VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

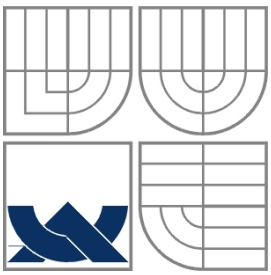
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

METODY REALISTICKÉHO ZOBRAZOVÁNÍ

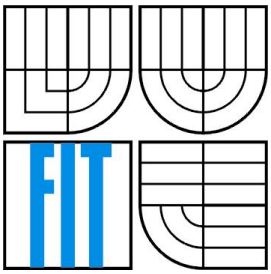
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE IVO VESELÝ
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

METODY REALISTICKÉHO ZOBRAZOVÁNÍ

METHODS OF REALISTIC RENDERING

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

IVO VESELÝ

VEDOUCÍ PRÁCE
SUPERVISOR

DOC. DR. ING. PAVEL ZEMČÍK

BRNO 2008

Abstrakt

Tato práce se zabývá možností běhu realistické grafiky na dnes běžně dostupném technickém vybavení. Za tímto účelem je v práci diskutováno použití metod pro realistické zobrazování, zvláště pak metoda sledování paprsku. Jsou zde řešeny také akcelerační techniky a možnosti abstrakce reality nutné pro reálný chod takového systému. Z těchto technik je diskutována především hierarchie obálek. Část textu se rovněž zabývá doplňujícími technikami pro výstavbu scény a materiálovými vlastnostmi modelů. Pozornost je věnována také osvětlování, neboť bez něj bychom nebyli schopni nic vidět. Velká část práce je zaměřena na techniky propojení standardních zobrazovacích metod a detailů počítaných v software. Na okraj jsou zmíněny alternativní techniky, které jsou považovány za možný budoucí krok.

Klíčová slova

počítačová grafika, realistické zobrazování, metoda sledování paprsku, hierarchie obálek, OpenGL

Abstract

This thesis deals with possibility of operation of realistic graphics on commonly used technical equipment. There are discussed methods of realistic rendering for the purposes of the thesis. Especially the method of Ray tracing. The acceleration techniques and abstractions of reality are explained. They are necessary for real-time running such a time-consuming system. The bounding volume hierarchy is hardly discussed. A part of the text is also focused on additional techniques for scene building and for material characteristics of models. There is a section focused on lighting models because of lighting lets us to see things. Great part of thesis is focused on techniques of connecting standard rendering methods and details computed independently at software. On the other hand there are mentioned alternative techniques that are considered as possible future move.

Keywords

computer graphics, realistic rendering, ray tracing, bounding volume hierarchy, OpenGL

Citace

Veselý, Ivo: *Metody realistického zobrazování.* Bakalářská práce, Brno, FIT VUT v Brně 2008.

Metody realistického zobrazování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Pavla Zemčíka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ivo Veselý
12. května 2008

Poděkování

Rád bych zde poděkoval vedoucímu své práce panu doc. Dr. Ing. Pavlu Zemčíkovi za jeho pomoc při řešení problémů a jeho trpělivost ve chvílích, kdy jsem nechtěl rozumět jeho slovům.

Obsah

1 Úvod

Počítačová grafika je jedním z oborů informatiky. Jejím cílem je zprostředkovat nám informace zpracované počítačem. V případě počítačových her a filmů je navíc jejím úkolem diváka vtáhnout do děje. K tomu je nutné, aby tento uměle vytvořený obraz, neboli scéna, byl co nejvíce podoben realitě.

Realistické zobrazování je ve své podstatě založeno na fyzikálních modelech prostředí a zobrazovaných objektů. Už z definice modelu však vyplývá, že je to pouhá abstrakce reality. Je to dáno tím, že nejsme schopni popsat dané prostředí dokonale především kvůli složitosti a stochastickým jevům. Příkladem může být jednoduchý zlatý prsten. Při nasvícení bude vrhat stín, světlo se od něj bude odrážet a vrhat odlesky na okolní objekty. Avšak prohlédneme-li si ho blíže zjistíme, že není dokonalou kružnicí; že má na povrchu škrábance, trhliny; že materiál má jistou mikrostrukturu. To vše se v realitě projeví na výsledném vjemu. I kdybychom ovšem byli schopni vytvořit dokonalý model s izomorfním vztahem k realitě, nebylo by nám to příliš platné. Je to způsobeno aktuálním stavem vývoje hardware a software, kde kvůli složitosti výpočtů není možné implementovat všechny atributy v plné míře. Z uvedeného tedy vyplývá, že na dnešním stupni vývoje nejsme schopni dosáhnout skutečně reálného zobrazení. To nám ovšem nevadí, stačí zvolit vhodnou míru abstrakce a tím snížit množství detailů a zjednodušit tak komplexní výpočty. Důležitým rozhodnutím tedy zůstává určení hranice mezi množstvím detailů a rychlostí zpracování. Existují složité grafické systémy vyznačující se vysokou kvalitou zobrazení, jejich vlastností je však kromě výkonu také vysoká cena, rozměry a nároky na chlazení a údržbu. Mohou si je dovolit pouze velká studia jako Industrial Light & Magic George Lucase, nebo Pixar Animation Studios. Právě ve filmovém průmyslu není nutností mít výsledek okamžitě k dispozici, připravená animace se sestavuje a renderuje například přes noc. Počítačové hry jsou však spouštěny na běžných, relativně levných osobních počítačích a proto je nutné hledat optimalizace metod pro dosažení vysoké kvality scény při vysoké rychlosti zobrazení.

Každý z Vás již alespoň jednou hrál nějakou počítačovou hru a jistě mi dáte za pravdu, že množství detailů je přímo úměrné požitku ze hry. Aby nebylo nutné s každou nově zakoupenou hrou kupovat také nový hardware je nutné se zabývat optimalizacemi zobrazovacích metod. Aby při projížďce městem s nablýskaným automobilem na jehož kapotě se zrcadlí domy, se tyto domy nezrcadlily i pokud projíždíte po venkově.

Vzhledem k tomu, že nynější hardware ani software není vybaven přímou podporou metod pro realistické zobrazování, je cílem této práce vybudovat systém využívající stávajících prostředků jako jsou běžné grafické knihovny a dostupný grafický hardware. Právě kvůli nepřítomnosti podpory ze strany software je nejdůležitějším krokem propojení grafické knihovny a detailů počítaných mimo ni.

Alternativou tohoto přístupu je vytvoření nové grafické knihovny s již zahrnutou podporou. Takové systémy existují avšak jsou vhodné spíše pro počítačové umění, než pro herní grafiku. Je to dáno vysokou úrovní detailů a minimálními optimalizacemi na úrovni dynamiky scény.

Prvním krokem nutným k nalezení správné cesty k řešení je studium existujících, již používaných metod a jejich optimalizací urychlujících výpočet bez ztráty detailů. Dále je nutné prostudovat možnosti lidského vnímání a tak možnost vyloučit některé detaily ze zpracování. Divák by si jich tak či tak nevšiml. Tato témata jsou diskutována v části věnované metodám realistického zobrazování scény. Druhým krokem je jistě zhodnocení těchto poznatků a výběr vhodných metod a jejich vlastností jako základu pro připravované řešení. Kritické zhodnocení současného stavu je podstatou třetí kapitoly. Jsou zde rovněž diskutovány možnosti napojení na standardní grafickou knihovnu, což je stěžejním tématem práce. V této kapitole jsou také stanoveny parametry kterých by řešení mělo dosáhnout. Čtvrtá kapitola je věnována implementaci a jejímu přínosu. Jsou zde vysvětleny principy a klíčové datové struktury a algoritmy. Na jejím konci jsou uvedeny faktické výsledky vytvářeného systému. Celá práce je zakončena závěrem, kde je diskutováno naplnění cílů a budoucí vývoj projektu a jeho možnosti.

2 Realistické zobrazování scény

Co rozumět pod pojmem „realistické zobrazování scény“ a co je to vlastně ta scéna? Vhodnou analogii je možné najít například v divadle či filmu, kde scéna představuje určitou malou část světa ve kterém se odehrává příběh. Část světa, která je aktuálně v záběru. Scéna je tedy vše (prostředí, předměty, osoby, zvířata atp.), co právě sledujeme prostřednictvím kamery. Realistické zobrazení je takové zobrazení scény při kterém pozorovatel nerozezná scénu uměle vymodelovanou a scénu ze skutečného světa. Zachycenou například na fotografii. Proto je posouzení realističnosti scény čistě subjektivní záležitostí každého člověka a je tedy nutné se zabývat faktory ovlivňujícími tento vjem. Kapitola se zabývá pouze tématy a metodami nutnými pro řešení této práce.

2.1 Faktory realističnosti

Faktory realističnosti mohou být považovány za atributy, které určují míru abstrakce modelu vůči realitě. Geometrie objektů, definice prostředí, charakter světla, vlastnosti materiálů nebo struktura povrchu, to vše může být považováno za faktory ovlivňující realističnost. Skutečně realistická vizualizace není možná. Skutečnost je stochastická, ne empirická nebo matematická. Často je používána zjednodušená geometrie objektů (země je dokonalá koule). Mezi objekty je uvažováno vakuum. Vlnový a kvantový charakter světla je běžně zjednodušen a nahrazen tzv. paprskem (angl. ray). Ten vychází z bodového či homogenního plošného zdroje světla, šíří se všemi směry stejně, rychlost šíření je rovna nekonečnu a jeho intenzita neklesá se vzdáleností. Nepřesnosti, nepravidelnosti, nehomogenity materiálu jsou abstrahovány. Povrch je geometricky hladký, mikro a makro struktura je realizována na úrovni textur. [1]

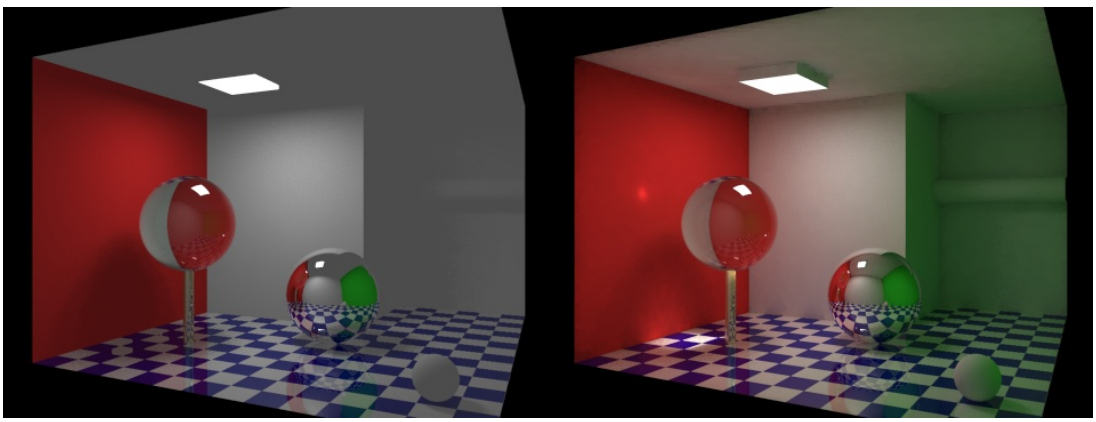
Objektové	Obrazové	Komplexní
Malá realističnost	Velká realističnost	Největší realističnost
Velmi rychlé zpracování	Pomalé zpracování	Velmi pomalé zpracování
Objekty scény jsou zpracovávány sekvenčně	Pixely obrazu scény jsou zpracovávány sekvenčně	Scéna je zpracovávána celá najednou
Žádné vztahy mezi objekty tj. žádné stíny	Nejsou globální vztahy ve scéně, není sekundární osvětlení tj. ostré stíny	Měkké stíny
Z-buffer, Scanline rendering	Ray tracing, atp.	Radiosity, atp.

Tabulka 2.1: Srovnání vizualizačních metod.

Na úrovni zpracování scény jsou využívány tři přístupy: objektové, obrazové a komplexní vizualizační metody. Základní srovnání jejich vlastností je uvedeno v tabulce 2.1. Obrazové a komplexní vizualizační metody spadají pod jednotné označení globální osvětlovací metody (angl. global illumination). Scéna je nejčastěji sestavena z polygonů. Dnešní grafický hardware je optimalizován pro práci s trojúhlníky (angl. mesh), což je speciální případ vždy konvexního polygonu. Zaručení konvexnosti je důležité pro optimalizaci výpočtů.

2.2 Global illumination

Jak již bylo řečeno výše, global illumination je společný název pro skupinu algoritmů používaných ve 3D grafice pro globální vykreslování scény. Scéna vykreslená použitím některé z globálních metod často vypadá mnohem realističtěji než scéna vykreslená pouze pomocí metod přímého osvětlování (angl. direct illumination). Nicméně je tato technologie mnohem náročnější na výpočetní výkon a tím mnohem pomalejší na generování snímků. Jedním z přístupů je vypočítat si globální osvětlovací model scény, například pomocí radiosity a tento model si uložit s geometrií scény, Tato data mohou být poté použita ke generování snímků s různou polohou kamery bez opětovného počítání osvětlovacího modelu. Použití některé z globálních metod zajistí vznik vztahů objektů ve scéně a tím umožní aplikaci optických pravidel. Globální metody jsou opravdu založeny na optice avšak světlo je zde považováno za paprsek, stejně jako při zkoumání optických jevů ve fyzice. Platí tedy zákon lomu a zákon odrazu. Různé materiály proto mohou mít různé optické vlastnosti a na základě těchto vlastností se světlo šíří od zdroje, láme se, odráží se, je pohlcováno a zanechává stíny. Obrázek 2.1 ukazuje pro srovnání scény vykreslené globální a lokální metodou.



Obrázek 2.1: Srovnání scény vykreslené lokální (vlevo) a globální metodou (vpravo).

Radiosity, Ray tracing, Ray casting, Photon mapping, Beam tracing, Cone tracing, Path tracing, Metropolis light transport a Ambient occlusion jsou příklady algoritmů používaných v global illumination s nepřímým osvětlením (angl. indirect illumination). Jako příklad algoritmů ze skupiny

direct illumination může být (soft) stencil shadows. Tato metoda je velice jednoduchá a rychlá. Poskytuje ostré stíny avšak jejím opakováním můžeme získat i dojem měkých stínů (přívlastek „soft“). Jak již název napovídá je založena na vykreslení části scény do paměti šablony (angl. stencil buffer) a prolnutí takto vytvořených vrstev pomocí matematických operací. Scéna v paměti šablony je posunuta oproti scéně v paměti barvy (angl. color buffer) na základě polohy světelného zdroje. Opakováním postupu s různým posunutím a různou hodnotou parametru funkce skládající vrstvy do výsledného obrazu získáme dojem měkkých stínů. Nikdy však na rozdíl od globálních metod nemohou být v této scéně zajištěny vztahy mezi objekty a tak odraz či lom světla. Odraz bývá prováděn prostým vykreslením odražených objektů znovu přes masku uloženou v paměti šablony. Tento postup není evidentně příliš efektivní kvůli zvyšování počtu objektů scény. Jinou možností je texturování metodou zvanou sférické mapování.

Důležitou operací globálních metod je výpočet průsečíků paprsku a objektů scény a dále výpočet osvětlovacího modelu v daném bodě, případně výpočet přenosu světelné energie. To jsou ale také nejsložitější výpočty celého vykreslovacího procesu. Způsob výpočtu a různé optimalizační metody jsou dány použitým algoritmem. Stěžejními metodami jsou radiosity a ray tracing. Z nich je odvozeno mnoho dalších metod a optimalizací.

Radiosity je globální metoda založená na teorii tepelného vyzařování. Spočívá ve výpočtu množství světelné energie přenášené mezi plochami ve scéně. Metoda byla vyvinuta kolem roku 1950 v oblasti výzkumu transportu tepla. Na problémy vykreslování v počítačové grafice byla aplikována v roce 1984 výzkumným týmem na Cornell University of New York. Obrázek 2.2 poskytuje pohled na scénu vykreslenou pomocí radiosity. Všechny povrchy ve scéně jsou rozděleny na jednu nebo více malých ploch (angl. patches). Pohledový faktor (angl. view factor) je spočítán pro každý pár ploch. Je to vlastně koeficient popisující jak dobře na sebe plochy vidí. Plochy které jsou daleko od sebe nebo k sobě navzájem nejsou rovnoběžné, budou mít malý pohledový faktor. Pokud jsou mezi nimi jiné plochy bude pohledový faktor značně redukován nebo roven nule v závislosti na tom, je-li překrytí úplné nebo částečné. Pohledový faktor je poté použit jako koeficient v linearizované formě vykreslovací rovnice. Vyřešení systému rovnic vede k získání vyzařování nebo-li jasu jednotlivých ploch. Progressive radiosity řeší systém rovnic iterativně a tím je možné zastavit vykreslování v okamžiku, kdy se detaily zdají být dostatečné. Výhodou radiosity je relativně jednoduchý algoritmus k jehož implementaci není nezbytně nutná vysoká matematika. Nevýhodou je velká režie nutná pro výpočet integrálu, který se v rovnici vyskytuje. Pro zjednodušení výpočtu metoda předpokládá, že veškerý rozptyl je dokonale difuzní, pro jiné parametry musí být použit zvláštní

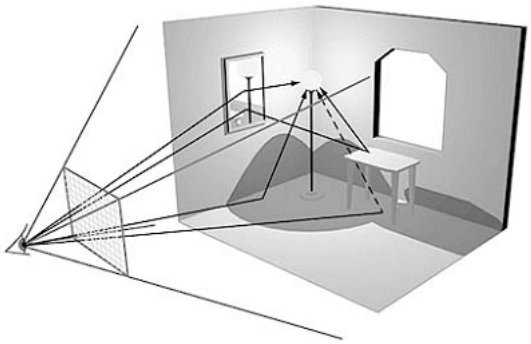
Obrázek 2.2: Radiosity.

přístup, který metodu prodražuje. Má také problémy při náhlých změnách viditelnosti ve scéně a špatně navržených objektech na úrovni návaznosti polygonů. [2]

2.3 Ray tracing a pojem „Back ray tracing“

Jak již název napovídá jedná se o metodu sledování paprsku. Základní koncept této metody vyvinul a popularizoval p. Turner Whitted na přelomu sedmdesátých a osmdesátých let. Ray tracing se stává více a více populárním díky nárustu výpočetního výkonu a kvůli jeho přirozené schopnosti pracovat se stíny a průhledností. Je používán v animaci a je pouze záležitostí času než se dostane do počítačových her. [3]

Paprsky se šíří od světelných zdrojů do scény. Každý paprsek je popsán směrem šíření, svou barvou a intenzitou. Některé paprsky zasáhnou objekty scény, kde se podle jejich optických vlastností lomí, odrážejí a rozptylují. Některé paprsky opustí scénu, některé dopadnou na projekční plochu a ty pak tvoří výsledný obraz scény. Popsaná metoda má však zásadní nedostatek v tom, že sledovat všechny paprsky vycházející ze světelného zdroje je nemožné, navíc mnoho paprsků scénu opustí a na výsledném obrazu se neprojeví, nebo jen minimálně. Proto se zavádí metoda zpětného sledování paprsku (angl. back ray tracing). Ta postupuje přesně naopak, projekční paprsky jsou vrženy skrze pixely obrazu scény a determinují množství světelné energie tímto paprskem přinesené. Protože se klasický ray tracing v praxi nepoužívá bude dále pod pojmy ray tracing a sledování paprsku, nebude-li uvedeno jinak, myšlena metoda zpětného sledování paprsku.



Obrázek 2.3: Zpětné sledování paprsku.

Obrázek 2.3 demonstruje metodu zpětného sledování parsku. Algoritmus funguje jednoduše. Z každého bodu obrazu scény je vyslán tzv. primární paprsek směrem do scény. Existuje-li průsečík s některým z objektů scény, je vyslán tzv. sekundární paprsek, který vzniká odrazem, nebo lomem paprsku, a tzv. stínové paprsky z bodu dopadu ke všem světelným zdrojům pro zjištění leží-li bod ve stínu. Pokud ne je v tomto bodě vyhodnocen osvětlovací model. Výsledný paprsek má barvu danou součtem osvětlení od nezakrytých zdrojů světla, odraženého a lomeného paprsku.

Nevýhodami metody sledování paprsku jsou zmíněné ostré stíny a závislost na bodových světelných zdrojích. Lesklé plochy sice odráží okolí, ale neodráží světlo, nejsou tedy sekundárními zdroji světla. Při sebemenší změně ve scéně je nutné vyhodnotit celou scénu znovu. Zobrazení scény probíhá stále se stejným vzorkováním, není adaptivní, nedokáže se tedy přizpůsobit situaci ve scéně. Existuje mnoho různých variant tohoto algoritmu. Některé vznikly kvůli zvýšení rychlosti výpočtu, jiné kvůli zvýšení kvality zobrazení, všechny však mají společný základ a tím je sledování paprsku. Nejběžnějšími variantami jsou: ray casting, beam tracing, cone tracing, photon mapping a distributed ray tracing.

Ray casting je vlastně ray tracing prvního řádu – vyhodnocují se pouze primární paprsky. V místě kde paprsek protne objekt scény je vyhodnocen osvětlovací model. Nejrozšířenější je tato metoda při přímém zobrazování objektů. Je velice rychlá avšak jejími výsledky jsou pouze vykreslené objekty bez jakýchkoliv interakcí ve scéně. Někdy bývá metoda doplněna tak aby produkovala stíny. V tom případě jsou navíc z průsečíku vysílány stínové paprsky ke světelným zdrojům.

Beam tracing nahrazuje paprsky s průřezem blížícím se k nule, paprsky tvaru n-bokého jehlanu. Řeší tak jisté problémy spojené se vzorkováním a aliasingem, které působí na běžný ray tracing. Nicméně výpočty nutné vykonat navíc jej činí nepopulárním. Aliasing je možné odstranit jinými způsoby a bude zmíněn v kapitole 2.6.

Cone tracing nahrazuje paprsky stejně jako beam tracing. V jeho případě mají spíše kruhový průřez než polygonální.

Photon mapping je metoda používaná k realistické simulaci šíření světla a přenosu světelné energie (např. po odrazu). Specifická je její schopnost simulovat refrakci světla v průhledných materiálech jako sklo nebo voda, odrazy mezi osvětlenými objekty a některé další efekty zapříčiněné částicovými efekty jako jsou kouř a vodní pára.

Největším problémem metody sledování paprsku je, že v případě výskytu odrazivých ploch a relativně malých světelných zdrojů, dochází ke vzniku značného šumu. To je dáno tím, že stínové paprsky redukují šum pouze ze světla přicházejícího přímo ze zdroje. Světlo, přicházející ze zdroje přes odrazivou plochu, dopadající na difuzní plochu vytváří vzory. Velice oblíbeným řešením tohoto problému je vybavení „raytraceru“ technologií zvanou Photon mapping. [3] V prvním průchodu jsou nezávisle sledovány paprsky ze světelného zdroje a paprsky z kamery dokud nevyhoví některé z podmínek pro ukončení. Metoda vysílá ze světelných zdrojů do scény částice nazývané fotony. Kdykoliv foton zasáhne povrch jsou průsečík, směr příchodu a energie fotonu uloženy ve struktuře nazývané „photon map“. Po dopadu na plochu je určen nový směr fotonu s použitím osvětlovací metody BRDF[1]. Existují dvě metody určující jak dlouho se má foton odrážet a putovat tak scénou. První metodou je snižování energie fotonu v každém průsečíku. V okamžiku kdy dosáhne jeho

1 Bidirectional reflectance distribution function je fyzikální osvětlovací model. Více viz. kapitola 2.5.

energie předdefinované prahové hodnoty jeho cesta končí. Druhý algoritmus využívá metodu Monte Carlo v technice zvané Ruská ruleta. [4] V druhém průchodu jsou pro všechny body scény spočítány osvětlovací modely na základě hodnot získaných v prvním průchodu.

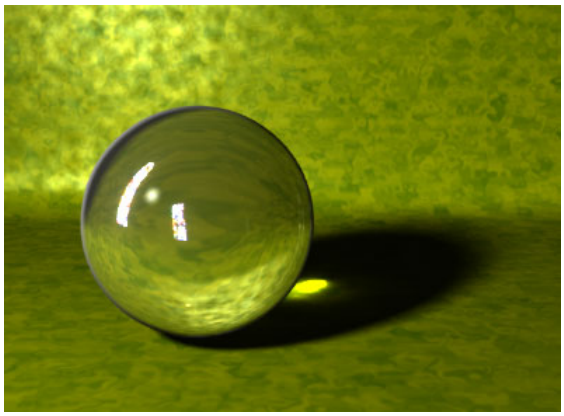
Pro účely této práce považujme foton za množství světla, které má pozici, směr šíření a vlnovou délku λ. Foton má také rychlost *c*, která závisí pouze na indexu lomu *n* média přes které prochází. Někdy je využívána frekvence světla

$$f = \frac{c}{\lambda} \quad , \tag{2.1}$$

což je výhodné zejména proto, že frekvence se na rozdíl od vlnové délky a rychlosti šíření nemění při lomu. Dalším atributem je množství energie *q* nesené fotonem, které je dáno následujícím vztahem:

$$q = h \cdot f = h \cdot \frac{c}{\lambda} \quad , \tag{2.2}$$

kde $h = 6{,}63 \cdot 10^{-34} Js$ je Planckova konstanta. [3]



Obrázek 2.4: Photon mapping.

Běžný ray tracing používá jediný paprsek pro sledování kolizí. Například při výpočtu barvy v daném bodě je z tohoto bodu zasílán jeden paprsek ke každému světelnému zdroji ve scéně. To vede k ostrým stínům, protože všechna světla jsou bodová a mají nulové okolí. Běžný ray tracing také typicky tvoří jeden odražený paprsek a jeden lomený paprsek v každém průsečíku. Výsledkem je, že odražený a lomený obraz jsou dokonale (nerealisticky) ostré. Distributed ray tracing odstraňuje tyto nevýhody průměrováním několika paprsků distribuovaných v určitém intervalu. Například měkké stíny mohou být získány distribucí stínových paprsků do oblasti kolem světelného zdroje. Stejně tak rozmazané odrazy a lomy. [5]

2.4 Matematický aparát

V počítačové grafice je matematika jedním z nejdůležitějších vědních oborů. Je nutné ji tedy věnovat patřičnou pozornost. Určení důležitých oblastí matematiky a jejich aplikace na jednotlivé datové struktury může pomoci při pozdější definici těchto struktur tak, aby byly efektivní. Všechny objekty ve scéně musí mít definovanou polohu a barvu. Jak poloha tak i barva bodu je trojrozměrná informace – tedy vektor. Je tedy nutné definovat efektivně operace nad vektory jako součet, rozdíl, násobení vektoru a skaláru, dělení vektoru a skaláru a dále běžné vektorové operace. Těmito operacemi je skalární součin (angl. dot product), vektorový součin (angl. cross product) a v některých případech může být výhodné definovat i smíšený součin (angl. triple product).

Stěžejním pojmem metody sledování paprsku je „paprsek“. Co to však je? Je to přímka definovaná dvěma body v prostoru či bodem a směrnicí. Je možné ji vyjádřit v několika tvarech jako rovnici. V počítačové grafice je nejběžnější vyjádření parametrickou rovnicí přímky. Tím se dostáváme k další významné části matematiky. Analitické geometrii. Na tomto odvětví matematiky je založen výpočet průsečíků paprsků s geometrickými primitivy. Bude-li scéna vystavěna pomocí CSG geometrie[2], budou se hodit kuželosečky a jejich matematické vyjádření. V případě scény složené z trojúhelníků to zase budou výpočty barycentrických souřadnic.

Máme-li průsečík, je možné spočítat barvu tohoto bodu. Nejprve je nutné zjistit barvu objektu. Tuto informaci nese textura. Aby bylo možné zjistit barvu textury v tomto bodě, je nutné vypočítat texturovací souřadnice. Zde se znovu uplatní barycentrické souřadnice. V případě koule je možné použít sférické souřadnice. Každý typ objektu může mít v rámci optimalizací jinak počítané texturovací souřadnice a nejen ty. Získáme-li barvu objektu, nabízí se výpočet osvětlovacího modelu. To jsou znovu vektorové operace. Musíme však znát normálu povrchu. Výpočet úhlu odrazu a lomu zase vychází z fyziky (pravidlo odrazu, Snellův zákon).

Je vidět, že počítačová grafika zasahuje do mnoha vědních disciplín. Tato kapitola neměla za cíl vysvětlit matematické operace, výpočty, ani fyzikální jevy. Měla za úkol shrnout nutné obory. Výpočet průsečíku se základními primitivy je uveden v příloze 1, 2 a 3.

2.5 Osvětlovací modely a materiálové vlastnosti

„Reálné 3D objekty jsou viditelné jenom díky tomu, že jsou osvětlené.“ [1] Toto je stěžejní myšlenka problematiky osvětlování. V počítačové grafice se dnes používají tři hlavní osvětlovací modely:

- Lambertův osvětlovací model,
- Phongův osvětlovací model,

2 Constructive solid geometry.

- BRDF (bidirectional reflectance distribution function).

Jejich úkolem je modelovat jak povrch objektu odráží dopadající světlo. První ze dvou jmenovaných jsou empirické osvětlovací modely, jsou zjednodušené pro zobrazení v reálném čase. BRDF model je zástupcem fyzikálních modelů, které realisticky popisují vlastnosti světla a jeho odraz. Fyzikální modely jsou však výpočetně značně náročné.

Lambertův osvětlovací model bere do úvahy pouze difuzní[3] složku odraženého světla. Výsledná intenzita bodu je tedy dána vztahem:

$$I = I_A + I_D \quad , \tag{2.3}$$

kde

$$I_D = I_L \cdot C \cdot r_D \cdot (\vec{N} \cdot \vec{L}) \quad . \tag{2.4}$$

Intenzita difuze tedy závisí na intenzitě světelného zdroje I_L, barvě materiálu C, difuzní odrazivosti materiálu danou materiálovou konstantou r_D a úhlem mezi normálou povrchu N a vektorem směřujícím ke světelnému zdroji L. Jinými slovy závisí na úhlu dopadu světla na povrch (tzv. Lambertovo kosínové pravidlo). Složku I_A zatím neberme v úvahu a považujme ji za nulovou.

Phongův osvětlovací model přidává k Lambertovu osvětlovacímu modelu spekulární složku[4]. Výsledná intenzita bodu je tedy dána vztahem:

$$I = I_A + I_D + I_S \quad , \tag{2.5}$$

kde

$$I_S = I_L \cdot r_S \cdot (\vec{V} \cdot \vec{R})^h \quad . \tag{2.6}$$

Intenzita spekulárních odlesků tedy závisí na intenzitě světelného zdroje I_L, spekulární odrazivosti danou materiálovou konstantou r_S a úhlem mezi vektorem V mířícím k pozorovateli a vektorem odrazu R umocněný parametrem h. Parametr h zde vystupuje jako materiálová konstanta udávající ostrost odlesku. Vektor odrazu R vychází z pravidla odrazu, kde

$$R = 2 \cdot (\vec{N} \cdot \vec{L}) \cdot \vec{N} - \vec{L} \quad . \tag{2.7}$$

Vraťme se nyní ke složce I_A. Tato složka se nzývá ambientní[5] a je používána ve scénách vykreslených bez globálních vztahů mezi objekty, neboť bez tohoto světla by scéna byla nereálně

3 Difuzní složka je světlo, které přichází na povrch z jednoho směru a po nárazu na povrch je rozptýleno rovnoměrně do všech směrů. [6]

4 Spekulární složka nebo také zrcadlová složka světla přichází také z konkrétního směru a odráží se od povrchu jedním směrem. Např. kovy mají velkou spekulární složku, křída pak téměř žádnou. [6]

5 Ambientní je světlo, které bylo rozptýlené prostředím natolik, že nelze určit jeho směr. [6]

tmavá. Ve scénách vykreslených s použitím globálních osvětlovacích metod tato složka není teoreticky potřebná. Avšak praxe je jiná. Aby její funkce byla nahrazena přirozenou distribucí světla bylo by nutné mít uzavřenou scénu, tak aby se všechny paprsky mohli ve scéně odrážet a tvořit tak toto ambientní osvětlení. Není-li scéna dokonale uzavřena dochází k úniku paprsků, které se již nikdy nevrátí a tak nepřispějí k osvětlení scény. Ambientní složka vychází ze vztahu:

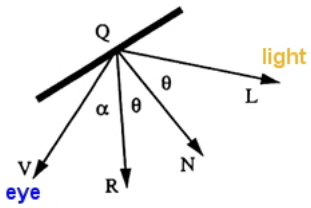
$$I_A = C \cdot r_A \quad , \tag{2.8}$$

kde C je barva materiálu a materiálová konstanta r_A udává ambientní odrazivost materiálu.

Je-li ve scéně přítomno více světelných zdrojů, sčítají se jejich difuzní a spekulární příspěvky. Ambientní složka je počítána vždy jen jednou, protože nezávisí na intenzitě světelného zdroje. Vztah 2.5 se tedy rozšíří následovně:

$$I = I_A + \sum_{i=1}^{n} (I_{D_i} + I_{S_i}) \quad , \tag{2.9}$$

kde n je počet světelných zdrojů ve scéně.



Obrázek 2.5: Phongův osvětlovací model.

Obrázek 2.5 představuje Phongův osvětlovací model, kde Q je průsečík, L vektor směřující ke světelnému zdroji a V takzvaný pohledový vektor (angl. view vector) směřující k pozorovateli.

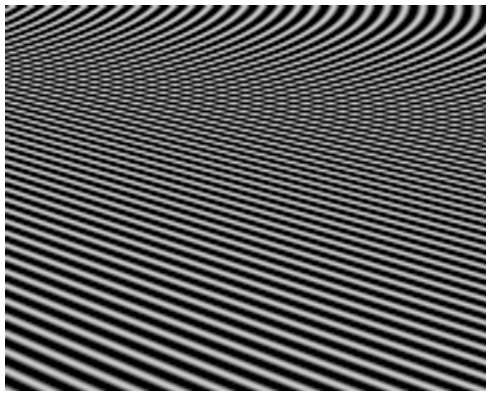
2.6 Aliasing a antialiasing

Aliasing je jev, ke kterému může docházet při převodu spojitého signálu na diskrétní. Tomuto převodu se říká vzorkování (angl. sampling). Aby k aliasingu nedocházelo musí vzorkovací frekvence dodržet tzv. Shannon-Nyquistův teorém. Tento teorém říká, že vzorkovací frekvence musí být rovna minimálně dvojnásobku nejvyšší frekvence obsažené ve vzorkovaném signálu. V počítačové grafice se aliasing objevuje jako zubaté okraje (obrázek 2.6), nebo jako efekt Moire (obrázek 2.7), kde je dokonale vidět zobrazení vysokých frekvencí do nízkých.

Z předešlého je možné usoudit, že pro potlačení aliasingu stačí zvýšit vzorkovací frekvenci. Není to však tak jednoduché. Především proto, že maximální frekvence v obrazu může být značně vysoká a není ji snadné určit. Aliasingu je možné předejít analiticky či diskrétně. Analitické řešení

spočívá ve vyhnutí se rasterizaci, což je v dnešních zobrazovacích technologiích takřka nemožné. Mezi diskrétní metody patří zmíněné zvyšování vzorkovací frekvence zvýšením rozlišení výsledného obrazu. Tím stoupají kvadraticky nároky na paměť a výkon počítače. Další možností je antialiasing. Antialiasing je souhrné označení postupů a metod jak se vyhnout projevům aliasingu.

Obrázek 2.6: Zubaté hrany (angl. jaggies).



Obrázek 2.7: Typický projev efektu Moire.

Existují tři hlavní přístupy: supersampling, náhodné vzorkování (angl. stochastic sampling) a adaptivní vzorkování (angl. adaptive sampling). Bylo definováno mnoho firemních standardů zastoupených mnohdy krkolomnými zkratkami. Všechny ovšem vycházejí z těchto základních přístupů, případně jejich kombinací.

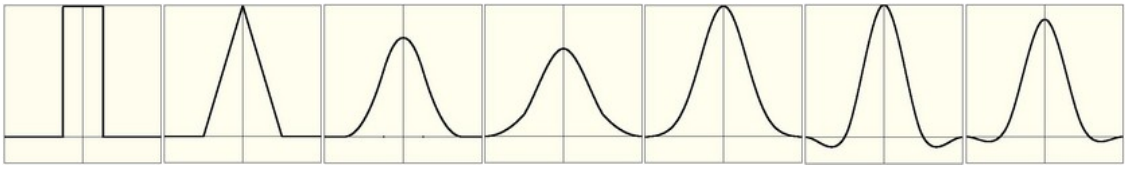
Supersampling spočívá v rozdělení každého pixelu na jistý počet subpixelů a výsledná hodnota pixelu je zjištěna vhodným konvolučním filtrem. Aplikujeme-li tento postup na metodu sledování paprsku, je jednoduše jedním pixelem vrženo několik paprsků. Tato metoda ovšem filtruje celý obraz bez ohledu na kompozici a tak je značně neefektivní kvůli snížení výkonu aplikace.

Metoda stochastic sampling vychází z metody supersampling. Zlepšuje pouze její vizuální vlastnosti. To s sebou samozřejmě nese režii na generování náhodných čísel. Tato režie může být minimalizována například tak, že se mřížka udávající rozmístění vzorků nebude generovat pro každý pixel, ale pro určitou skupinu pouze jednou. Další hojně používanou možností je předdefinovat několik mřížek a z těch náhodně vybírat. Možností rozvoje této metody je mnoho.

Doposud uvedené metody mají zásadní nevýhodu ve značném nárustu náročnosti. Ačkoliv jsou tyto metody implementačně jednoduché. Nejvýznamějším zástupcem metod adaptive sampling je multisampling. Vyznačuje se nízkým snížením výkonu zobrazování ale nemá vliv na zobrazení textur a proto je potřeba doplnit implementaci o filtrování textur. Metoda je založena na násobném vzorkování pouze hraničních pixelů. Všechny pixely uvnitř objektu jsou vzorkovány pouze jedním paprskem.

Pro zvýšení realističnosti je používáno filtrování. Toto filtrování je odlišné od toho o kterém byla řeč doposud. Myšlenkou je, že vrhneme-li pixelem několik paprsků, pak ty blíže ke středu pixelu by měli mít větší váhu, než ty u krajů. V praxi je používán přístup, kdy se na mřížku aplikuje filtr,

který přemístí vzorky tak, že jejich koncentrace na jednotku plochy je největší právě u středu pixelu. Tento přístup je mnohem efektivnější, než přiřazování vah jednotlivým paprskům. Mezi nejpoužívanější filtry patří Box, Tent, Quadratic, Cubic, Gaussian, Catmull-Rom a Mitchell-Netravali.



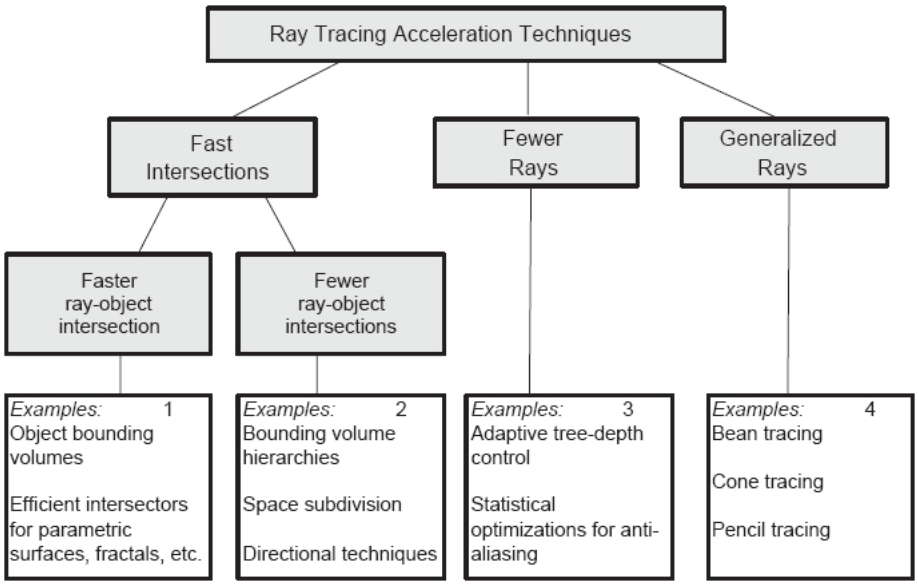
Obrázek 2.8: Filtry, zleva Box, Tent, Quadratic, Cubic, Gaussian, Catmull-Rom a poslední Mitchell-Netravali.

Vzhledem k metodám realistického zobrazování je volba metody odstraňující aliasing důležitá. Tato může značně přispět k vnímané realističnosti scény. Stejně tak ale může značně prodražit a znehodnotit řešení a složité optimalizace zobrazovací metody samotné. Některé zobrazovací metody jsou dokonce bez antialiasingu nepoužitelné kvůli vysokému šumu ve výsledném obraze.

2.7 Optimalizace

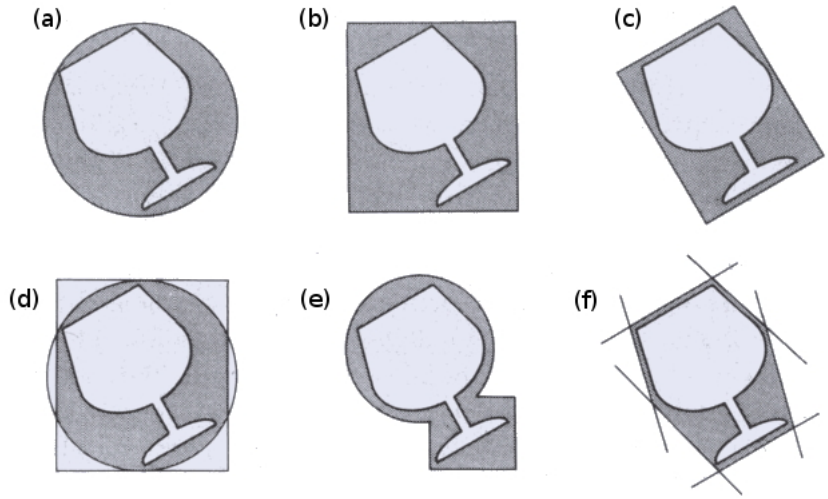
Nevýhodou všech algoritmů pro realistické zobrazování je jejich velká výpočetní náročnost. Základní algoritmus sledování paprsku v podobě v jaké byl definován může scénu vykreslovat i několik hodin. Pro dosažení reálné časové složitosti, je nutné použít optimalizace výpočtu. Optimalizovat je potřeba nejen algoritmy, ale i vlastní kód, to ovšem závisí na implementačním jazyce. Tento typ optimalizací bude probrán v kapitole 4. Při návrhu optimalizace metody sledování paprsku je nutné brát v úvahu následující faktory: aplikovatelnost, výkon, dostupné zdroje a jednoduchost. Základní klasifikace akceleračních technik je na obrázku 2.9 od Jamese Arva a Davida Kirka.

Nalezení průsečíku mezi paprskem a objektem je výpočetně náročná operace. Bez urychlení (hrubou silou), musí být každý paprsek otestován se všemi objekty a poté je nejmenší t (parametr v parametrické rovnici přímky) prohlášeno za nejbližší průsečík. Pro scénu obsahující O objektů, jejímž výsledkem je obraz sestavený z P pixelů, bude složitost $O \cdot P$. Nebudou-li uvažovány výpočty odrazů a lomů a nebudou-li uvažovány metody na odstranění aliasingu. Jeden milión objektů a scéna o velikosti 1024 x 768 pixelů vyústí v nutnost výpočtu skoro osmi set miliard průsečíků v jednom průchodu. [7] Jak je vidět na obrázku 2.9 existuje několik stěžejních metod akcelerace metody sledování paprsku. Mezi ty nejvíce optimalizující patří metody dělení prostoru na podprostory (angl. 3D spatial subdividion) a „obalení objektů obálkami“ které jsou poté hierchicky řazeny (angl. bounding volumes and hierarchies). Ačkoliv toto tvrzení je relativní, protože vše záleží na charakteru scény a objektů.



Obrázek 2.9: Klasifikace akceleračních technik.

Bounding volume hierarchy, neboli hierarchie obálek, byla vyvinuta na School of computing University of Utah. Bounding volume hierarchy je jednoduše strom obálek. Tyto obálky (angl. bounding box) v daném uzlu pouze zapouzdřují obálky potomků. Obálky v listech již zapouzdřují vlastní primitiva. Pokud paprsek mine obálku jistého uzlu, je zřejmé, že mine i všechna primitiva v listech podstromu vycházejícího z tohoto uzlu. Tím je dáno, že s těmito objekty není nutné počítat průsečíky.



Obrázek 2.10: Srovnání různých typů obálek. Každý reprezentuje jiný poměr složitost/chybovost. (a) Koule. (b) Osově zarovnaná obálka. (c) Orientovaná obálka. Pro nižší chybovost může být také použit (d) průnik koule a kvádru, (e) sjednocení koule a kvádru nebo (f) průnik ořezových ploch. [5]

Obálky primitiv mohou mít různé tvary. Čím přesněji bude obálka kopírovat objekt, tím přesnější bude určení objektů, se kterými je nutné počítat průsečíky. Avšak bude-li tvar obálky příliš komplikovaný dojde ke ztrátě výhody kterou obálky představují a to je rychlost výpočtu průsečíku. V případě obálky není důležité, kde se paprsek protne, ale jestli se s obálkou protne.

Nejběžněji používanou metodou jsou díky jednoduchosti implementace osově zarovnané obálky (angl. axis-aligned bounding box). Tyto obálky seřadíme a vystavíme binární objemově zhruba vyvážený strom. Nad vícerozměrnými daty však neexistuje jednoznačně definované řazení. Nejjednodušší cestou jak vybudovat strom je vzít seznam obsahující jednotlivá primitiva a seřadit je podle jedné z os. Poté rozdělit seznam na poloviny a kolem každé z nich vybudovat obálku. Takto je možné postupovat rekurzivně kolem všech os. Tuto metodu uvedl p. Brandon Mansfield pod názvem QSplit. Existuje mnoho způsobů jak lze tuto metodu modifikovat tak, aby produkovala dokonalejší hierarchii a tím i lepší výsledky. [8]

```
najdi střed m obálky uzlu A podél osy O
rozděl A na dva seznamy s délkami k a (N-k) kolem bodu m
levý = new uzel(A[0..k], (O+1) mod 3)
pravý = new uzel(A[k+1..N-1], (O+1) mod 3)
obálka = slouči(levý.obálka, pravý.obálka)
```

Algoritmus 2.1: Metoda QSplit.

Vše je ovšem závislé na typu scény a přítomných primitivech.

3 Shrnutí současného stavu

Kapitola 2 pojednává o současných metodách v oblasti realistického zobrazování. Každá aplikace má však jiný charakter. Je proto nutné vybrat takovou množinu metod, která bude splňovat všechny požadavky na výsledek.

Prvním požadavkem je zajištění globálních vztahů ve scéně. Pro testovací účely bude postačující metoda poskytující bodové světelné zdroje a ostré stíny.Vhledem k pozdějšímu propojení s OpenGL by měla být schopna řídit míru detailů. Budou-li počítány detaily pro jistou část scény měla by metoda umožnit zabývat se pouze touto částí scény bez nutnosti vykreslit ji celou. Komplexní algoritmy nejsou kvůli režii na sestavení scény vhodné. V dynamických scénách herní grafiky se objekty neustále pohybují a ačkoliv není nutné přepočítávat vztahy ve scéně při pohybu kamery, je nutné scénu znovu sestavit při změně pozice objektů a světel. Nejlepším kandidátem jsou tedy metody založené na sledování paprsku. Tato metoda nabízí vynikající možnosti optimalizací. Nebudeme-li požadovat přenos světelné energie odrazem, je vhodnou metodou ray tracing.

Definujme paprsek, který bude vyslán do scény. Pokud se paprsek setká při své cestě scénou s objektem je nutné vypočítat průsečík. Tato operace však vyžaduje optimalizaci. Zkombinujme oba hlavní přístupy. Jednak snížení počtu průsečíků a jednak snížení počtu paprsků. Přístup, kdy se průřez paprsků mění z limitně se blížícího k nule na různé jehlany či kužely je nepraktický. Nepraktický z hlediska vysoké režie na vlastní vytvoření paprsku. Snížením počtu paprsků není myšleno vzorkování každého pátého pixelu, ale dynamické ovládání hloubky zanoření. V nejjednodušším případě je možné napevno definovat počet odrazů a tím hloubku rekurze. Mnohem efektivnějším přístupem je takový, kdy si paprsek nese s sebou informaci o intenzitě. Klesne-li tato intenzita pod určitou mez, přestane se paprsek ve scéně odrážet. Výpočet průsečíků urychlíme vyloučením objektů, se kterými se paprsek protnout nemůže, nebo zjednodušením vlastního výpočtu. Metody zjednodušení vlastního výpočtu se s výhodou používá v CSG geometrii. Ta se vyznačuje především rychlým budováním scény, neboť je vystavěna ze základních tvarů. V našem případě však budou modely sestaveny z polygonů, nebo přesněji z trojúhelníků. To kvůli propojení s OpenGL[6]. Cílem tedy bude co největší snížení počtu počítaných průsečíků s objekty scény. Hierarchie obálek je silným nástrojem pro určení, které objekty se s paprskem pravděpodobně střetnou a které se s paprskem jistě nestřetnou. Pro jednoduchost budou použity osově zarovnané obálky. Výpočet průsečíku s těmito obálkami je rychlý a je možné si některé informace předpočítat. To samozřejmě výpočet urychlí (viz. příloha 3). Struktura binárního stromu dovoluje mnoho algoritmických optimalizací. Jednou z nejdůležitějších je konstrukce stromu v poli. Odstraněním ukazatelů by mělo dojít ke zrychlení operace procházení stromem. Rychlou výstavbu stromu zajistí použití algoritmu QSplit popsaného

6 Dnešní grafické karty se vyznačují velkou mírou optimalizace právě pro trojúhelníky.

v kapitole 2.7. Tento algoritmus má však jeden nedostatek. Negarantuje, že se obálky nemohou překrývat. To může zapříčinit značné zdržení při nutnosti testovat několik primitiv. Kvůli nedefinované operaci řazení vícerozměrných dat, jsou primitiva řazena cyklicky podél os. Pokud je nutné přejít na vedlejší objekt může to znamenat návrat ve stromové struktuře až o tři patra výše a poté znovu provedení všech testů. Je-li tento strom definován v poli je tato režie minimální. Jedním z řešení je udržovat si vztahy mezi sousedy ve stromě. Takový strom by ovšem bylo velice složité sestavit.

Vraťme se tedy k našemu paprsku, který narazil na objekt. V průsečíku proběhl výpočet parametru *t* (parametrické rovnice přímky) a normály (viz. příloha 1 a 2). Normála je nutná pro výpočet osvětlovacího modelu. Z modelů uvedených v kapitole 2.5 se pro naše účely hodí Phongův osvětlovací model. Lambertův nabízí malou úroveň realističnosti. BRDF je výpočetně příliš náročný. Pro výpočet osvětlovacího modelu jsou tedy potřebné informace: intenzita osvětlení v daném bodě, barva povrchu, materiálové vlastnosti a intenzita světla přinášeného odraženým a lomeným paprskem. Také potřebujeme normálu v tomto bodě a pozorovací úhel. Intenzitu osvětlaní v daném bodě získáme součtem výsledku vztahu 2.9 a intenzity světla přinášeného oběma sekundárními paprsky. Pro zvýšení realističnosti je možné uplatnit faktor útlumu (angl. attenuation factor), jehož aproximací je kvadratická funkce vzdálenosti světelného zdroje. Barva povrchu je barvou textury v daném bodě. Texturovací souřadnice nejsnadněji získáme při výpočtu průsečíku. Pro testovací účely bude stačit definovat pouze několik procedurálních textur. Například jednobarevná a šachovnicová textura. Materiálové vlastnosti jsou důležité pro vnímání povrchu objektu. Je tedy nutné znát intenzitu odraženého ambientního, difuzního a spekulárního světla. Také lesk a průhlednost objektu rozhodují. Pro zjištění intenzity světla přineseného odraženým a lomeným paprskem vyšle algoritmus sekundární paprsky ve směru daném příslušnými fyzikálními zákony. Takto se algoritmus rekurzivně opakuje, dokud intenzita paprsku neklesne pod danou mez. Poté se začne osvětlovací model odzadu vyčíslovat až získáme barvu pixelu, ze kterého vzešel prvotní paprsek.

Výsledný obraz získaný RT jednotkou může být zašuměný a především malé objekty mohou mít značné vady kvůli malému množství paprsků, které na ně dopadlo. Pro odstranění tohoto jevu a vyhlazení hran využijeme antialiasing. Supersampling sice zvýší složitost výpočtu ale na rozdíl od multisamplingu nepotřebuje doplňující techniky pro vyhlazení textur. Pro urychlení se mřížka může vygenerovat před započetím vykreslování a po celou dobu se může používat jedna a tatáž. Bylo by možné vygenerovat na začátku mřížek několik a ty v průběhu vykreslování střídat, nebo několik mřížek uložit napevno do zdrojového kódu a náhodně vybírat jednu z nich.

Pro dosažení ještě lepšího výsledku je vhodné na mřížku aplikovat filtr, který rozmístí vzorky tak, aby byl výsledek vyhlazen, ale zachoval si ostré kontury. Přehled základních filtrů je uveden

v kapitole 2.6. Box filtr nemění váhu vzorků. Tent filtr je implementačně jednoduchý a má dobrý poměr právě mezi vyhlazením a ostrostí. Gaussův filtr je známý svým efektem rozmazání, není proto vhodný pro aplikaci na malé detaily. Ty by totiž mohli zcela zaniknout a práce RT jednotky by přišla vniveč. Mitchell-Netravali je velice kvalitní filtr umožňující volit hranici mezi vyhlazením a ostrostí. Je však výpočetně náročný a svou kvalitou se hodí spíše pro statické scény.

Metoda sledování paprsku i její alternativy mohou stát samostatně jako nezávislé zobrazovací jednotky (angl. rendering engine). Avšak už z principu tyto metody nemohou dosahovat takových parametrů jako standardní knihovny. Tyto knihovny používají objektové vizualizační metody a tak dosahují vysokého výkonu za cenu nízkých detailů a ztráty vazeb mezi objekty scény. Nalezneme-li kompromis mezi množstvím detailů a vykreslovacím výkonem je možné tyto dva systémy spojit a nechat je pracovat společně. V podstatě jde o to vykreslit scénu pomocí standardní knihovny. Ta označí objekty, které nebude schopna vykreslit dostatečně realisticky a předá řízení RT jednotce. Tato do scény dodá potřebné detaily a vykreslí scénu na výstupní zařízení. Tato RT jednotka už nemusí procházet celou scénu, ale pouze označené části. Tím dojde k masivnímu zrychlení.

V dnešní době jsou ve velké míře používány dva systémy. Volně dostupná knihovna OpenGL konsorcia ARB a komerční knihovna Direct3D, součást balíku Microsoft DirectX. Vzhledem k licenční politice a vývoji napříč platformami je vhodnější systém OpenGL.

Míra detailů je prvním rozhodnutím, které je nutné učinit. K tomu lze přistoupit v závislosti na tom, pro jaký účel bude výsledné spojení využito. V herní grafice může být například očekáváno reálné zobrazení odrazů na lesklých materiálech a průhledností. Toho je možné dosáhnout označením objektů, jejichž materiálové vlastnosti nabývají určitých hodnot, u kterých by tyto jevy byly pozorovatelné pouhým okem. Například odrazivost materiálu větší než 0,2. V souvislosti s tím je dosaženo další optimalizace – jsou zanedbány detaily, které člověk nebude schopen rozeznat.

Další důležitou otázkou je místo ve vykreslovacím procesu vhodné k propojení obou systémů. Na základě výše uvedených informací je zřejmé, že do procesu vykreslování je nutné vstoupit poté, kdy je již učiněno rozhodnutí o způsobu vykreslení. Zároveň ale před tím, než dojde k finálnímu vykreslení. Vzhledem k nemožnosti vstoupit do procesu rasterizace a vzhledem k tomu, že data poskytnutá RT jednotkou jsou rastrová. Je tedy možné vstoupit do tohoto procesu texturováním, vkládáním pixmap (tzv. billboarding) či přímým přepisem frame-bufferu.

Billboarding se dnes ve velké míře používá v herní grafice k vykreslení vegetace. Princip je následující, místo modelování např. stromu s tisíci polygony se vytvoří z několika málo polygonů plochy, které mají společnou osu y. Na tyto plochy se nanese pixmapa s průhledností a strom je hotov. Billboarding je metodou, která je na první pohled výkonově náročnější. Avšak její síla tkví v možnosti paralelizace výpočtů. Stejně jako v případě uvedeného stromu se namísto daného objektu vykreslí plocha. Tato plocha bude vždy pozicována kolmo na osu kamery. Poté stačí nanést obraz

objektu. Aby nebylo nutné vykreslovat celou scénu, jsou při její výstavbě kritické objekty označeny. Před vlastním výpočtem průsečíku se pak vyhodnotí, zda je nutné průsečík počítat. Při použití této metody se uplatní rychlý přístup k informacím v listech stromu.

Pan Karol Myszkowski z Max-Planckova institutu pro informatiku navrhl jiné řešení. Scéna je plně rasterizována v OpenGL s vyjímkou označených objektů. Ty jsou vykresleny jednotnou, předem danou barvou (tzv. obětní barva). Před finálním vykreslením scény na výstupní zařízení je tato scéna předána RT jednotce. RT jednotka projde tuto pixmapu a v místech, kde nalezne danou obětní barvu vrhne do scény paprsky a vygeneruje tak v těchto bodech obraz scény. Tím dojde k doplnění detailů. Pouze je nutné při vytváření scény v OpenGL dát pozor na osvětlení, aby nezměnilo barvu označených objektů. To by později znemožnilo její identifikaci RT jednotkou. Velkou nevýhodou tohoto řešení je vysoké vytížení sběrnice, na kterou je připojen grafický adaptér. Během jednoho zobrazovacího procesu je totiž nutné dvakrát přesunout celý obraz po sběrnici. Jednou do operační paměti a jednou zpět do grafické paměti kvůli vykreslení na výstupní zařízení.

Jak zvolit takzvanou obětní barvu? Existují histogramy udávající četnosti jednotlivých barev v různých druzích scén. Je také možné si takový histogram vytvořit pro aktuální scénu a tak určit barvu která ve scéně není zastoupena, nebo má nejmenší četnost výskytu. V praxi je to ovšem zbytečné. Jde pouze o to zvolit vhodně takovou barvu, aby nebyla ve scéně zastoupena příliš. V nejhorším případě bude počítáno více detailů než je nezbytně nutné.

Všechny doposud uvedené metody vedou na výpočet detailů v CPU jejich přenos do GPU a vykreslení. Právě přenos rastrových dat po sběrnici je úzkým hrdlem láhve. Je to nejpomalejší část procesu a pokud je nutné tento přenos realizovat několikrát během jediného vykreslení, může to být příčinou nepoužitelnosti metody. Do jaké míry se projeví architektura sběrnice PCIe x16, je již otázkou testů. Existuje mnoho metod jak realisticky vykreslit scénu. Hlavním měřítkem však je, jak jsou tyto metody použitelné na běžném PC. Nejvýznamějšími alternativami jsou: paralelizace procesů a distribuce výpočtů, výpočet detailů přímo v GPU a hardwarové řešení.

Dnešní grafické karty jsou jako počítač v počítači, jsou nezávyslé. Mají vlastní procesor, vlastní operační paměť, je možné je programovat. Proč tedy nevyužít možnosti a nerozdělit výpočet mezi CPU a GPU? Na těchto procesorech může probíhat výpočet současně a tím dojde k urychlení vlastního výpočtu, přičemž ušetřený čas může být využit pro přenos dat z hlavní paměti do paměti grafické karty.

Výpočet detailů přímo v grafickém procesoru a několik dalších technik používá knihovna OpenRT. Ta je alternativou knihovny OpenGL. Jejími vlastnostmi je větší realističnost scény a API kompatiblní s OpenGL. Každé moderní jádro grafického procesoru obsahuje takzvané shaderovací jednotky (vertex shader unit, geometry shader unit a pixel shader unit). Tyto jednotky umožňují ovlivňovat činnost grafického procesoru v různých stádiích vykreslovacího procesu na základě

programu v jazyce GLSL[7]. Většina dnešních grafických jader obsahuje těchto jednotek několik, je tak možné procesy paralelizovat a dosáhnou většího objemu zpracovávaných dat. Reálné testy však ukazují, že jejich výkon především při velkých objemech dat není srovnatelný s CPU. To je dáno především tím, že dnešní CPU mají vysoce optimalizované operace nad souvislými bloky dat a velkou úspěšnost pamětí cache. Více o této problematice viz. práce [9]. Stávající grafická jádra by měla být nahrazena novými platformami, takzvanými GPGPU[8].

Hardwarové řešení je nejsložitější a nejnákladnější alternativou. Je také alternativou přinášející nejlepší výsledky. Avšak ani největší firmy v grafickém průmyslu nejsou za jedno, je-li to krok správným směrem. To dokládá i postoj výkonného ředitele společnosti nVIDIA Corporation Jen-Hsun Huanga: „Hardwarový ray tracing není na hry!“ Prezentovaný na 6th nVIDIA Editor's day in Santa Clara, California.

7 OpenGL Shading Language. V knihově DirectX je to jazyk High Level Shader Language (HLSL).

8 General-Purpose computation on GPU. Například CUDA společnosti nVIDIA Corporation.

4 Doplňování detailů v praxi

Tato kapitola popisuje vlastní implementaci systému majícího přednosti obrazových zobrazovacích metod při snaze udržení rychlosti objektových zobrazovacích metod. Nejprve se zaměří na celkovou strukturu systému a jeho komunikační cesty prostřednictvím hierarchie tříd. A poté projde jednotlivé stavební kameny a přiblíží tak řešení stěžejních struktur a algoritmů. Na konci budou diskutovány výsledky tohoto systému.

4.1 Struktura systému

Systém se skládá ze dvou hlavních částí. První z nich je hlavní program, který propojuje knihovnu OpenGL a RT jednotku. Druhou částí je pak RT jednotka, reprezentovaná třídou Raytracer. Tato třída zapouzdřuje všechny datové struktury a operace nutné pro práci RT jednotky. Nese tedy veškeré informace o scéně prostřednictvím datové struktury scene manager (třída SceneMgr) a hierarchii obálek bounding volume hierarchy (třída BVH). Scene manager udržuje informace o všech objektech scény včetně kamery (třída Camera), světel (třída Light) a grafických primitiv (třída Shape). Každý objekt nese informace o své geometrii, materiálu (třída Material) a své textuře (třída Texture). Takto vypadá základní struktura. Na obrázku v příloze 4 je jasně viditelná kompletní struktura systému. V následujících kapitolách bude rozvedena struktura a funkce jednotlivých stavebních bloků.

4.2 Základní datové struktury

Datové struktury RGB, Vector2 a Vector3 jsou v příloze 4 zakresleny samostatně, neboť jsou základními datovými strukturami a jsou využívány všemi částmi RT jednotky. Tím je také dána důležitost efektivity operací nad těmito datovými strukturami. Všechny často používané metody jsou definovány jako inline funkce. To sice poněkud zvětší výsledný program, ale zásadním způsobem ovlivní rychlost běhu. Překladač sice může odmítnout některé metody definovat jako vložené, avšak většina těch nejdůležitějších jsou tak jednoduché, že by se to stát nemělo. Může se zdát, že některé části kódu porušují zásady objektového přístupu (například porušení zapouzdření). Je to postaveno na tvrzení [3], že u některých překladačů je přímý přístup k položkám struktury rychlejší než přístup k položkám přes metody. Během práce se ukázalo, že používaný překladač se tak nechová[9]. Všechny tři struktury jsou vlastně vektory reprezentující různá data.

9 Všechny zdrojové kódy byly překládány a laděny překladačem gcc verze 3.4.5 v OS Windows XP Professional SP2 a gcc verze 4.2.3 v OS CentOS 64bit Linux.

RGB je struktura uchovávající jak již název napovídá barvu. Má proto tři složky red, green a blue. Nad tímto vektorem jsou definovány základní vektorové operace sčítání, odečítání, změna znaménka. Dále násobení vektorů, dělení vektorů a násobení skalárem a dělení skalárem. Nad touto strukturou je také definována operace porovnání a ořezání (metoda Clamp()). Ořezání pro případ, že potřebujeme zajistit aby v žádném případě nedošlo k opuštění intervalu *[0;1]*.

Struktury Vector2 a Vector3 uchovávají body ve dvou-rozměrném respektive tří-rozměrném prostoru. Jejich struktura i definované operace jsou stejné. Vector3 má pochopitelně vše o jednu souřadnici širší. Složky tedy x, y, případně z. Základní operace jsou stejné jako u struktury RGB. Mají pouze přidáno několik dalších významných operací. Těmi jsou: výpočet délky vektoru (metoda Length()), výpočet délky umocněné na druhou (metoda Length2()) a výpočet jednotkového vektoru (metoda UnitVector()). Dále čistě vektorové operace zmíněné v kapitole 2.4. Skalární součin (metoda DotProduct()), vektorový součin (metoda CrossProduct()) a smíšený součin (metoda TripleProduct()).

4.3 Objekty scény

Výše zmíněná třída Shape je abstraktní virtuální třídou. Abstrahuje rozhraní jednotlivých primitiv tak, že je možné s nimi pracovat jednotným způsobem. Pokud by se systém měl držet striktně modelování pomocí trojúhleníků mohla by tato třída být úplně vypuštěna a nahrazena jejím potomkem, třídou Triangle. Z testovacích důvodů však bylo zvoleno toto řešení, neboť umožňuje rychlé modelování a testování pomocí objektů jako koule (třída Sphere), kterou by bylo jinak nutno modelovat pomocí stovek trojúhleníků. Tato koncepce také umožňuje testovat jiné metody modelování, například CSG. Rozhraní objektů je jednoduché. Třída Shape deklaruje tři metody, které musí mít každý potomek. Metoda Intersect() počítá a vrací průsečík s daným primitivem. Metoda Shadow() počítá zda došlo k protnutí paprsku s primitivem a vrací tedy pouze odpověď booleovského typu. A nakonec metoda BoundingBox(), která vrací obálku tohoto primitiva. Je zřejmé, že všechny tyto funkce jsou pro metodu sledování paprsku stěžejní. Třída Shape také definuje strukturu IntersectInfo, jejíž název napovídá, že jde o strukturu vracenou metodou Intersect(). V případě, že Metoda Intersect() najde průsečík objektu a paprsku, bude tato struktura naplněna následujícími informacemi: parametr *t* (parametrické rovnice přímky), průsečík, normálu v tomto bodě, materiál, texturu a texturovací souřadnice.

Tvorba polygonálních modelů se poněkud komplikuje má-li být program paměťově efektivní. Definujme pro účely tohoto programu datovou strukturu Vertex nesoucí informace o vrcholu trojúhelníku. Takový vertex se skládá z polohy bodu v prostoru, normály v tomto bodě a texturovacích souřadnic pro tento bod. Pokud by každý mesh musel nést informace o třech vertexech, materiálu a textuře, narůstaly by neúnosně nároky na paměť. Tento jev není problémem

pouze tohoto systému nebo RT jednotky, ale je problémem řešeným obecně. Podle [3] je ve všech detailnějších scénách každý vertex sdílen průměrně šesti trojúhelníky. Aby nedocházelo k ukládání redundantních informací, definuje program takzvaný seznam vertexů (angl. vertex list). Každý trojúhelník pak udržuje pouze tři indexy definující záznamy v tomto seznamu. Chybou by také bylo ukládat společně s každým trojúhelníkem odkaz na materiál a texturu. Tak jako tomu je u koule. Často jsou ve scéně modely složeny ze stovek trojúhelníků používajících stejné materiálové vlastnosti a jednu texturu. Všechny tyto problémy jsou vyřešeny třídou Model, která udržuje vertex list a referenci na materiál a texturu. Struktura scény poté vypadá následovně. Scéna obsahuje několik modelů, tyto modely jsou složeny z trojúhelníků. Každý model vlastní odkaz na materiál a texturu. Každý trojúhelník vlastní odkaz na model, kterému náleží a tři indexy do vertex listu.

4.4 Materiály a osvětlení

Materiály a osvětlování jdou ruku v ruce. V programu jsou proto definovány dvě důležité třídy Material a Light. Třída Material udržuje materiálové konstanty nutné pro výpočet osvětlení. Ambientní, difuzní a spekulární odrazivost a parametr udávající ostrost odlesku (shininess). A jiné vlastnosti materiálu jako je index lomu (refractiveIndex). Jednotlivé světelné složky jsou počítány odděleně stejnojmennými funkcemi. Pro získání odraženého a lomeného paprsku definuje metody CreateReflectedRay() a CreateRefractedRay().

Světla jsou bodového charakteru a jsou definována polohou, barvou (intenzitou) a faktorem útlumu. Třída definuje pouze dvě metody, přičemž výsledkem obou je intenzita světla. S tím rozdílem, že druhá z nich uvažuje faktor útlumu. Ten je počítán na základě následujícího vzorce

$$I_L = \frac{C}{a \cdot d^2} \quad , \tag{4.1}$$

kde C je intenzita světelného zdroje, a je faktor útlumu a d je vzdálenost zdroje od místa, kde určujeme jeho intenzitu.

4.5 Texturování

Texturování má na starosti třída Texture. Opět jde o abstraktní virtuální třídu. Výhodou tohoto přístupu je modulární přidávání nových textur, především procedurálních. Abstraktní třída má jednu metodu Value() vracející barvu textury na zadaných texturovacích souřadnicích. Implementovány jsou tři moduly. SolidTexture je modul který definuje jednolitou barvu povrchu. Jeho implementace je tedy jednoduchá. Dalším modulem je ChessboardTexture ten definuje šachovnicový vzor. Vytvoření textury vyžaduje zadání dvou barev, které jsou poté použity jako barvy polí. Implementace

je opět velice jednoduchá. Posledním definovaným modulem je ImageTexture. Ten se ve spojení s třídou BMP stará o texturování objektů obrázkem uloženým ve formátu 24-bit Windows bitmap. Mapování probíhá metodou zvanou Nearest neighbour. Tato metoda je implementačně jednoduchá a rychlá. Její nevýhoda spočívá ve značném projevu aliasingu typu Moire, kvůli absenci jakéhokoliv filtrování. To však není problém, neboť jak bude uvedeno v kapitole 4.7, používá RT jednotka supersampling, který vyhladí i textury.

4.6 Hierarchie obálek

Hlavní akcelerační technikou použitou v programu pro urychlení metody sledování paprsku je tedy metoda nazvaná Bounding volume hierarchy. Její struktura byla popsána v kapitole 2.7 a proto zde budou uvedeny pouze nezbytné detaily řešení. Stromová struktura umožňuje maximálně využít rekurzivního přístupu. Procházení stromem je pak velice jednoduché (viz. algoritmus 4.1). Důležitým avšak možná ne úplně přehledným způsobem je řešen datový typ uzlů. Jsou totiž potomky třídy Shape. Tento přístup byl zvolen právě kvůli rekurzivnímu průchodu stromem. Prochází-li algoritmus strom, není nutné zjišťovat zda jde o list či jen další uzel. Mají totiž stejné rozhraní definované jejich předkem. Vzhledem ke konstrukci stromu nejsou žádné listy označeny jako neobsazené, tedy NULL. Pro vybudování stromu je použita metoda QSplit tak jak byla definována panem Brandonem Mansfieldem. Její vlastnost – možnost překrývání obálek je řešena tak, že funkce Intersect() přijímá jako paramety minimální a maximální hodnoty *t*, tedy parametru v parametrické rovnici přímky. Minimální hodnota je využita pro nápravu zaokrouhlovací chyby ve výpočtu průsečíků. Maximální hodnota je pak určena pro jakousi náhradu Z-Bufferu. Najde-li funkce půsečík v jedné větvi stromu, vrátí jeho parametr *t* a ten je poté použit pro průchod zbytkem stromu. Je-li nalezen další průsečík, je platný pouze pokud jeho parametr *t* je menší než ten aktuální.

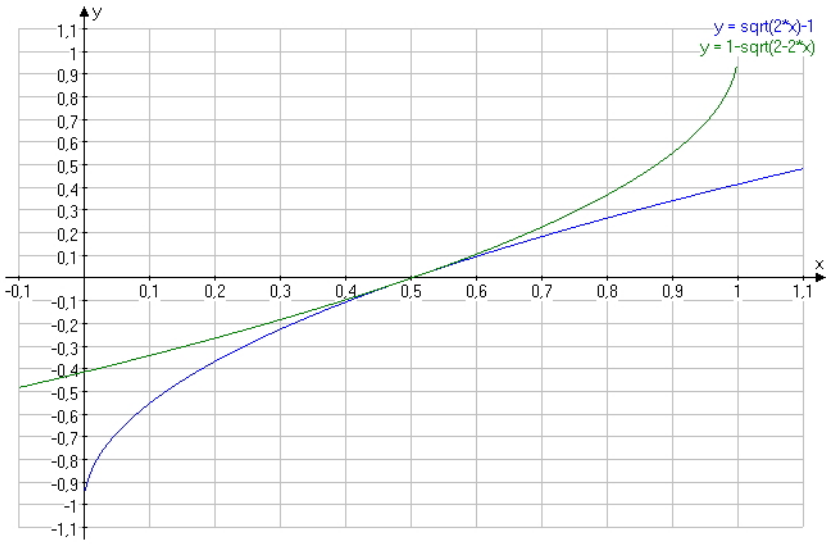
```
BVH::Intersect(min, max, průsečík) {
  if (paprsek prochází touto obálkou) {
    průsečík.t = max
    levýPodstrom = levý.Intersect(min, průsečík.t, průsečík)
    pravýPostrom = pravý.Intersect(min, průsečík.t, průsečík)
    return (levýPodstrom or pravýPodstrom)
  }
  return false
}
```

Algoritmus 4.1: Průchod stromem.

Klíčovou operací této akcelerační techniky jsou však výpočty průsečíků s obálkami. Vlastní výpočet zda se paprsek protne s obálkou nebo ne je uveden v příloze 3. Pro urychlení výpočtů je uvedený vektor *a* předpočítán a uložen v datové struktuře paprsku. To zabrání opakovanému dělení tím, že jej nahradí násobením, které je na většině platforem rychlejší.

4.7 Kamera a vzorkování

Kamera je důležitou součástí scény. Stejně jako u filmu i zde kamera určuje jaká část scény bude vidět a jak. Implementovaná kamera se nevyznačuje žádnými vlastnostmi podporujícími realističtější vzhled scény. Je to proto aby se výsledný obraz co nejvíce podobal svými parametry kameře používané v OpenGL. Kamera je definována bodem ve kterém se nachází, bodem který sleduje a vektorem směřujícím nahoru. Tento vektor (angl. up vector) určuje polohu kamery. Vlastní kamera je definována třídou Camera a RT jednotka, stejně jako OpenGL umožňuje mít v jednu chvíli aktivní pouze jednu kameru. Atributy nastavení kamery jsou v této třídě voleny obdobně jako v OpenGL. Metoda LookAt() nastavuje všechny zmíněné parametry. Analogicky gluLookAt() pro standardní grafickou knihovnu. Každá kamera musí mít také informaci o horizontálním pohledovém úhlu (angl. field of view). Z tohoto parametru je metodou Focus() vypočítána směrnice udávající horní ořezávací plochu pohledového objemu. Ostatní ořezové plochy jsou určeny na základě poměru stran obrazu (angl. viewport aspect ratio) a na základě požadavku na čtvercový průřez pixelů. Hodnotu poměru stran získáme podělením velikostí stran předávaných ústřední metodě CreateRay(). Tato metoda vrací primární paprsek vrhaný do scény. Horizontální pohledový úhel i poměr stran je v OpenGL nastavován funkcí gluPerspective() která navíc nastavuje i přední a zadní ořezovou plochu. Tyto parametry třída Camera neuvažuje.



Obrázek 4.1: Charakteristika použitého filtru.

Jako metoda pro boj s aliasingem je implementován supersampling. Jeho výhodami je implementační jednoduchost a kvalita výsledku. Protože má vliv i na filtrování textur není nutné jej explicitně řešit pomocí složitých algoritmů. Nevýhoda, kterou je časová složitost je potlačena použitím pouze na určitou část obrazu (pouze tam, kde obraz zpracovává RT jednotka). Mřížka

používaná pro supersampling je generována jednou před započetím vykreslování. Pro testování byly definovány pravidelné mřížky s jedním, dvěmi a čtyřmi vzorky. A také stochastická metoda jittering. Ta se vyznačuje dobrými parametry i v oblasti čar v problémových úhlech 0°, 45° a 90°. Pro dosažení dokonalejšího rozložení vzorků implementuje jednotka sampling také Tent filtr. Jeho charakteristika je uvedena na obrázku 4.1. Tento filtr se aplikuje na vytvořenou mřížku voláním funkce TentFilter().

4.8 RT jednotka a propojení s OpenGL

RT jednotka je takový uzel, kde se všechny doposud uvedené algoritmy a datové struktury setkávají a tvoří funkční celek. Nejdůležitějšími částmi třídy Raytracer jsou frame-buffer a privátní metoda Trace(). Frame-buffer je úsek paměti, ve kterém se provádí doplňování detailů. Do této paměti, která je vlastně jen pole hodnot typu float, je zkopírován color-buffer z paměti grafické karty. A z této paměti je po doplnění detailů také obnoven. Algoritmus 4.2 ukazuje detailně tyto přesuny. Je to vlastně stěžejní algoritmus celého systému, neboť definuje právě ono propojení s OpenGL.

```
OnDisplay() {
  //vymaže frame-buffer
  glClear()
  sestav scénu
  označ objekty pomocí maskColor
  //vynutí vykreslení OpenGL scény a počká na dokončení
  glFinish()
  //nastaví buffer ze kterého se přesunou data do RT jednotky
  glReadBuffer()
  //přesune data z OpenGL do RT jednotky
  glReadPixels()
  //doplní detaily do obrazu scény na místa označená maskColor
  RT.DrawDetails(maskColor)
  //nastaví buffer do kterého se přesunou data z RT jednotky
  glDrawBuffer()
  //přesune data z RT jednotky zpět do OpenGL
  glDrawPixels()
  //vykreslí výsledek na výstupní zařízení
  glutSwapBuffers()
}
```

Algoritmus 4.2: Propojení s OpenGL.

Již zmíněná funkce Trace() je naopak duší RT jednotky. Po zavolání metody DrawDetails() jsou ve smyčce přes všechny vzorky na pozici pixelů označených danou barvou generovány funkcí CreateRay() paprsky. Tyto paprsky jsou předány funkci Trace(). Funkce implementuje základní algoritmus metody sledování paprsku a po jejím skončení je v členské proměnné color třídy Ray barva daného vzorku. Po aritmetickém zprůměrování vzorků je výsledná barva zapsána do frame-bufferu. Tímto způsobem jsou přepsány všechny pixely ve frame-bufferu označené danou barvou. Po skončení je obraz scény z frame-bufferu zkopírován zpět do color-bufferu grafické karty.

4.9 Výsledky

Systém byl testován na scéně složené z jedenácti objektů. Tato scéna představuje místnost s několika typickými kandidáty na zachování globálních vztahů. Podlaha je vyrobena z keramických dlaždic. Bude tedy zrcadlit zbytek scény. Odráží převážnou část difuzní složky světla (asi 80%) a její zrcadlová složka je nastavena na 15 procent. Dlaždice jsou imitovány procedurální texturou s šachovnicovým motivem. Levá a zadní stěna a strop místnosti jsou plně difuzní, jejich spekulární složka se blíží nule. Barvu stěn udává opět procedurální textura. Tentokrát však jednobarevná. Na zadní stěně visí zrcadlo, které odráží scénu před ním. Jeho materiálovými vlastnostmi je dána vysoká odrazivost jak difuzní tak spekulární (80%). Zastřený obraz vzniká aplikací světle šedé textury. Posledním modelem ve scéně je zlatá koule. Jejími vlastnostmi jsou především 30-ti procentní spekulární odraz a ostrý odlesk. Tato scéna byla vykreslena dvěma způsoby. Poprvé s větší plochou počítaných detailů. Všechny převážně difuzní materiály (stěny) byly vykresleny pomocí OpenGL, objekty s vyšší odrazivou složkou byly dokresleny RT jednotkou. Maska této scény je vidět na obrázku P.2 v příloze 5. Podruhé byla stejná scéna vykreslena s menší plochou počítaných detailů. V této scéně byla i podlaha vykreslena pomocí OpenGL. Její maska je na obrázku P.4.

Obě výsledné scény (obrázek P.3 a obrázek P.5) mají rozlišení 640 x 480 pixelů, detaily byly vykreslovány s pomocí supersamplingu. Barva každého pixelu je průměrem čtyř vzorků rozmístěných podle mřížky generované před započetím vykreslování metodou jittering. Na mřížku byl aplikován tent filtr pro lepší distribuci vzorků. Časy nutné pro vykreslení obou scén jsou uvedeny v tabulce 4.1.

Číslo	**Scéna 1 [s]**		**Scéna 2 [s]**	
měření	**Čas RT jednotky**	**Čas vykreslení**	**Čas RT jednotky**	**Čas vykreslení**
1.	2,047	2,093	0,890	0,921
2.	2,047	2,063	0,875	0,890
3.	2,047	2,063	0,875	0,906
4.	2,047	2,063	0,875	0,907
5.	2,046	2,062	0,875	0,890
Průměr	2,046	2,068	0,878	0,902

Tabulka 4.1: Tabulka uvádí časy vykreslení celé scény (Čas vykreslení) a čas spotřebovaný samotnou RT jednotkou (Čas RT jednotky) pro pět měření.

Z tabulky lze snadno vypočítat režii na přenos rastrových dat z paměti grafického adaptéru do operační paměti a zpět. Pro scénu 1 je to průměrně 22 ms, pro scénu 2 průměrně 24,8 ms. V této režii je zahrnuto i generování masky, avšak tento čas je díky hardwarové akceleraci zanedbatelný.

Ztráta globálních vztahů ve scéně vykreslené pomocí OpenGL je jasně patrná z chybějícího stínu koule na zadní stěně scény 1, ačkoli podlaha jej odráží. Tento stín by musel být dokreslen standardními metodami. Vlastnost metody sledování paprsku, lesklé plochy nepřenáší světelnou energii, je v této scéně zřejmá z odrazu koule v zrcadle. Tento odraz ukazuje část koule, která je ve stínu i když by se do těchto míst světlo dostalo odrazem od zrcadla a relativně světlé stěny.

Všechny numerické výsledky v této kapitole byly získány na testovacím počítači osazeném procesorem Intel Celeron D331 (2.66 GHz, 256 kB L2), operační pamětí 1 GB DDR2 dual-link na FSB 533 MHz a grafickou kartou ATI Radeon X1300 na PCIe x16 s 256 MB GDDR. Operační systém Microsoft Windows XP Professional SP2. Překladač gcc verze 3.4.5, při použití optimalizace O2. Verze OpenGL neznámá. GLUT verze 3.7.6. Hodnoty byly naměřeny s využitím standardních součástí hlavičkového souboru ctime algoritmem 4.3.

```
clock_t t1 = clock()
//měřený úsek programu
clock_t t2 = clock()
trvání v sekundách = (t2-t1)/CLOCKS_PER_SEC
```

Algoritmus 4.3: Měření doby trvání části programu.

5 Závěr

Cílem mé práce bylo navrhnout a implementovat programový systém vkládání detailů zpracovávaných v software do výstupů OpenGL. Tento cíl byl splněn.

Poznatky z prvního bodu zadání, prostudovat literaturu na téma realistické zobrazování scény a literaturu zabývající se standardní knihovnou OpenGL, jsou diskutovány ve druhé kapitole. Druhý bod přímo vychází z bodu prvního a jeho výsledky jsou opět probírány napříč druhou kapitolou. Splnění třetího bodu zadání je výsledkem úvah uvedených ve třetí kapitole. Čtvrtý bod zadání, implementace programového systému vkládání detailů zpracovávaných v software do výstupů OpenGL je diskutován v kapitole čtvrté.

Mým prvotním záměrem bylo zkounstruovat jednotku pracující na principu metody sledování paprsku, neboť mě metody pro realistické zobrazování zaujaly. Tento cíl se později rozšířil o vykreslování v reálném čase za cenu snížení kvality výsledku. První výsledky během implementace mě však zaskočili. Vzhledem k metodě propojení se standardní grafickou knihovnou OpenGL jsem očekával, že přenos dat po sběrnici bude úzkým hrdlem láhve, které znehodnotí snažení. Avšak na základě výsledků uvedených v kapitole 4.9 je vidět, že dnešní technologie jsou v tomto směru značně výkonné. Zato výsledky RT jednotky jsou neslavné. Ztráta je pravděpodobně způsobena neoptimální implementací výpočtů v oblasti osvětlování. Praktickým cílem budoucího vývoje tedy bude optimalizace výpočtu osvětlovacího modelu a výpočtu odražených a lomených paprsků (lom není v současnosti uvažován). Vhodným směrem optimalizací je podle mého názoru další snižování počtu paprsků. Také implementaci stromu obálek bych rád modifikoval do zmíněné podoby stromu implementovaného v poli. Tyto úpravy by měli přispět ke zvýšení výkonu. Už z principu metody sledování paprsku vyplývá, že tento výkon nebude srovnatelný s výkonem standardních knihoven. Pro tento cíl by bylo vhodné hledat alternativy v oblasti programování hardware (rychlejší vektorové operace), paralelizace, a hardwarové řešení.

Naučil jsem se mnoho o počítačové grafice a realistickém zobrazování. A výsledky uvedené v kapitole 4.9 potvrzují že tato metoda vkládání detailů může pracovat v reálném čase za předpokladu, že plocha vkládaných detailů nebude příliš velká. Umožňuje to právě hodnota režie pro přesun dat po sběrnici. Pokračování tohoto projektu vidím tedy v hledání možných optimalizací. Pro možnost testovat větší scény by bylo vhodné implementovat modul pro načítání scénických souborů programů typu Art of Illusion či POV-Ray. S tím souvisí i transformace objektů. Přidat lom světla a přenos světelné energie, ačkoliv tyto dvě úpravy výkon spíše sníží. A v neposlední řadě také vytvořit pravidla pro určení, které objekty budou vykresleny pomocí OpenGL a které RT jednotkou. Tato pravidla by měla být podložena lidským vnímáním a jeho možnostmi.

Literatura

[1] Žára, J.: *Moderní počítačová grafika*. Brno, Computer Press 2004.

[2] Goral, C. M., Torrance, K. E., Greenberg, D. P., Battaile, B.: *Computer Graphics*. Svazek 18, 1984, číslo 3, strana 213.

[3] Shirley, P., Morley, K. R.: *Realistic ray tracing*. Massachusetts, Natick 2003.

[4] Jensen, H. W.: *Global Illumination using Photon Maps*. In Rendering Techniques '96. Svazek 10, Springer-Verlag 1996, strana 21.

[5] Classner, A. S.: *Ray Tracing*. 2. vydání, San Francisco, Morgan Kaufmann 2002.

[6] Shreiner, D., Woo, M., Neider, J., Davis, T.: *OpenGL: průvodce programátora*. Brno, Computer Press 2006.

[7] Humphreys, G.: *Ray Tracing Acceleration Techniques*. [přednášky k předmětu syntéza obrazu] University of Virginia, Charlottesville, Virginia.

[8] Smits, B.: *Overview of Bounding Volume Hierarchies*. 19. února 1999. Dokument dostupný na URL http://www.cs.utah.edu/~bes/papers/fastRT/paper-node8.html (květen 2008).

[9] Christen, M.: *Ray Tracing on GPU*. [diplomová práce] University of Applied Sciences, Basel-Landschaft, Switzerland.

[10] Lengyel, E.: *Mathematics for 3D: game programming and computer graphics*. Massachusetts, Charles River Media 2004.

[11] Möller, T., Haines, E.: *Real-time rendering*. Massachusetts, Peters 2002.

[12] Dutré, P.: *Global Illumination Compendium*. 29. září 2003. Dokument dostupný na URL http://www.cs.kuleuven.ac.be/~phil/GI/TotalCompendium.pdf (květen 2008).

Seznam příloh

Výpočet průsečíku přímky a koule

Ve 3D může být přímka zadána parametrickou rovnicí, kde parametr je reálné číslo udávající pozici na přímce. Ve vektorovém zápisu vypadá rovnice následovně:

$$\vec{P}(t)=\vec{O}+t\cdot\vec{D} \quad . \tag{P.1}$$

Hledáme-li průsečík s koulí se středem $C = (c_x, c_y, c_z)$ a poloměrem r bude reprezentována rovnicí:

$$(x-c_x)^2+(y-c_y)^2+(z-c_z)^2-r^2=0 \quad . \tag{P.2}$$

Přepíšeme rovnici ve vektorovém zápisu:

$$(\vec{P}-\vec{C})\cdot(\vec{P}-\vec{C})-r^2=0 \quad . \tag{P.3}$$

Každý bod P, který vyhoví této rovnici, leží na povrchu koule. Dosazením vztahu P.1 do této rovnice, získáme řešením parametr t, který reprezentuje průsečík:

$$(\vec{O}+t\cdot\vec{D}-\vec{C})\cdot(\vec{O}+t\cdot\vec{D}-\vec{C})-r^2=0 \quad . \tag{P.4}$$

Roznásobením získáme kvadratickou rovnici s jedinou neznámou t:

$$(\vec{D}\cdot\vec{D})\cdot t^2+2\cdot\vec{D}\cdot(\vec{O}-\vec{C})\cdot t+(\vec{O}-\vec{C})\cdot(\vec{O}-\vec{C})-r^2=0 \quad . \tag{P.5}$$

Řešením kvadratické rovnice je vzorec:

$$t=\frac{-b\pm\sqrt{b^2-4\cdot a\cdot c}}{2\cdot a} \quad . \tag{P.6}$$

Diskriminant nám říká kolik reálných kořenů má řešení. Pokud je diskriminant záporný je odmocnina imaginární a neexistuje žádný průsečík mezi přímkou a koulí. Pokud je kladný existují dvě řešení jedno v místě, kde přímka do koule vstupuje a druhé když ji opouští. Pokud je diskriminant nulový je přímka tečnou koule. Dosazením získáme rovnici:

$$t=\frac{-\vec{D}\cdot(\vec{O}-\vec{C})\pm\sqrt{(\vec{D}\cdot(\vec{O}-\vec{C}))^2-(\vec{D}\cdot\vec{D})\cdot((\vec{O}-\vec{C})\cdot(\vec{O}-\vec{C})-r^2)}}{(\vec{D}\cdot\vec{D})} \quad . \tag{P.7}$$

V rámci optimalizace je nutné nejprve vyhodnotit diskriminant a postupně určovat zda má význam pokračovat ve výpočtu. Samozřejmostí je všechny výpočty provádět jen jednou, proto opakující se části výrazu by měly být vyhodnoceny a uloženy pro pozdější potřebu.

Normálový vektor je pak rozdílem průsečíku P a středu koule C. Jeho normalizaci provedeme podělením tohoto vektoru poloměrem koule r:

$$\vec{N}=\frac{(\vec{P}-\vec{C})}{r} \quad . \tag{P.8}$$

Výpočet průsečíku přímky a trojúhelníku

Trojúhelník je definován třemi body *A*, *B* a *C*. Pokud tyto body neleží na jedné ose, definují plochu. Běžným způsobem popisu této plochy jsou barycentrické souřadnice.

$$\vec{P}(\alpha,\beta,\gamma)=\alpha\cdot\vec{A}+\beta\cdot\vec{B}+\gamma\cdot\vec{C} \quad , \tag{P.9}$$

kde

$$\alpha+\beta+\gamma=1 \quad . \tag{P.10}$$

Transformací podmínky P.10 do tvaru $\alpha=1-\beta-\gamma$ a její aplikací na vztah P.9 získáme rovnici:

$$\vec{P}(\beta,\gamma)=\vec{A}+\beta\cdot(\vec{B}-\vec{A})+\gamma\cdot(\vec{C}-\vec{A}) \quad . \tag{P.11}$$

Nyní body náleží trojúhleníku pouze když splní následující podmínky:

$$\beta+\gamma<1 \quad , \quad 0<\beta \quad \text{a} \quad 0<\gamma \quad . \tag{P.12}$$

Dosazením rovnice přímky do vztahu P.11 získáme vektorový zápis rovnice pro výpočet průsečíku:

$$\vec{O}+t\cdot\vec{D}=\vec{A}+\beta\cdot(\vec{B}-\vec{A})+\gamma\cdot(\vec{C}-\vec{A}) \quad . \tag{P.13}$$

Tuto rovnici můžeme rozepsat na soustavu tří rovnic o třech neznámých β, γ a t.

$$\begin{aligned} o_x+t\cdot d_x&=a_x+\beta\cdot(b_x-a_x)+\gamma\cdot(c_x-a_x) \quad , \\ o_y+t\cdot d_y&=a_y+\beta\cdot(b_y-a_y)+\gamma\cdot(c_y-a_y) \quad , \\ o_z+t\cdot d_z&=a_z+\beta\cdot(b_z-a_z)+\gamma\cdot(c_z-a_z) \quad . \end{aligned} \tag{P.14}$$

Tyto rovnice mohou být přepsány do podoby matice:

$$\begin{bmatrix} a_x-b_x & a_x-c_x & d_x \\ a_y-b_y & a_y-c_y & d_y \\ a_z-b_z & a_z-c_z & d_z \end{bmatrix}\cdot\begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix}=\begin{bmatrix} a_x-o_x \\ a_y-o_y \\ a_z-o_z \end{bmatrix} \quad . \tag{P.15}$$

Vyřešením této soustavy rovnic o třech neznámých pomocí Cramerova pravidla získáme neznámé β, γ a t. S jejich pomocí získáme z rovnice P.10 neznámou α. Nyní máme všechny potřebné parametry pro výpočet průsečíku, normály i texturovacích souřadnic. Stejně jako v případě koule, kontrolujeme podmínky P.12 a v případě, že nevyhoví výpočet okamžitě ukončíme.

Normálu vypočítáme s pomocí získaných barycentrických souřadnic ze vztahu:

$$\vec{N}=\vec{N}_A\cdot\alpha+\vec{N}_B\cdot\beta+\vec{N}_C\cdot\gamma \quad , \tag{P.16}$$

kde N_A, N_B a N_C jsou normály v bodech *A*, *B* a *C*.

Výpočet průsečíku přímky a obálky

Definujme osově zarovnanou obálku ve 3D jako objem definovaný šesti přímkami. Tento objem je možné popsat intervalem:

$$(x, y, z) \in [x_0, x_1] \times [y_0, y_1] \times [z_0, z_1] \quad . \tag{P.17}$$

Test průsečíku může být rozdělen na jednotlivé intervaly. Uveďme pouze výpočty pro souřadnici x, protože ostatní souřadnice jsou analogické. Prvním krokem je výpočet parametru t_{MIN}, kde x nabývá nejnišší hodnoty:

$$t_{MIN} = \frac{x_0 - o_x}{d_x} \quad . \tag{P.18}$$

Analogicky vypočítáme parametr t_{MAX}. Nyní zbývá ověřit, zda se intervaly $[t_{MIN};t_{MAX}]$ všech souřadnic překrývají.

Uvedené řešení má však jeden nedostatek. Řešení je omezeno pouze na kladné d_x. Tato závyslost je odstraněna v algoritmu P.1.

```
if (ray.direction.x >= 0) {
  tMin = (x0-ray.origin.x)/ray.direction.x
  tMax = (x1-ray.origin.x)/ray.direction.x
}
else {
  tMin = (x1-ray.origin.x)/ray.direction.x
  tMax = (x0-ray.origin.x)/ray.direction.x
}
```

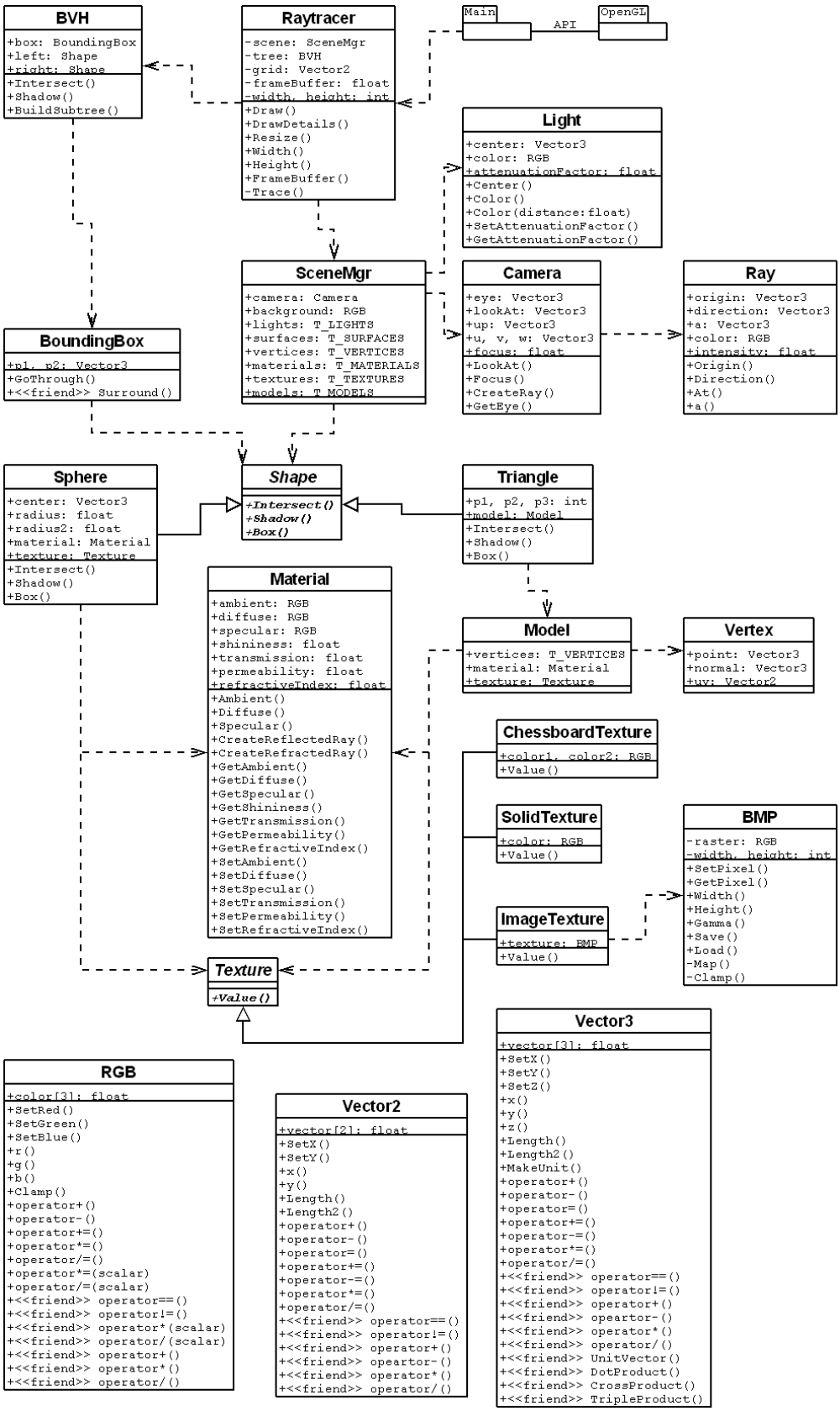
Algoritmus P.1: Úprava řešení pro všechny směry paprsku.

Stále však algoritmus není dokonalý. Pojďme se podívat co by se stalo, pokud by směr paprsku byl souhlasný s některou z os. Je evidentní, že by docházelo k chybám při porovnávání intervalů, neboť norma IEEE 754 stanovuje pro reálné kladné číslo a následující pravidla:

$$\begin{aligned} +a/0 &= +\infty \\ -a/0 &= -\infty \end{aligned} \quad , \qquad \begin{aligned} +a/-0 &= -\infty \\ -a/-0 &= +\infty \end{aligned} \quad .$$

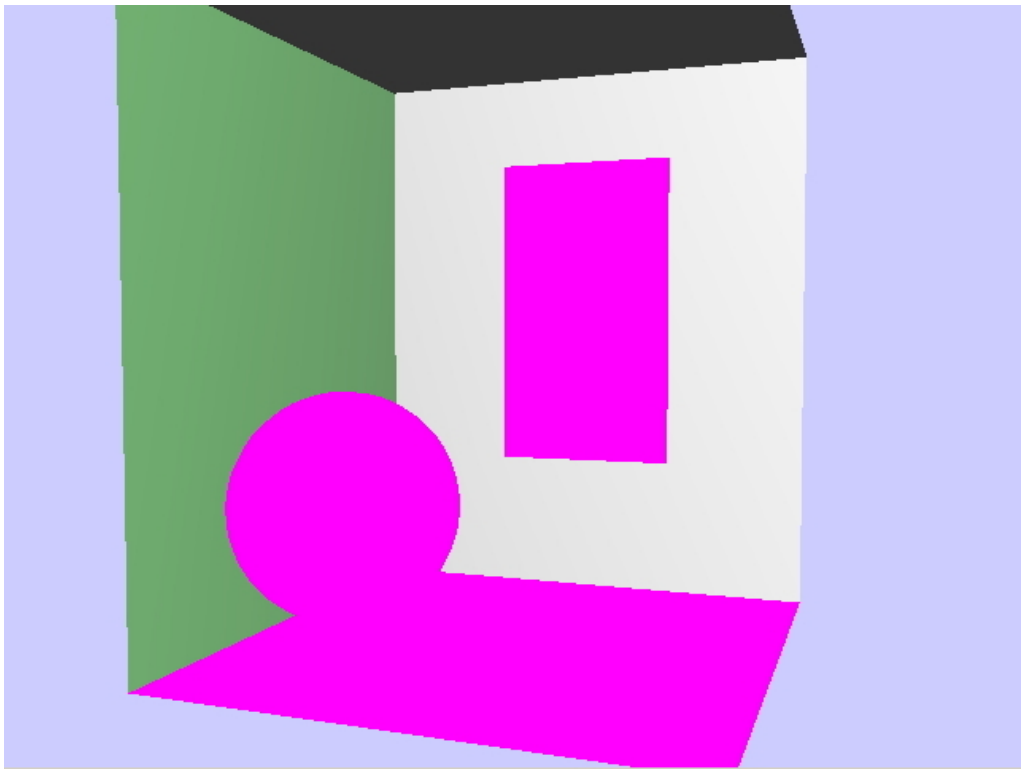
Řešením této situace je předpočítat si hodnotu $a_x = 1/d_x$ a tuto hodnotu kontrolovat v podmínce algoritmu P.1 místo d_x. K dělení bude sice stále docházet, ale kontrola výsledného znaménka je již zahrnuta při rozhodování o směru paprsku. Navíc na mnoha architekturách budou dělení a dvě násobení rychlejší, než dvě dělení. Uložením této informace ve struktuře paprsku docílíme dokonce jen jednoho dělení na výpočet průsečíků se všemi obálkami.

Schéma hierarchie tříd

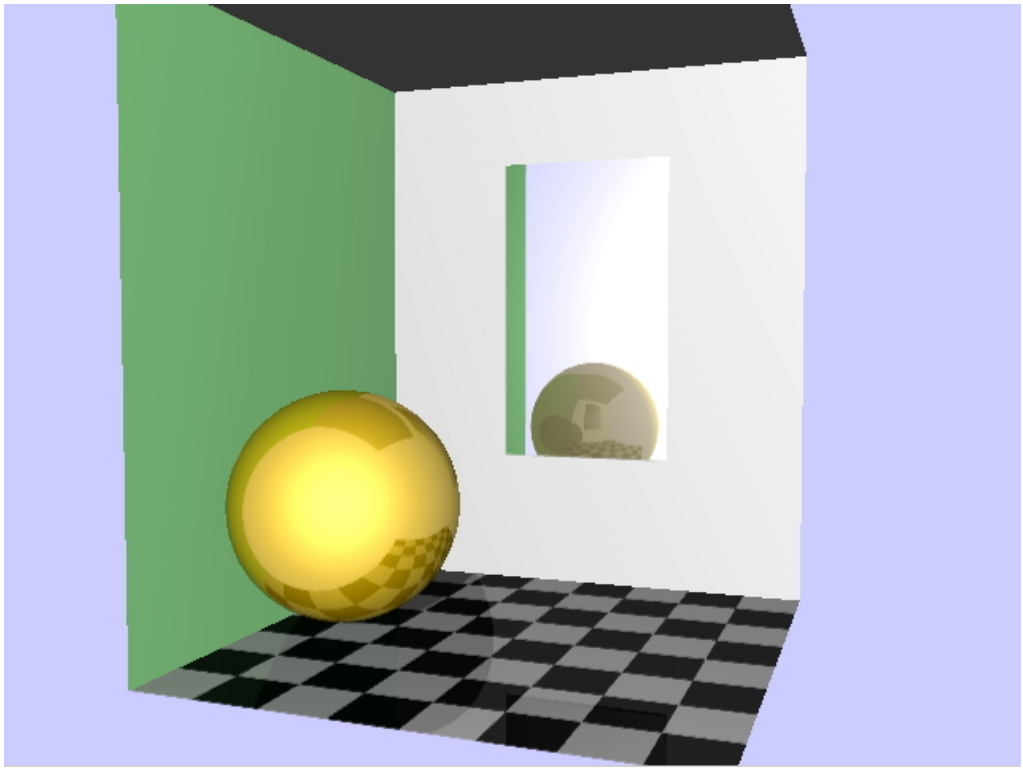


Obrázek P.1: Schéma hierarchie tříd.

Obrázky – scéna 1

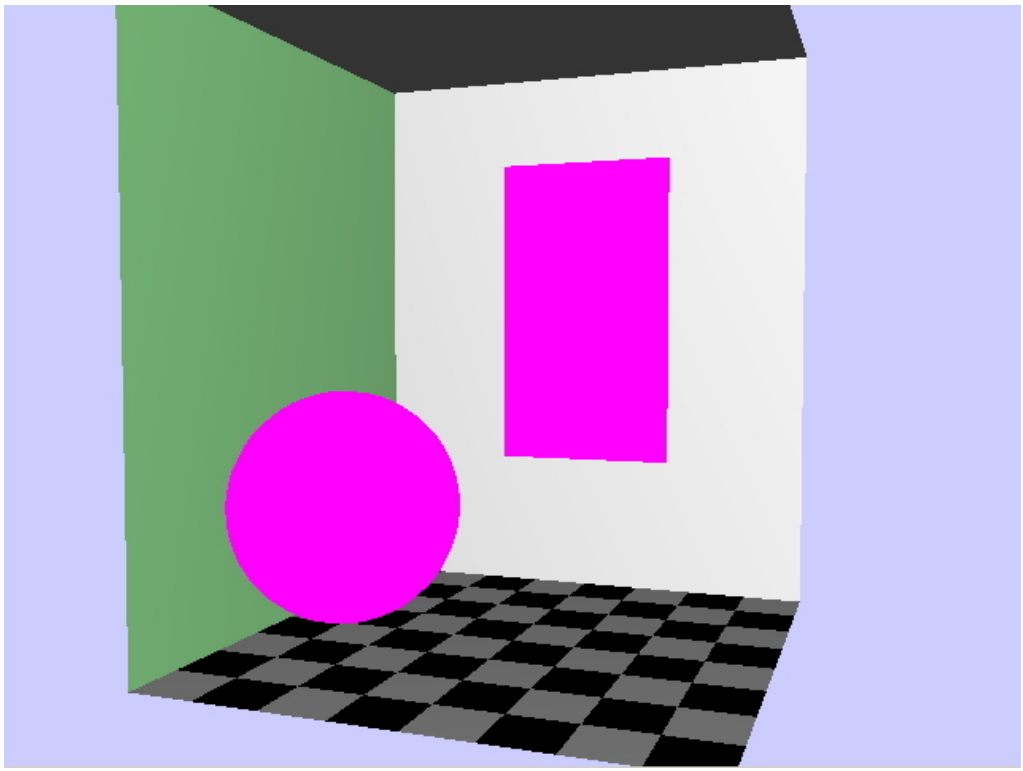


Obrázek P.2: Maska scény s větší plochou detailů.

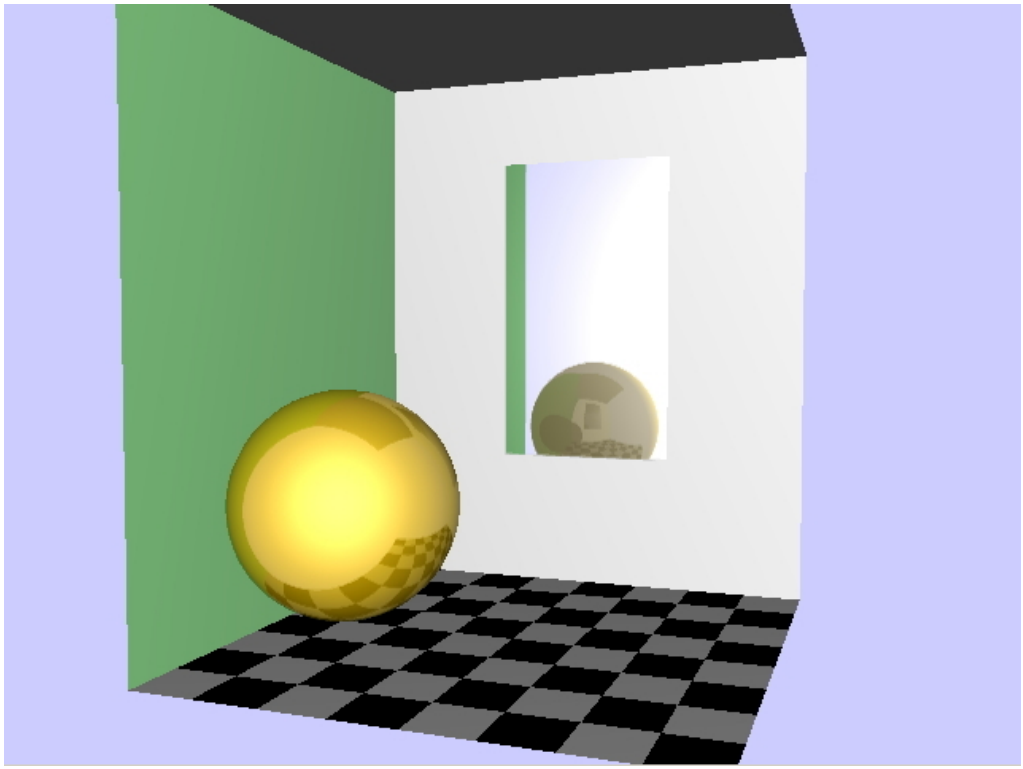


Obrázek P.3: Výsledná scéna s vyššími detaily.

Obrázky – scéna 2



Obrázek P.4: Maska scény s menší plochou detailů.



Obrázek P.5: Výsledná scéna s nižšími detaily.