazatelů na objekty scény, ukazatel na typ Color – frame_buffer, ukazatel na projekční plochu image_plane, kameru cam a proměnné které používám pro určování délky trvání animace.

Objekty scény jsou reprezentovány typem Primitive, který obsahuje ukazatel na vlastní objekt, kterým může být například koule nebo plocha, ukazatele na funkce, které lze s objektem provádět, obsahuje příznak, jestli daný objekt je světlo a množinu vlastností povrchu. Vlastnosti povrchu byly popsány v předchozí kapitole.

Objekt scénu typu koule je definován typem Square. Obsahuje souřadnice středu, poloměr a poloměr na 2, který souží pro urychlení výpočtu průsečíků.

Objekt typu plocha je reprezentován datovým typem Plane. Obsahuje střed plochy, normálový vektor plochy, vzdálenost plochy od počátku souřadnic a velikost plochy.

Funkcemi pro každý objekt jsou výpočet průsečíku objektu s paprskem, získání normály pro bod na povrchu tělesa a funkce pro pohyb tělesa ve scéně.

3.3.6 Zobrazení snímku

Vlastní zobrazení snímku je realizováno pomocí knihovny OpenGL. Obraz je uložen v poli frame_buffer, jež je součástí datového typu Scene. Obsah frame_bufferu je poté pomocí příkazů knihovny OpenGL zapsán do paměti grafické karty, přesněji do zadního bufferu a poté je vykreslen přehozením předního a zadního bufferu. V programu tedy využívám doublebufferingu, čímž se zamezí nepříjemnému problikávání obrazu. Více informací o této problematice naleznete v literatuře [2].

4 Závěr

Zadáním byl dán požadavek na vytvoření grafického intra pomocí sledování paprsku a velikostí spustitelného souboru omezenou na 64kB. Intro by mělo být zobrazováno plynule na běžném počítači. Všechny požadavky zadání se mi podařilo splnit a s výsledkem své práce jsem spokojen. Povedlo se mi implementovat algoritmus sledování paprsku optimalizovaný technikou adaptivního vyhlazování. Dosáhl jsem tím plynulého vykreslování animační scény při rozlišení 640x480 bodů, při zachování přiměřené kvality výsledku s ohledem na optické jevy, které přináší použitá technologie. To bych také uvedl jako největší přínos mé práce.

Možnosti dalšího pokračování na této práci jsou široké. Jednou cestou by bylo vylepšení obrazových vlastností. V programu používám pouze jednobarevné objekty. Bylo by vhodné toto rozšířit o možnost na objekty nanést texturu a tím vylepšit vizuální stránku intra. Druhá cesta by spočívala v urychlování algoritmu sledování paprsku a aplikace samotné. Sledování paprsku je výpočetně velmi náročná metoda. Nevýhoda je v tom, že výpočty probíhají pouze na procesoru počítače a grafická karta se stará pouze o vykreslení výsledného obrazu. Tím ovšem zůstává její výpočetní potenciál nevyužit. Bylo by zajímavé pokusit se některé výpočty upravit tak, aby je bylo možné provádět na grafické kartě. Urychlení by také bylo možné dosáhnout tak, že by aplikace byla upravena pro zpracování na paralelních systémech nebo tím, že by aplikace byla naprogramována jako vícevláknová. To by umožnilo využít vysoký výkon moderních dvou a čtyř jádrových procesorů.

Literatura

[1] Glassner, A., S., An introduction to ray tracing, 9, San Francisco, Califonia, Morgan Kaufmannn Publisher, Inc. ISBN 0122861604

[2] Wright S. R., Jr., Lipchak B., Haemel N. OpenGL SuperBible. Boston, Addison-Wesley. 2007 ISBN 0321498828

[3] Buss R., S., 3D Computer Graphics: A mathematical approach with OpenGL. Cambridge University Press 2003 ISBN 0521821037

Seznam příloh

Příloha č. 2 – datové typy

```
// struktura pro uchování vektoru
typedef struct vector3
{
      float x, y, z;          // jednotlivé složky vektoru
} Vector3;



// struktura představující kouli
typedef struct sphere
{
      Point centre;           // střed koule
      float radius;           // poloměr koule
      float radius2;          // poloměr koule na druhou
} Sphere;



// struktura představující plochu
typedef struct plane
{
      float d;                // vzdálenost plochy od počátku souřdanic
      Vector3 normal;         // normálový vektor plochy
      Point origin;           // střed plochy
      Vector3 size;           // rozměry plochy
} Plane;



// struktura pro uchování informací o povrchu objektu
typedef struct surface
{
      Color color;            // barva objektu
      float reflection;       // index odrazivosti
      float diffuse;          // index pro difusní složku
      float specular;         // index pro spekulární složku
      float refraction;       // index lomeného paprsku
      float refraction_index; // relativní index lomu objektu
} Surface;



// struktura představující objekt průmětny
typedef struct primitive
{
      void* object;                       // ukazatel na objekt
      int(*intersect)(Ray,void*,float*);  // ukazatel na průnikovou funkci
      Vector3(*getNormal)(Point,void*);   // ukazatel na funkci normály
      void(*move)(float,float,void*);     // ukazatel na pohybovou funkci
      int is_light;           // příznak, jestli se jedná o zdroj světla
      Surface surf;           // povrchové vlastnosti objektu
      int is_visible;         // příznak, jestli se ma objekt zobrazovat
} Primitive;
```

```
// struktura pro uchování informací o paprsku
typedef struct ray
{
      Vector3 direction;      // směrový vektor paprsku
      Point origin;           // výchozí bod paprsku
} Ray;



// struktura s informacemi o nastaveni kamery
typedef struct cam
{
      Point eye;              // oko pozorovatele
      Point image_plane;      // stred obrazové plochy
      Vector3 sky;            // vektor směřující k obloze
      Vector3 width_vector;   // vektor ve směru šířky průmětny
      vector3 heigth_vector;  // vektor ve směru výšky průmětny
      int width;              // šířka průmětny / 2
      int height;             // výška průmětny / 2
      float diff;             // poměr rozměrů průmětnykrozlišení obrazu
} Camera;



// struktura bodu projekční plochy
typedef struct squarePoint
{
      Color color;                  // výsledná barva bodu
      void* obj[TRACE_DEPTH];       // pole ukazatelů na zasažené objekty
      int light_state[MAX_LIGHTS_COUNT]; // stavy světel
} SquarePoint;



// struktura představující projekční plochu
typedef struct imagePlane
{
      SquarePoint points[TRUE_HEIGHT][TRUE_WIDTH]; // projekční plocha
} ImagePlane;



// struktura představující scénu
typedef struct scene
{
      int count;                    // počet objektů ve scéně
      Primitive* objects[10];       // pole ukazatelů na objekty
      Color* frame_buffer;          // ukazatel na
      ImagePlane* image_plane;      // ukazatel na projekční plochu
      Camera cam;                   // kamera scény
      float how_long;               // čas jak dlouho už animace probíhá
      float last_time;              // čas posledního vykreslení
      int width;                    // šířka projektční plochy
      int height;                   // výška projekční plochy
} Scene;
```