Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky

Diplomová práce

Výpočet efektů globálního osvětlení v rasterizačním řetězci

Bc. Tomáš Altman

Vedoucí práce: Ing. Jiří Bittner, Ph.D.

13. května 2014

Poděkování

Děkuji svému zadavateli Ing. Jiří Bittnerovi, Ph. D. za vstřícnost a trpělivost při vývoji aplikace. Děkuji rovněž Dr. Dipl.-Inf. Oliverovi Mattauschovi za poskytnutou implementaci s množstvím komentářů. Závěrem děkuji také své manželce a rodičům za jejich vytrvalou podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 13. května 2014 .....

Abstrakt

Tato práce se zabývá popisem a implementací nové metody, pomocí které je možno vypočítat efekty globálního osvětlení v rasterizačním řetězci. K urychlení výpočtů osvětlení jsou využívány dotazy zastínění a časová koherenci po sobě jdoucích snímků. Implementace nové metody je testována na třech rozsáhlejších scénách a porovnána s dostupnou implementací GPU ray tracingu [AL09].

Abstract

This paper describes and implements a new method, which computes global effects through rasterization pipeline. To acceleration are used Occlusion Queries and exploited temporal coherence of following frames. The implementation is tested on three expansive scenes and compared with the available implementation GPU Ray tracing [AL09].

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V oblasti počítačové grafiky jsou 3D data vykreslována dvěma způsoby, rasterizací a metodami založenými na vrhání paprsků. Rasterizace představuje rychlou, hardwarově podporovanou metodu, která umožňuje přímé osvětlení scény. Chceme-li však umožnit přenášení světla mezi objekty navzájem, musíme se přesunout k náročnějším metodám založeným na vrhání paprsků (viz obr. 1.1). Mezi ně patří Ray Tracing, Path Tracing, Photon Mapping, Ambient Occlusion atd. Výpočetní náročnost rapidně vzrůstá s počtem vržených paprsků a tím se vykreslování stává trhaným.

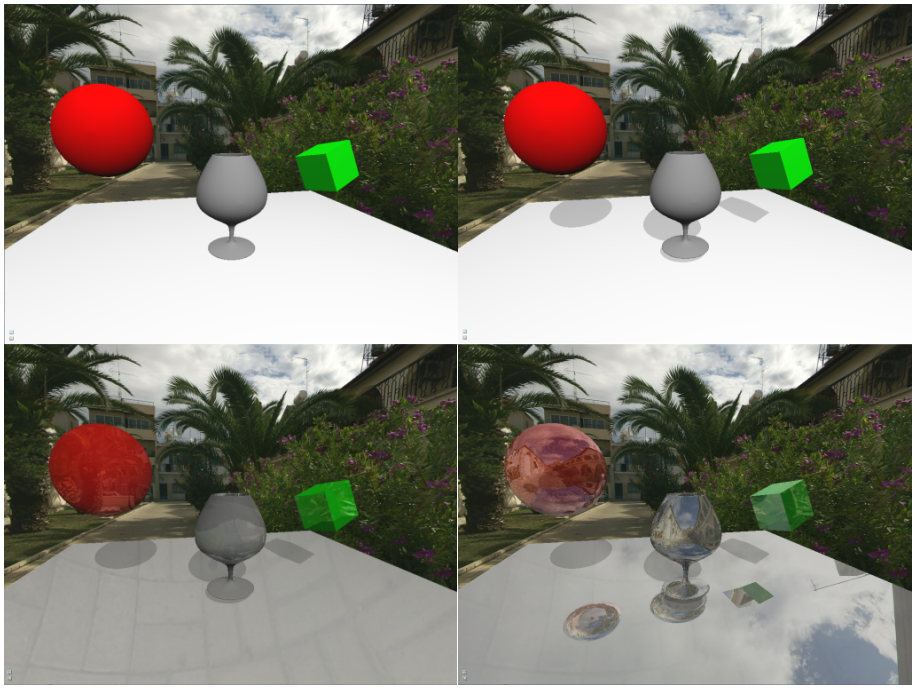
Tato práce vychází z předchozí práce Dr. Dipl.-Inf. Mattausche a Ing. Bittnera, Ph.D. Popisuje hybridní metodu propojující rasterizaci s metodou sledování paprsků. K urychlení výpočtů osvětlení využívá hierarchii párů (uzel BVH, svazek paprsků), kterou pomocí dotazů zastínění a dalších optimalizací průběžně upravuje.

1.1 Cíl práce

Cílem mé práce je poskytnout srozumitelný pohled do nové metody výpočtu efektů globálního osvětlení v rasterizačním řetězci, která k urychlení výpočtů osvětlení využívá dotazy zastínění. Tato metoda bude demonstrována na třech rozsáhlejších scénách a porovnána s existující metodou sledování paprsků na GPU [AL09].

1.2 Struktura práce

V první části jsou vysvětleny teoretické základy, které se zabývají přímým a nepřímým osvětlením, základními způsoby stínování a rozdíly mezi rasterizací a sledováním paprsků. Následuje popis metod, které jsou při výpočtu osvětlení závislé na viditelnosti objektů. Mezi ně patří stínové mapy, stínové objemy, sledování paprsků a s nimi úzce související speciální typ – vrhání paprsků po-



Obrázek 1.1: Porovnání osvětlení scény. Pouze přímé osvětlení (vlevo nahoře), se stíny (vpravo nahoře), s průhlednými objekty (vlevo dole) a s odlesky (vpravo dole).

mocí hierarchie paprsků. Hlavní část práce se zabývá metodou vrhání paprsků pomocí dotazů zastínění. Metoda je podrobně popsána, implementována a testována. Výsledky testování jsou porovnány s existující implementací na GPU [AL09] a závěrem jsou shrnuty přínosy nové metody a nastíněny možnosti, jak ji lze dále rozvíjet.

Kapitola 2

Teoretické základy

V této kapitole jsou popsány druhy osvětlení a stínování. Kapitolu uzavírá porovnání rasterizace se sledováním paprsků.

2.1 Výpočet osvětlení

Výchozím bodem při zobrazování scény je simulace světla, které putuje scénou a interaguje s povrchy objektů. V reálném světě světlo z velké části nedopadá do našeho oka přímo ze světelného zdroje, nýbrž z povrchů objektů či z opticky aktivního prostředí (mlha, dým...). Simulace reálného transportu světla ve scéně je tak komplexní a obtížnou úlohou.

V počítačové grafice dělíme osvětlení na přímé a nepřímé; oba typy jsou dále popsány.

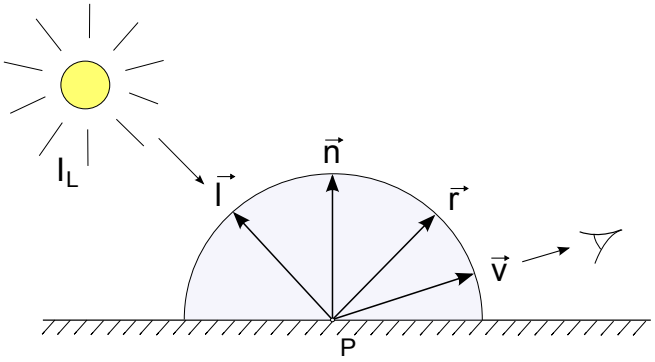
2.1.1 Přímé osvětlení

Přímé osvětlení (*Direct Illumination*) zahrnuje do výpočtů pouze světlo přicházející ze světelných zdrojů. Další transport světla mezi povrchy objektů není brán v úvahu. Jedním z velmi známých a široce používaných modelů přímého osvětlení je Phongův osvětlovací model.

Phongův osvětlovací model

Bui-Tuong Phong v roce 1975 navrhl empiricky odvozený osvětlovací model, který umožňuje výpočet odrazu světla na povrchu materiálu [Pho75]. Odraz je určen směrem dopadajícího vektoru $\vec{l}$, vektorem $\vec{v}$ směřujícím od bodu P k pozorovateli, normálovým vektorem v místě dopadu $\vec{n}$ a zrcadlově odraženým vektorem $\vec{r}$ (viz obr. 2.1).

Ve Phongově osvětlovacím modelu se rozlišují tři typy odrazu světla: ambientní, difúzní a zrcadlový (spekulární) (podle [ŽBSF04]). Složením těchto typů vznikne výsledné osvětlení.



Obrázek 2.1: Phongův osvětlovací model pro místo dopadu P. Podle [ŽBSF04].

Ambientní odraz světla simuluje ze všech směrů přicházející světlo. Jeho přítomnost zabraňuje, aby stěny odvrácené od světla byly zcela černé. Lze jej vyjádřit jednoduchým vztahem:

$$I_a = I_A * r_A, \tag{2.1}$$

kde I_A značí množství okolního světla a r_A je koeficient určující schopnost povrchu odrážet okolní světlo.

Difúzní odraz světla je závislý na směru přicházejícího světla. Čím je dopad světla bližší normále, tím je difúzní příspěvek větší. Platí to pouze pro $\vec{l} \cdot \vec{n} > 0$, protože v opačném případě je povrch odvrácen od světla a příspěvek je nulový. Difúzní odraz lze vyjádřit pomocí vztahu:

$$I_d = I_L * r_D * (\vec{l} \cdot \vec{n}), \tag{2.2}$$

kde I_L reprezentuje barevné složení dopadajícího paprsku a r_D koeficient difúzního odrazu, který ovlivňuje zastoupení difúzní složky v celkově odraženém světle.

Zrcadlový odraz světla je vyjádřen jako:

$$I_s = I_L * r_S * (\vec{v} \cdot \vec{r})^h, \tag{2.3}$$

kde I_L stejně jak u difúzního odrazu značí barvu dopadajícího paprsku. r_S vyjadřuje míru zastoupení zrcadlové složky v celkovém odrazu. h má vliv na ostrost odlesku. Vektor $\vec{v}$ představuje jednotkový vektor pohledu. Jednotkový zrcadlový vektor $\vec{r}$ je podle normály $\vec{n}$ symetrický k vektoru $\vec{l}$ (viz obr. 2.1). Vyjadřuje směr ideálního zrcadlového odrazu.

Výsledný celkový odraz je dán součtem ambientního, difúzního a zrcadlového odrazu:

$$I_V = I_a + I_d + I_s. \tag{2.4}$$

2.1.2 Nepřímé osvětlení

Nepřímé osvětlení (*Indirect Illumination*) sleduje přenášení světla mezi objekty navzájem. Umožňuje vykreslit pokročilé efekty, mezi něž patří stíny, odlesky, lom světla, kaustiky, Color Bleeding aj.

K výpočtu efektů nepřímého osvětlení slouží metody: Ray Tracing, Path Tracing, Ambient Occlusion, Photon Mapping atd.

Globální osvětlení (*Global Illumination*) je sloučením přímého a nepřímého osvětlení.

2.2 Stínování

Stínování je způsob, jak rozprostřít barvu z několika styčných bodů na celou plochu. Existují tři nejběžněji používané typy stínování: konstantní stínování, Gouraudovo stínování a Phongovo stínování.

Konstantní stínování je nejjednodušším a nejrychlejším typem stínování. Pracuje s předpokladem, že každá plocha má jednu normálu, podle níž se dopočítá barva. Tato barva je při rasterizaci přiřazena všem pixelům plochy. Konstantní stínování se používá pro zobrazení rovinných ploch.

Gouraudovo stínování umožňuje plynulé stínování křivých povrchů. Barva pixelu je vypočítána z barev vrcholů plošky pomocí bilineární interpolace. Barva vrcholů je získána vyhodnocením osvětlovacího modelu. Gouraudovo stínování neposkytuje zcela věrný obraz, protože neumožňuje lokální zvýšení jasu.

Phongovo stínování nezískává barvu interpolací barev vrcholů jak předešlé stínování, ale je založeno na interpolaci normálových vektorů. Při rasterizace plochy jsou současné interpolovány normály, přičemž vyhodnocením osvětlovacího modelu pro každý pixel je vypočítána barva. Phongovo stínování umožňuje správné zvýšení jasu uprostřed plochy.

Odložené stínování

V rozsáhlých scénách může při rasterizaci vzniknout mnoho fragmentů, které se neobjeví na výsledném plátně. Jsou zakryty fragmenty bližšími k pozorovateli. Stínování zakrytých fragmentů je nevýhodné a používáme-li výpočetně náročný způsob stínování, způsobuje to zpomalení celého vykreslování. Proto v roce 1988 představil Deering a kol. ([DWS+88]) metodu odloženého stínování (*Deffered Shading*), jejímž záměrem je oddělit geometrické výpočty od výpočtů osvětlení.

Deffered Shading se skládá ze dvou vykreslovacích průchodů. V prvním průchodu je scéna rasterizována a do připravených textur, souhrnně nazvaných *G-buffer*, jsou uloženy informace o geometrii a materiálech objektů viditelných z pozice kamery. Jedna buňka textury odpovídá jednomu pixelu obrazu. V druhém průchodě se rasterizuje obdélník přes celý obraz a z dat G-Bufferu je dopočteno osvětlení pro každý pixel.

Nevýhodou odloženého stínování je, že G-Buffer může obsahovat informace pouze o neprůhledné geometrii. V případě průhledné geometrie by musela jedna buňka textury obsahovat informace o průhledné i neprůhledné geometrii, která se nachází pod průhlednou geometrií. To ovšem z podstaty G-Bufferu nelze. Řešením problému je vykreslení průhledné geometrie až v dodatečném průchodu.

2.3 Rasterizace a sledování paprsků

Jak bylo zmíněno již v úvodu, existují dva základní způsoby vykreslení 3D dat: rasterizací a metodami založenými na vrhání paprsků.

Rasterizace je výsledek vzorkování grafických objektů do mřížky obrazu. Je to rychlá metoda, která v současné době umožňuje vykreslování rozsáhlých scén v reálném čase. Rasterizace je hardwarově podporována a lze jí využívat v rozšířených grafických knihovnách, jako jsou OpenGL či Direct3D.

Rasterizace zpracovává každý objekt jednotlivě. Z hlediska implementace se jedná o smyčku přes všechny objekty scény, které jsou rasterizovány odděleně. Každý objekt tedy zastupuje při vykreslování jen sám sebe a počítá se pro něj lokální osvětlovací model.

Druhým způsobem vykreslení 3D dat jsou metody založené na vrhání paprsků; jednou z nejrozšířenějších je sledování paprsků (*Ray Tracing*). Je to metoda založená na vrhání paprsků pro každý pixel obrazu. Umožňuje vykreslení globálních efektů, mezi které patří stíny, odlesky, zalomení světla aj.

Naivní řešení sledování paprsků – testování průniku jednotlivých paprsků se všemi objekty – je výpočetně velmi náročné (kvadratická asymptotická složitost). Proto byl Ray Tracing předmětem rozsáhlého výzkumu; podrobně v 3.3.

Kapitola 3

Řešení viditelnosti pro výpočet osvětlení

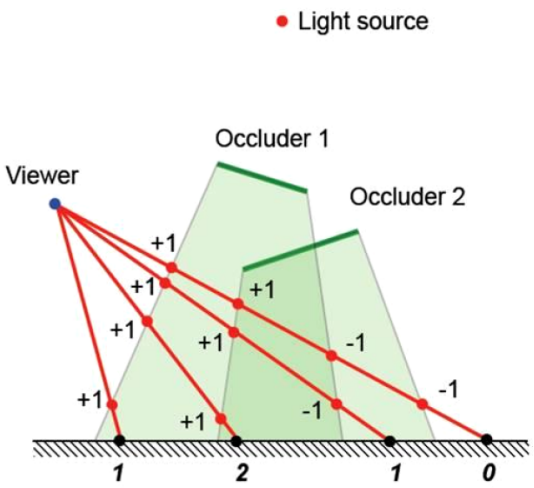
Cílem algoritmů, které řeší viditelnost, je nalézt ty objekty a jejich části, jež jsou viditelné z určitého místa. Umístíme-li do scény světlo, můžeme pomocí těchto algoritmů vypočítat stíny. Jedním z rozšířených algoritmů je metoda stínových objemů.

3.1 Metoda stínových objemů

Metoda stínových objemů je jedna z velmi používaných technik k zobrazení stínů. Poprvé ji představil Frank Crow v roce 1977 [Cro77] pod názvem *Z-pass*. Myšlenka je jednoduchá: siluety objektů protáhneme ve směru světla a poté určujeme, zda bod leží alespoň v jednom ze vzniklých stínovém objemu. Pokud ano, bod je zastíněn. Vykreslování scény se dělí na tři průchody, přičemž ve všech průchodech se scéna rasterizuje z pohledu kamery.

V prvním průchodu se scéna klasicky rasterizuje a do připravených textur se uloží ambientní barva a hloubka. V druhém průchodu jsou vykreslovány stěny stínových objemů. Zápis do color bufferu i do Z-bufferu je zakázán, zatímco testování hloubky s výše zmíněným Z-bufferem je povoleno. Nejprve se vykreslí přední stěny stínových objemů a poté zadní stěny. Splní-li přední stěna hloubkový test, inkrementuje se hodnota ve stencil bufferu. Splní-li hloubkový test zadní stěna, hodnota se dekrementuje. V posledním třetím průchodu se scéna vykreslí i s difúzním a spekulárním osvětlením, ovšem pouze v místech, kde hodnota stencil bufferu je 0. Znázornění algoritmu je na obr. 3.1.

Z-pass technika dokáže správně projektovat stíny na jakémkoliv povrchu. K problémům dochází tehdy, když se pozorovatel nachází uvnitř stínového objemu nebo je objekt protínán přední rovinou pohledového jehlanu (*near plane*). K vyřešení těchto chyb vznikla technika Z-fail.



Obrázek 3.1: Metoda stínových objemů, Z-pass technika. Převzato z [ZB11].

Z-fail se oproti technice Z-pass liší v pořadí vykreslení stěn stínových objemů a v inkrementaci ve stencil bufferu. Nejprve se vykreslí zadní stěny stínových objemů, poté přední stěny. Nesplní-li zadní stěna hloubkový test, hodnota ve stencil bufferu se inkrementuje. Projde-li přední stěna hloubkovým testem, hodnota ve stencil bufferu se dekrementuje. Z-fail řeší většinu problémů s pozicí pozorovatele nebo umístěním near plane. K chybě může dojít pouze v případě, protíná-li zadní plocha pohledového jehlanu (*far plane*) objekt. Situace se vyřeší tím, že se far plane posune do nekonečna.

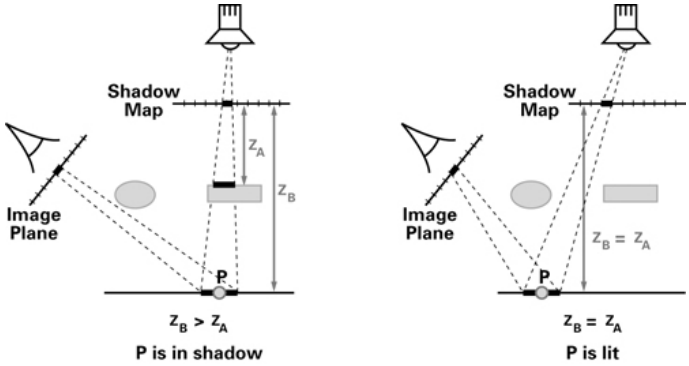
Vývoj metody stínových objemů doznal mnoha dalších úprav, které vedly ke snížení časových nároků. Přehledný popis celého vývoje je rozepsán v článku [ZB11].

3.2 Metoda stínových map

Jedna z nejběžněji používaných metod zobrazení stínů je metoda stínových map. V roce 1978 jí ve své publikaci [Wil78] představil L. Williams. Díky této metodě mohou být stíny vykresleny na libovolný povrch.

Algoritmus se skládá ze dvou kroků. V prvním kroku se scéna vykreslí z pohledu světla. Vzdálenost mezi světlem a nejbližším objektem se uloží do připojeného Z-bufferu, nazývaným stínová mapa. Během tohoto kroku probíhá testování hloubky, při němž dochází implicitně k odstranění skrytých povrchů. V druhém kroku se scéna vykreslí z pohledu kamery. Pozice jednotlivých fragmentů se převede do prostoru světla, kde se porovná hloubka fragmentů s odpovídající vzdáleností na stínové mapě. Je-li vzdálenost mezi světlem a objektem větší než vzdálenost uložená ve stínové mapě, je objekt zastíněn. V

opačném případě je osvětlen. Porovnávání hloubky je ilustrováno na obr. 3.2.



Obrázek 3.2: Metoda stínových map – porovnání hloubky. Převzato z [Web07].

Kvalita stínů je úzce spjata s rozlišením hloubkové mapy a numerickou přesností Z-bufferu. S malou přesností Z-bufferu dochází k aliasingu, který má podobu různě „zubatého“ nebo „pruhovaného“ stínu. Aliasing lze rozdělit do tří základních skupin. První skupinu představuje perspektivní aliasing, který je způsoben nedostatečným rozlišením stínové mapy. Druhou skupinou pak je projektivní aliasing, který vzniká, je-li stínící těleso paralelní se směry paprsků. A na závěr aliasing způsobený nedostatečnou přesností hloubky. Vývojem metod, které zamezují aliasingu, se zde nebudu dále zabývat. Přehledný popis je k dispozici zde: [LP09].

3.3 Metoda sledování paprsků

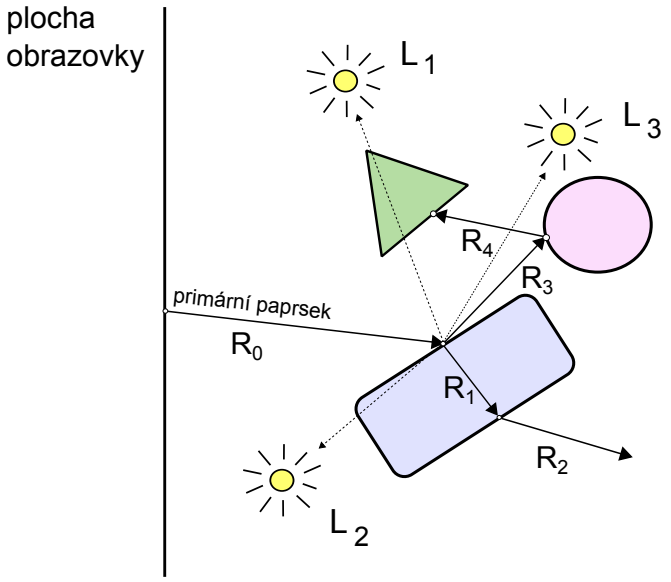
Poprvé představil metodu sledování paprsků Whitted a kol. v roce 1979 [FW79].

U metody sledování paprsků se rozlišují dva typy paprsků. Primární, které jsou vrhány z pohledu pozorovatele na scénu, a sekundární, které jsou již druhotně vytvářeny z míst, kde dochází k průniku paprsků. Schéma sledování paprsků je znázorněno na obr. 3.3.

Po nárazu primárního paprsku na povrch se mohou generovat další sekundární paprsky, které lze rozdělit do tří základních skupin: stínové paprsky, zrcadlové paprsky a paprsky lomu.

3.3.1 Stínové paprsky

Stínové paprsky mají jednoduchou úlohu: určit, zda-li je průnik zastíněný, nebo osvětlený. Vznikají v místě průniku primárního paprsku s plochou scény a směřují ke světlu. Protne-li stínový paprsek na cestě ke světlu nějaký objekt, je místo průniku zastíněno.



Obrázek 3.3: Schéma sledování primárního paprsku. Nakresleno podle [ŽBSF04].

Při implementaci je nutné posunout pozici stínového paprsku o malou vzdálenost tak, aby nedocházelo k nežádoucímu protnutí s výchozím objektem. Směr, jímž se pozice posouvá, se většinou určuje podle směru stínového paprsku nebo podle normály objektu.

3.3.2 Zrcadlové paprsky

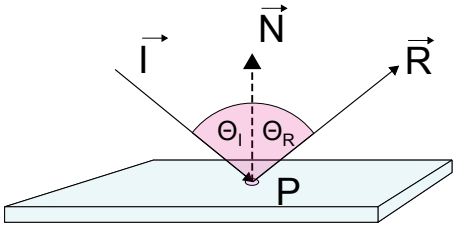
Zrcadlové paprsky přidávají k vykreslení scény jeden ze základních efektů globálního osvětlení: odlesky.

Podíváme-li se blíže na zrcadlové paprsky, zjistíme, že směr zrcadlového paprsku svírá s normálou lesklého povrchu stejný úhel jako normála s příchozím paprskem ($\Theta_I = \Theta_R$), viz obr. 3.4.

Z $\Theta_I = \Theta_R$ lze při uplatnění goniometrických pravidel dopočítat rovnici pro výpočet směru zrcadlového paprsku:

$$\vec{R_D} = \vec{I} - 2 * (\vec{I} \cdot \vec{N}) * \vec{N}, \tag{3.1}$$

kde $\vec{I}$ značí příchozí paprsek, P průnik příchozího paprsku s povrchem, $\vec{N}$ normálu povrchu. Všechny zmíněné vektory jsou normalizované. Přehledný popis celého odvození je k dispozici na [Web04].



Obrázek 3.4: Odraz zrcadlového paprsku. $\Theta_I = \Theta_R$.

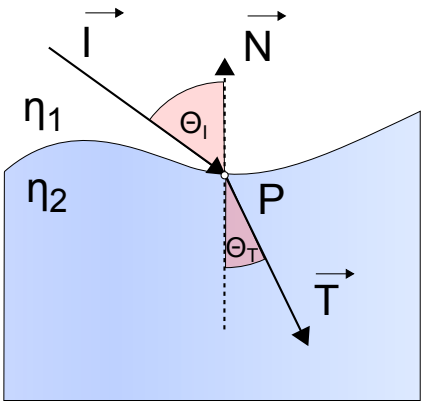
Sledujeme-li více odrazů zrcadlového paprsku, také příspěvek k výslednému odlesku klesne s počtem předchozích odrazů paprsku.

3.3.3 Paprsky lomu světla

Výpočet paprsků lomu světla vychází ze Snellova zákonu, jenž zní:

$$\eta_1 * sin\Theta_I = \eta_2 * \sin\Theta_T, \tag{3.2}$$

kde η_1 a η_2 představují indexy lomu světla dvou různých látkových prostředí a Θ_I a Θ_T úhly, které svírá vůči normále povrchu N příchozí, resp. odchozí paprsek, (viz obr. 3.5).



Obrázek 3.5: Snellův zákon: $\eta_1 * \sin\Theta_I = \eta_2 * \sin\Theta_T$.

Snellův zákon lze dále upravit na:

$$sin\Theta_T = \frac{\eta_2}{\eta_1} * sin\Theta_I. \tag{3.3}$$

V případě, že by nastalo $\sin\Theta_I > \frac{\eta_2}{\eta_1}$, muselo by též platit $\sin\Theta_T > 1$, což není matematicky možné. Proto je nutné přidat k Snellovu zákonu podmínku:

$$\sin\Theta_I <= \frac{\eta_2}{\eta_1}. \tag{3.4}$$

Tato podmínka je uplatňována v případě přechodu z prostředí s větším indexem lomu do prostředí s menším indexem lomu.

Uplatněním Snellova zákona spolu s dalšími goniometrickými pravidly lze dospět k rovnici pro výpočet směru paprsku lomu světla:

$$\vec{T_D} = \frac{\eta_1}{\eta_2} * \vec{I} - (\frac{\eta_1}{\eta_2} * cos\Theta_I + \sqrt{1 - sin^2\Theta_T}) * \vec{N} \iff \sin\Theta_I <= \frac{\eta_2}{\eta_1} \tag{3.5}$$

Další popis celého odvození se nalézá zde: [Web04].

Specifická situace může nastat tehdy, když paprsek narazí na lesklý a přitom poloprůhledný povrch. V takovém případě je nutno se rozhodnout, jak velký příspěvek dodá zrcadlový paprsek oproti paprsku lomu.

3.3.4 Fresnelovy rovnice

Při nárazu na poloprůhledný, lesklý objekt platí, že příspěvek zrcadlového i paprsku lomu pokrývá celý příspěvek osvětlení v místě průniku:

$$T + R = 1, \tag{3.6}$$

kde T je příspěvek lomu světla a R je příspěvek zrcadlového paprsku.

Poměr mezi T a R není náhodný, ale závisí na indexech materiálů η_1 a η_2 a rovněž na úhlu dopadu Θ_I a úhlu lomu Θ_T světla. Fresnelovy rovnice přistupují ke světlu z fyzikálního hlediska. Světlo je elektromagnetické vlnění, které může být polarizováno. Je-li polarizace světla $\perp$, směr elektromagnetického vlnění je kolmý na rovinu dopadu. Je-li polarizace $\parallel$, je směr vodorovný s rovinou dopadu. Fresnelovy rovnice lze upravit do tvaru: [dG06]

$$R_\perp(\Theta_i) = (\frac{\eta_1 * \cos\Theta_I - \eta_2 * \cos\Theta_T}{\eta_1 * \cos\Theta_I + \eta_2 * \cos\Theta_T})^2 \tag{3.7}$$

$$R_\parallel(\Theta_i) = (\frac{\eta_2 * \cos\Theta_I - \eta_1 * \cos\Theta_T}{\eta_2 * \cos\Theta_I + \eta_1 * \cos\Theta_T})^2 \tag{3.8}$$

Směry elektromagnetického vlnění přinášejí různě velký příspěvek. Často se však používá pouze jejich průměr. Ve výsledku lze příspěvek zrcadlového a paprsku lomu vyjádřit jako:

$$R(\Theta_i) = \begin{cases} \frac{R_\perp + R_\parallel}{2} & \text{pro} \quad \sin\Theta_I <= \frac{\eta_2}{\eta_1} \\ 1 & \text{pro} \quad \sin\Theta_I > \frac{\eta_2}{\eta_1} \end{cases} \tag{3.9}$$

$$T(\Theta_i) = 1 - R(\Theta_i). \tag{3.10}$$

Praktickým příkladem užití Fresnelových rovnic může být pohled na moře. Díváme-li se kolmo na hladinu moře, je voda průzračná a spatřujeme v ní kamení a písek. Podíváme-li se dále směrem k horizontu, voda ztrácí svoji průhlednost, zato však ve větší míře odráží oblohu nad sebou.

3.3.5 Urychlení sledování paprsků

Naivní sledování paprsků, kdy detekujeme průnik jednotlivých paprsků se všemi objekty, je výpočetně velmi náročný. Naznačuje to i asymptotická složitost $O(N * R)$, kde N značí počet objektů a R počet paprsků. Proto byl ray tracing předmětem mnoha výzkumných prací, které se ji snažily akcelerovat do praktického využití. Obecně jsou to tři okruhy možné akcelerace: urychlení výpočtu paprsků, snížení počtu paprsků a sledování více paprsků naráz.

3.3.5.1 Urychlení výpočtu paprsků

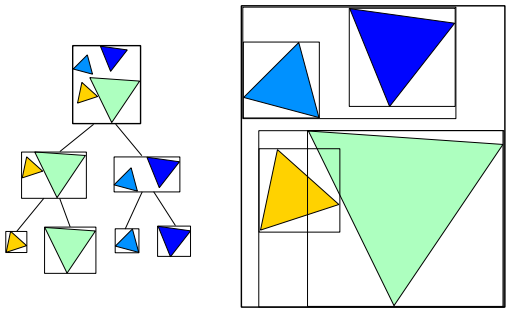
Často je hlavním důvodem náročnosti sledování paprsků detekce průniku paprsků s objekty. Podle [SAGC+12] tvoří 75% až 95% celkového výpočetního času. Proto je velmi žádoucí urychlit a rovněž snížit počet detekcí průniku paprsků s objekty.

Urychlení výpočtu paprsků lze docílit dvěma způsoby. Jednak vlastní optimalizací metody detekce průniku, např. snahou se vyhnout počítání barycentrických souřadnic trojúhelníku, nejsou-li dále třeba. Za druhé snížením počtu detekcí průniku; k tomuto účelu jsou zpravidla stavěny hierarchické datové struktury.

Hierarchických datových struktur je velké množství, proto zde bude představena jen jedna z nich, která byla použita i v nové metodě pro výpočet globálních efektů.

Hierarchie obálek (*Bounding Volume Hierarchy*, dále BVH) představuje binární strom, který sdružuje objekty scény do hierarchie obálek (viz obr. 3.6).

Obálkou může být jak paměťově nejméně náročná koule, osově zarovnaný kvádr (*Axis Aligned Bounding Box*, dále AABB), natočený kvádr (*Oriented bounding box*), komplexnější polytop, tak nejtěsnější konvexní obálka. Čím je obálka těsnější, tím více vzrůstají paměťové nároky obálky spolu s výpočetní náročností detekce průniku paprsku s obálkou. Proto je často jako kompromis zvolena AABB, která je určena jen minimálním a maximálním vrcholem obálky.



Obrázek 3.6: Příklad hierarchie obálek (BVH).

BVH může být stavěna odshora dolů (tříděním), odspoda nahoru (shlukováním) a inkrementálně (vkládáním). Stavba BVH odshora dolů je rychlá a snadno implementovatelná. Narozdíl od metody shlukováním, která je často výpočetně náročnější, nedosahuje tak kvalitních stromů. Obě metody – tříděním i shlukováním – potřebují mít všechna primitiva k dispozici na začátku stavby BVH (*off-line metoda*). Stavba BVH vkládáním je vhodná pro scény, kde se dostupnost objektů během chodu algoritmu mění. Zaměříme se pouze na stavbu BVH odshora dolů, která je jednoduše implementovatelná a dosahuje uspokojivých výsledků.

Při stavbě BVH odshora dolů postupujeme od obálky celé scény binárním rekurzivním dělením do doby, než je splněna daná ukončovací podmínka. Ukončovací podmínkou může být dosažení maximálního počtu objektů v uzlu BVH nebo maximální hloubka BVH. Při dělení uzlu BVH se snažíme nalézt co nejvhodnější řeznou osu a logicky rozdělit trojúhelníky mezi vzniklé listy BVH. Je zřejmé, že správné rozdělení trojúhelníků mezi uzly má zásadní vliv na výslednou kvalitu celé hierarchie. Výběr řezné osy může být vybrán na základě jednoduše cyklického střídání os x, y, z. Druhou možností je dělení v ose nejdelší strany obálky, což vede k rovnoměrnějšímu členění hierarchie. Rozdělení trojúhelníků je možné provést seřazením objektů podle jejich středu a dělícího pivota vybrat tak, aby bylo na levé i pravé straně stejné množství objektů (*Object Median*). Další možností je *Spatial Median*, který řadí objekty podle středu obálky. Dělícím pivotem se stává ten, který je uprostřed těchto objektů. Poslední možností, jak zvolit dělícího pivota, jsou heuristiky. Popíšeme jednu z nich, často používanou surface-area heuristiku.

Surface-area Heuristic (dále SAH) pracuje na základě odhadu ceny řezu. Paprsek, který protne rodičovskou obálku B_p s povrchem $SA(B_p)$, protne obálku jejího potomka B_c s povrchem $SA(B_c)$ s pravděpodobností $SA(B_c)/SA(B_p)$.

Máme-li rodiče B a potomky B_1 a B_2, lze pomocí SAH vyjádřit cenu traverzace jako:

$$C = C_t + \frac{SA(B_1)}{SA(B)}|P_1|C_i + \frac{SA(B_2)}{SA(B)}|P_2|C_i, \tag{3.11}$$

kde C_t je cena kroku traverzace, $|P_1|$ a $|P_2|$ značí počet trojúhelníků v jednotlivých potomcích a C_i je cena průniku paprsku s trojúhelníkem [SFD09].

3.3.5.2 Snížení počtu paprsků

Další možností, jak urychlit ray tracing, je snížení počtu paprsků.

Ve scéně, kde chceme vykreslit odlesky, můžeme omezit hloubku rekurze zrcadlového paprsku konstantou. S hlubší rekurzí není efekt odlesku již patrný a tak lze tyto výpočty od určitého momentu vynechat.

V dalším případě se můžeme rozhodnout nevrhat paprsek pro každý pixel zvlášť, ale adaptivně vzorkovat. Tj. zaměřit se na místa, která nesou informaci, a pro ně použít více paprsků než pro místa, která informaci nenesou. Zbylé pixely, pro které jsme nevrhali paprsky, můžeme dopočítat bilineární interpolací vzorkovaných pixelů.

3.3.5.3 Vrhání více paprsků naráz

Vrhání paprsků lze urychlit převedením navzájem nezávislých výpočtů na GPU. Jedná se především o výpočty spojené s traverzací hierarchickou strukturou a detekování průniku paprsků s objekty.

Jedním z významných přínosů je práce představená v roce 2009 [AL09], díky níž bylo docíleno momentálně nejrychlejšího GPU ray tracingu. Práce se zaměřuje na pochopení efektivity traverzace paprsků na GPU a na základě těchto poznatků představuje řešení, které podstatně navyšuje rychlost celé traverzace. Výsledky jsou demonstrovány pro primární, Ambient Occlusion a Diffuse Interreflection paprsky. Obdobně jak na CPU, tak na GPU paprsky používají svůj zásobník, pomocí kterého postupují skrz hierarchii obálek. Detekce průniku paprsku s obálkou či objektem je optimalizována v CUDA 2.1.

Další typ vrhání paprsků na GPU je podrobně popsán v následující části.

3.4 Vrhání paprsků pomocí hierarchie paprsků

V roce 2007 představili D. Roger a kol. metodu [RAH07], která přesouvá ray tracing zcela na stranu GPU. V článku se zaměřují především na zrcadlové paprsky, ale tato metoda je schopná adaptace na libovolný druh paprsků.

Algoritmus vypadá následovně: [RAH07]

1. Vykreslení scény, bez odlesků.

2. Generování první množiny sekundárních paprsků

3. Zkonstruování hierarchie paprsků

   a) Zpracování sekvence párů (uzel hierarchie, trojúhelníky)

   b) Rekurzivní dělení uzlů

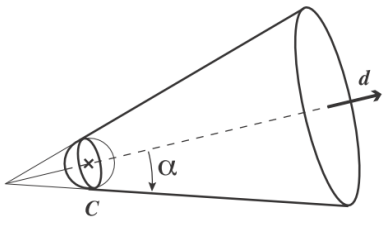
   c) Odstranění irelevantních trojúhelníků

4. Finální průnik paprsků s trojúhelníky a stínování

Prvním krokem je vykreslení scény bez odlesků. Využit je standardní rasterizér s per-pixel osvětlením (Fragment shader). Stejný shader je použit ke generování první množiny počátků a směrů paprsků. Poté je vybudována hierarchie paprsků. Jádrem algoritmu je průnik uzlů paprsků s trojúhelníky. Každý uzel představuje svazek paprsků, které jsou vrhány na kulovité obálky trojúhelníků scény. Výsledkem je kulovitá obálka trojúhelníků (*Bounding Sphere*), kterou svazek paprsků protíná. Traverzace hierarchií paprsků začíná odshora dolů, od kořene paprsků hlouběji až k listům hierarchie. V závěru traverzace obsahuje každý paprsek reference na trojúhelníky, jejichž kulovitou obálku protíná. Poslední krok spočívá v detekci průniků paprsků s trojúhelníky, nalezení nejbližšího průniku a závěrečné stínování.

Stavba hierarchie paprsků

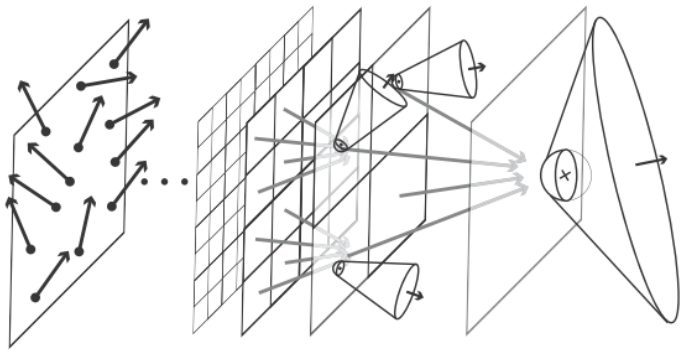
Efektivita algoritmu významně závisí na stavbě hierarchie paprsků. Tato metoda [RAH07] používá velmi kompaktní způsob shlukování paprsků do svazků. První část uzlu hierarchie představuje koule, která obaluje počátky paprsků svazku. Druhou částí je kužel, který obsahuje zmíněnou kouli a dále směry shluknutých paprsků, viz obr. 3.7. Tento kompaktní způsob umožňuje jeden svazek paprsků zakódovat pouze 8 floaty: 4 pro kouli (střed a poloměr) a 4 pro kužel (směr a úhel rozšíření).



Obrázek 3.7: Uzel hierachie paprsků tvořený koulí a kuželem. Převzato z [RAH07].

Samotná stavba hierarchie je prováděna odspoda nahoru. Počínaje jednotlivými sekundárními paprsky až ke kořeni. V každé úrovni hierarchie se

provede čtvercovité shlukování svazků paprsků (v první úrovni pouze samostatných paprsků). Každý rodič hierarchie tedy odkazuje na čtyři potomky o úroveň níže. Přehledně je to ilusturováno na obr. 3.8. Tento postup má fixní počet kroků a je zcela přesunut na stranu GPU. Obdobným postupem je generování mip-map.



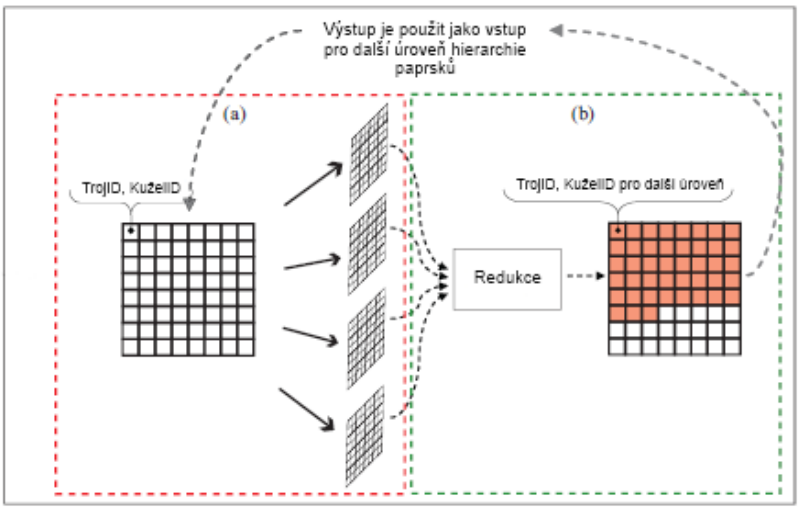
Obrázek 3.8: Stavba hierachie paprsků odspoda nahoru. Převzato z [RAH07].

Jakmile je postavena hierarchie paprsků, můžeme se přesunout k samotné traverzaci.

Traverzace hierarchie paprsků

Traverzace probíhá odshora dolů, tj. od kořene hierarchie až k listům. Na počátku je nutné vyplnit texturu všemi trojúhelníky tak, aby každá buňka obsahovala pár (id trojúhelníku, id kořenového kuželu). Pro všechny čtyři potomky kořene se detekuje průnik s kulovitými obálkami trojúhelníků. Naleznemeli průnik s obálkou, uloží se pár (id trojúhelníku, id kuželu) do připravené textury. V opačném případě, nebyl-li nalezen průnik, zůstane texel prázdný. Každý ze čtyř potomků zapisuje nalezené průniky do své přiřazené textury. Výsledkem jsou tedy čtyři textury, které obsahují na některých místech pár (id trojúhelníku, id uzlu hierarchie paprsků). K ušetření paměti byly představeny redukční metody, které sloučí tyto čtyři textury do jedné textury, která je pak vstupem pro úroveň hierarchie níže, viz 3.9.

Po traverzaci obsahuje každý paprsek seznam id trojúhelníků, které jsou jím potenciálně protnuty. Paprsek protíná prozatím jen kulovitou obálku trojúhelníků. Na závěr se spočítá průnik paprsků s trojúhelníky a pro nejbližší průnik dopočítá osvětlení.



Obrázek 3.9: Přehled algoritmu vrhání paprsků pomocí hierarchie paprsků. Přeloženo z [RAH07].

3.5 Řešení viditelnosti pomocí dotazů zastínění

K odstranění neviditelných, popř. zastíněných objektů z rasterizačního řetězce byly vyvinuty dotazy zastínění (*Occlusion Query*, dále OQ). Výsledkem OQ je počet fragmentů, které se dostaly na výstup rasterizačního řetězce. Je-li počet těchto fragmentů roven nule, je objekt nebo jeho obálka zcela zastíněna a lze jej ze zpracování odstranit.

Postavíme-li nad scénou hierarchii obálek (BVH, viz kap. 3.3.5.1), můžeme traverzování BVH za pomoci OQ efektivně prořezat.

Podíváme-li se blíže na OQ, průběh emitace OQ vypadá následovně: (podle [PF05])

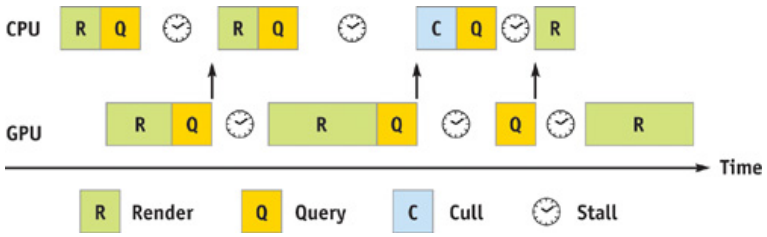
1. Inicializace OQ
2. Zákaz zápisu do color buffer a depth buffer. Vypnutí přebytečných stavů OpenGL
3. Vykreslení obálky objektu
4. Ukončení OQ
5. Vyčkání na výsledek dotazu

S dotazy zastínění jsou spojeny dvě obtíže, které celé vykreslování mohou značně zpomalit: čekání na výsledek OQ a nadměrné množství OQ. Tyto

problémy řeší metoda s názvem *Coherent hierarchical culling*, popsaná v knize [PF05].

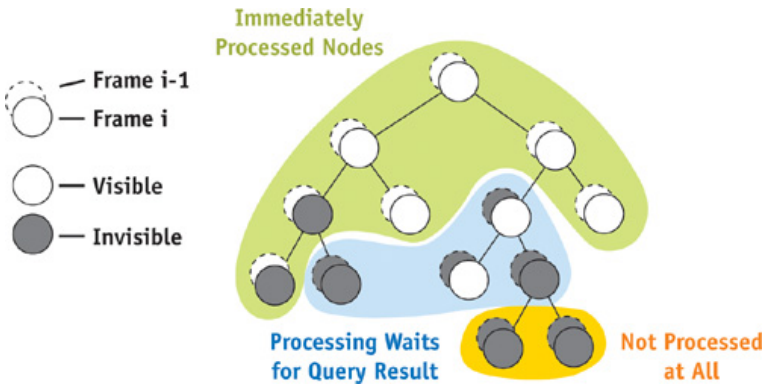
3.5.1 Čekání na výsledek dotazu zastínění

Problém nastává při emitaci OQ – čekání na výsledek OQ, což při vykreslování způsobuje čekání CPU na GPU (viz obr. 3.10).



Obrázek 3.10: Čekání CPU na GPU a minimální produktivita GPU způsobená dotazy zastínění. Převzato z [PF05].

Čekání lze vhodně předejít předvídáním výsledku dotazu. Předpověď není čistě náhodná, ale vychází z časové koherence za sebou jdoucích snímků. Je velmi pravděpodobné, že mezi dvěma po sobě jdoucími snímky budou zastíněné i nezastíněné objekty totožné. Chybovat můžeme ve dvou případech. Zaprvé, je-li objekt považován za viditelný a přitom je zastíněn – vykreslení objektu bylo zbytečné. Zadruhé, je-li objekt považován za zastíněný a přitom je viditelný – objekt měl být vykreslen. V obou případech není příliš důležitý náš odhad – jakmile se dozvíme výsledek dotazu, zkontrolujeme správnost našeho odhadu. Bude-li to vhodné, upravíme viditelnost pro nadcházející snímek a vykreslíme nově viditelné objekty. Oba případy jsou znázorněny na obrázku 3.11.



Obrázek 3.11: Předvídání výsledku dotazu zastínění. Převzato z [PF05].

3.5.2 Nadměrné množství dotazů zastínění

Druhý problém nastává v případě, kdy je příliš mnoho objektů viditelných. Počet dotazů zastínění se navýší do té míry, že se algoritmus stává pomalejším než bez použití OQ.

Nadbytek dotazů zastínění je způsoben zejména dotazováním vnitřních uzlů BVH. Snížení počtu OQ se docílí pouze dotazováním dříve viditelných listů BVH a největších možných zastíněných uzlů. Počet dotazů zastínění tím ve výsledku klesne k počtu viditelných objektů ve scéně.

Spojíme-li obě řešení, tedy zamezíme nadměrnému čekání na výsledek dotazu i nadbytku dotazů, dosáhneme významného urychlení vykreslování. Těmito optimalizacemi je inspirována nová metoda vrhání paprsků, která využívá dotazů zastínění.

Kapitola 4

Metoda vrhání paprsků pomocí dotazů zastínění

U rozsáhlejších scén je dobré zaměřit se pouze na ty objekty, které mají vliv na výsledný obrázek. Jinak by došlo k mnoha zbytečným výpočtům, které by vykreslování zbytečně ztížily. Jednoduché je to při přímém osvětlení: Je-li objekt zakryt bližším objektem, je možné jej na základě OQ odstranit a tak ušetřit čas výpočtu jeho osvětlení. Chceme-li však postihnout i nepřímé osvětlení, může mít každý objekt roli ve výsledném osvětlení a odstranění zastíněných objektů se tak stává obtížným úkolem.

Nová metoda vrhání paprsků se pokouší začlenit OQ při párování pixelů obrazu s AABB BVH. Barva každého pixelu obrazu je určena jedním nebo více sledovanými paprsky. Nad pixely lze postavit hierarchii obdélníků, která rozděluje pixely a tím i paprsky do menších skupin. Vrháme-li pak skupinu paprsků na AABB, můžeme pomocí OQ určit, zda svazek paprsků protíná obálku, nebo ne.

Následující oddíl obsahuje přehled celého algoritmu a podrobný popis jednotlivých částí.

4.1 Přehled algoritmu

Algoritmus je založen na odloženém stínování (viz 2.2), které umožňuje výpočet osvětlení jen pro viditelné místa scény.

Druhou výhodou odloženého stínování je, že pomocí G-Bufferu je možné generovat sekundární paprsky a tím simulovat globální efekty, jako např. v kap. 3.4.

Aby se snížila detekce průniku paprsků s objekty, je na začátku algoritmu postavena hierarchie BVH, která shlukuje objekty do osově zarovnaných kvádrů (AABB). Dále je postavena hierarchie paprsků v prostoru obrazu (*Screen Patch Hierarchy*, dále SPH), která shlukuje paprsky do obdélníků. Následuje

samotné párování uzlů SPH mezi uzly BVH. Začíná se tím, že protnou kořeny obou hierarchií. Protíná-li alespoň jeden paprsek obálku scény, označíme pár za viditelný a posuneme se jednou z hierarchií o patro níže. Dosáhneme-li místa, kdy již nemůžeme ani v jedné hierarchii níže postupovat, vrhneme paprsky na objekty uzlu BVH a dopočítáme osvětlení.

Přehled algoritmu se nachází zde:

1. Vykreslení scény a uložení informací o geometrii a materiálech do G-bufferu
2. Výpočet Phongova osvětlení
3. Inicializace sekundárních paprsků
4. Párování BVH a SPH za pomoci OQ
   a) Výpočet osvětlení předchozích viditelných párů
   b) Sdružování předchozích neviditelných párů
   c) Rekurzivní párování BVH a SPH za pomoci OQ
   d) Dopočítání osvětlení nově viditelných párů
5. Závěrečné prolnutí osvětlení paprsků s Phongovým osvětlením

Následuje podrobnější popis datových struktur a celého algoritmu.

4.2 Datové struktury

Před samotným vykreslováním je třeba zkonstruovat dvě hierarchie: hierarchii v prostoru objektů a hierarchii v prostoru obrazu.

4.2.1 Hierarchie obálek (BVH)

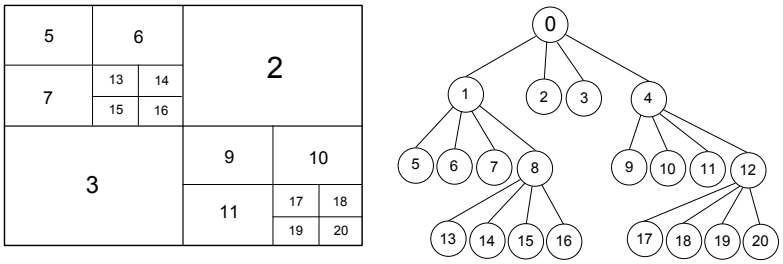
Kapitola 3.3.5.1 obsahuje jak různé způsoby stavby hierarchie obálek, tak jejich možné využití. V BVH platí, že je zde každý objekt obsažen právě jednou, i když se jejich obálky mohou vzájemně překrývat.

V této metodě vrhání paprsků byla zvolena stavba BVH odshora dolů (tříděním), která je rychlá a v implementaci nenáročná. Typ a pozice řezné roviny jsou vybrány na základě surface-area heuristiky (viz kap. 3.3.5.1).

4.2.2 Hierarchické dělení prostoru obrazu (SPH)

Hierarchie postavená nad pixely obrazu (*Screen Patch Hierarchy*, dále SPH) rozděluje paprsky do skupin ohraničených obdélníkem. Rekurzivní dělení je prováděno vždy v obou osách obdélníků zároveň, což vede ke vzniku čtyř stejně velkých obdélníků. Jedná se tedy o quad tree, kde každý z rodičů má

čtyři potomky (viz obr. 4.1). Ukončovací podmínkou při stavbě SPH je buď hloubka rekurze nebo minimální počet pixelů v obdélníku.



Obrázek 4.1: Příklad hierarchie postavené nad prostorem obrazu (SPH).

Po stavbě hierarchií BVH a SPH se přistupuje k samotnému vykreslení scény.

4.3 Zobrazovací průchody v rasterizačním řetězci

Nová metoda vrhání paprsků využívá víceprůchodové vykreslování, aby dosáhla nepřímého osvětlení scény. Prvním zobrazovacím průchodem je rasterizace scény s uložením geometrie a materiálů do G-bufferu.

4.3.1 Vykreslení scény s uložením dat do G-bufferu

V prvním průchodu rasterizačním řetězcem se scéna pomocí *ModelViewProjection* matice transformuje do ořezového prostoru, odkud je dále převedena do prostoru obrazu. Připojená hloubková mapa zachovává hloubku nejbližších fragmentů, jejichž geometrie a materiály se uloží do připravených textur (G-buffer).

Struktura textur v G-bufferu je velmi důležitá a při špatném rozložení může výrazně zpomalit celý běh aplikace. Důležité je použít sférické kódování normály, které třírozměrnou normálu převede do dvou úhlů. Při použití 4B na jeden texel lze tak dosáhnout přesnější reprezentace normály, díky čemuž je možné přesněji vypočítat osvětlení. Určité zpomalení způsobí kódování a dekódování normály na GPU, nijak zásadně se tím však běh algoritmu nezpomalí.

Ambientní barva bývá obvykle pro celou scénu shodná. Proto není třeba ji ukládat, stačí na závěr přičíst konstantní hodnotu.

	Byte			
Texture	0	1	2	3
0	depth			
1	normal.x		normal.y	
2	diffuse.r	diffuse.g	diffuse.b	free
3	specular.r	specular.g	specular.b	free
4	shininess	transparency	indexOfRefraction	free

Tabulka 4.1: Struktura textur pro odložené stínování.

Ve výsledku je třeba, aby se G-buffer skládal z pěti textur, kde by každá buňka textury měla velikost čtyř bajtů. Struktura G-bufferu je popsána v tabulce 4.1.

Po rasterizaci scény a uložení informace o viditelné geometrii a materiálech následuje výpočet Phongova osvětlení.

4.3.2 Výpočet Phongova osvětlení

Z předchozího zobrazovacího průchodu máme k dispozici informaci o hloubce geometrie a použitých materiálech. Nyní stačí vykreslit přes celou obrazovku obdélník a po jeho rasterizaci dopočíst každému pixelu Phongovo osvětlení (viz kap. 2.1.1).

Hloubka geometrie nestačí pro výpočet Phongova osvětlení. Je potřeba dopočítat pozici ve světových souřadnicích. Souřadnice fragmentu se nejprve převede ze souřadnic obrazu do normalizovaných souřadnic a poté inverzní maticí *InvViewProjection* do světových souřadnic. Pozice ve světových souřadnicích je důležitá rovněž proto, že je přibližným počátkem první množiny sekundárních paprsků.

4.3.3 Inicializace sekundárních paprsků

Po výpočtu Phongova osvětlení se ve stejném zobrazovacím průchodu inicializují sekundární paprsky, tj. stínové, zrcadlové a paprsky lomu.

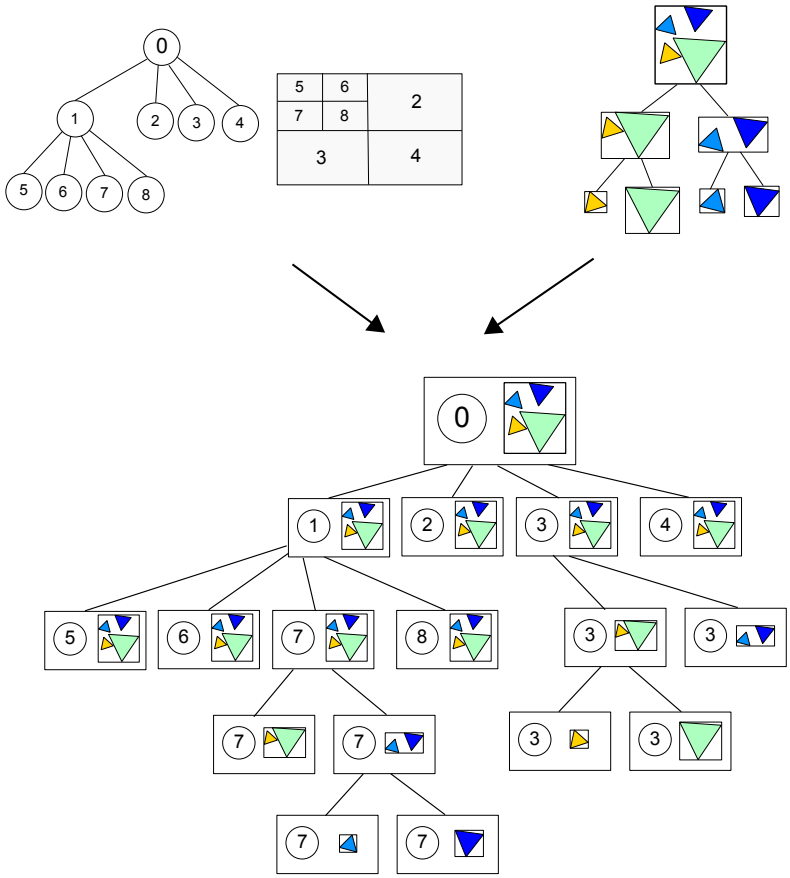
Pozice těchto paprsků se shoduje s vypočítanou pozicí z hloubkové textury, která je posunuta o malý offset ve směru paprsků. Tím se zabrání průniku paprsků s plochou, z které vycházejí. Výpočet směrů paprsků je popsán v kapitole o ray tracingu (viz kap. 3.3).

Sekundární paprsky umožňují vypočítat nepřímé osvětlení, které je výpočetně stále velmi náročné. Proto byla vyvinuta nová metoda, který přináší značné zrychlení.

4.3.4 Párování BVH a SPH za pomoci dotazů zastínění

Principem nové metody je párování uzlů BVH s uzly SPH, neboli obálky s množinami paprsků obklopené obdélníkem. To, zda je pár viditelný, nebo ne, určuje výsledek OQ.

Důležité je, že z párů se průběžně staví hierarchie (*QueryPair Hierarchy*, dále QPH), kde kořen QPH odpovídá kořenům BVH a SPH. Každý z vnitřních uzlů může mít čtyři nebo dva potomky, v závislosti na tom, zda se dělí v BVH (dva potomci) nebo SPH (čtyři potomci)(viz obr. 4.2). Dělit se může pouze viditelný pár a to na základě výsledku OQ. Pokud je pár neviditelný (obálku BVH neprotíná žádný z dané množiny paprsků), dělení postrádá smysl a pár se stává listem QPH .



Obrázek 4.2: Příklad párování BVH s hierarchií nad prostorem obrazu.

Je pravděpodobné, že většina paprsků mezi dvěma po sobě jdoucími snímky

bude protínat totožné objekty nebo alespoň jejich obálky. Této časové koherence využívá párování BVH a SPH, které nestaví QPH pro každý snímek, ale průběžně ji aktualizuje. Vždy se vychází ze stavu QPH předchozího snímku s tím, že v iniciálním snímku se vychází od páru spojujícího kořen BVH s kořenem SPH.

S využitím předvídání výsledku OQ a dotazování se pouze listů QPH (viz kap. 3.5) byly navrženy čtyři kroky pro rychlou traverzaci skupin paprsků skrze BVH.

4.3.4.1 Sdružení předchozích neviditelných párů

Jak je výše již napsáno, nová metoda využívá časové koherence po sobě jdoucích snímků. Proto v každém snímku vycházíme z předešlé podoby QPH, v které jsou začleněny viditelné a neviditelné páry.

Prvním krokem je sdružení neviditelných párů. Při náhlém zastínění většího počtu objektů, může nastat situace, kdy všichni potomci jednoho rodiče jsou zastíněni. V této situaci nemá smysl dále uchovávat tyto čtyři potomky, ale naopak je třeba je odstranit a ponechat pouze jejich rodiče, který se stane listem QPH. Díky tomu se zamezí zbytečné traverzaci skrz QPH a nadbytečnému dotazování neviditelných párů.

Po sdružení neviditelných párů následuje výpočet osvětlení viditelných párů.

4.3.4.2 Výpočet osvětlení předchozích viditelných párů

V tomto kroku představuje každý vnitřní uzel QPH viditelný pár a zároveň je viditelným párem i některý z listů. Výpočet osvětlení vnitřních uzlů je zbytečný, protože tyto páry jsou postihnuty listy. Postačí proto vrhat paprsky na objekty v listech QPH a dopočítat z průniků osvětlení.

Při hledání viditelných listů QPH si rovněž ukládáme viditelné i neviditelné listy do fronty *TraversalQueue*, kterou následně použijeme v dalším kroku metody.

4.3.4.3 Rekurzivní párování BVH a SPH za pomoci OQ

K párování BVH a SPH použijeme dvě fronty. První v pořadí je fronta *TraversalQueue*, která obsahuje listy QPH. Druhá je *QueryQueue*, která bude průběžně ukládat emitované OQ.

Samotné párování probíhá ve smyčce, která končí, teprve když jsou obě fronty prázdné. Tj. až všechny dotazy a listy QPH jsou zpracovány.

Prvním krokem párování je emitace OQ pro všechny neviditelné páry uložené v *TraversalQueue*. Tím si ověříme, jestli i nadále jsou neviditelnými, nebo se staly viditelnými. Pro viditelné páry rovněž platí, že může dojít ke změně jejich viditelnosti a stanou se zastíněnými. Osvětlení viditelných párů je pokaždé vypočítáno v předchozím kroku, jejich případné zastínění do výsledného

obrázku nezanese žádnou chybu. Z toho důvodu je možné dotazovat se viditelných párů jen náhodně, pro některé ze snímků, a tím vyvažovat mezi počtem emitovaných OQ a počtem vykreslených párů. Jsou-li zpracovány všechny páry v *TraversalQueue*, následuje vyhodnocení OQ uložených v *QueryQueue*.

Jak bylo popsáno v kapitole 3.5, není výsledek emitovaného OQ hned dostupný, ale musí se počkat na jeho vyhodnocení. Částečně se tomu vyhýbá předchozím dávkováním OQ, úplně odstranit časovou prodlevu při čekání však nelze. Podle výsledku OQ z *QueryQueue* určíme, zda-li je pár viditelný nebo neviditelný. Je-li pár neviditelný, dělení páru nemá již další smysl. V případě viditelného páru je nutné se rozhodnout, zda-li ho ještě dále dělit nebo na základě ukončovacího kritéria dělení ukončit. Viditelný pár může být rozdělen buď na dva nebo čtyři páry. Pokud dělíme geometrii páru (uzel BVH) vzniknou dva páry, kde oba obsahují stejně velkou množinu paprsků, ale rozdílnou obálkou. Dělíme-li prostor obrazu, vznikají čtyři páry, kde všechny páry obsahují stejnou obálku, ale různou množinu paprsků. Všechny vzniklé páry se ukládají v *TraversalQueue* a jsou opět zpracovány výše zmíněným způsobem.

Nalezneme-li při zpracování OQ nově viditelné páry, uložíme si je do vektoru pro pozdější zpracování.

4.3.4.4 Výpočet osvětlení nově viditelných párů

V okamžiku, kdy např. změníme pohled kamery, vznikají nové viditelné páry, které potřebujeme dále zpracovat. Nemůžeme zpracování nechat na začátek dalšího snímku, protože by tak vznikaly nežádoucí artefakty. Proto pro každý nově viditelný pár vrhneme paprsky páru na příslušné objekty uzlu BVH a dopočítáme osvětlení.

4.3.5 Závěrečné prolnutí přímého osvětlení s nepřímým osvětlením

Předchozí párování může zahrnovat různé typy paprsků. V ray tracingu jsou to kromě primárních paprsků, paprsky stínové, zrcadlové a paprsky lomu. K dosažení výsledného obrázku je třeba prolnout dopočítané osvětlení paprsků s Phongovým osvětlením, které jsme získali z předchozího kroku pomocí odloženého stínování.

4.4 Parametry algoritmu

Novou metodu můžeme vhodně konfigurovat v závislosti na druhu scény. Lze tak dosáhnout optimálního dělení mezi BVH a SPH, které ve výsledku znamená rychlejší vykreslení scény.

4.4.1 Kritéria dělení BVH a SPH

Kritérium dělení BVH a SPH ovlivňuje výslednou kvalitu hierarchie párů. Dělení v SPH zabraňuje překreslení obdélníků, čímž snižuje počet opakovaně vržených paprsků. Dělení v SPH samo o sobě nestačí, je nutné snížit počet počítaných průsečíků, což umožňuje dělení v BVH. Při vývoji nové metody byla implementována a otestována dvě kritéria dělení.

Prvním kritériem je dělení v prostoru obrazu, dokud se nedostaneme k listům SPH. Spoléháme se přitom na časovou koherenci, která redukuje počty dělení v SPH. Například, dělíme-li v SPH a všichni potomci se stanou neviditelnými, nemá smysl je dále ponechávat a jsou z traverzačního stromu odstraněny.

Druhým kritériem je dělit na základě porovnání relativního povrchu obálky BVH a obdélníku SPH. Tím se snažíme dosáhnout vyvážení dělení, tak aby velikost obdélníku SPH přibližně odpovídala velikosti obálky BVH.

4.4.2 Ukončovací podmínka dělení

Ukončovací podmínka dělení je závislá na hierarchii obálek a obrazu. K ukončení dochází, když se dosáhne listů BVH a SPH. Pro BVH může být ukončovacím kritériem hloubka dělení, počet trojúhelníků aj. Pro SPH: hloubka dělení nebo maximální počet pixelů obdélníku.

4.4.3 Využití časové koherence

Nová metoda využívá časové koherence po sobě jdoucích stromů. Hierarchie párů není stavěna pro každý snímek nově, ale využívá řezu hierarchie párů předchozího snímku.

Kapitola 5

Implementace

Při implementaci nové metody vrhání paprsků jsem vycházel z implementace Dr. Dipl.-Inf. Mattausche, která mi byla poskytnuta. Protože se jedná o rozsáhlejší projekt, který vyžaduje mnoho předchozích znalostí, rozhodl jsem se po poradě s vedoucím Ing.Bittnerem, Ph.D. nově metodu implementovat a tak lépe pochopit celý její obsah. Proto nadále budu představovat vlastní implementaci, která se vůči výše uvedené značně liší. Převzal jsem z ní pouze samotné párování BVH s SPH vysvětlené v sekci 4.3.4, struktury *QueryPair* a *GiOcclusionQuery* a optimalizované metody pro výpočet průniku paprsků s AABB a s trojúhelníky.

Implementace podporuje hraniční reprezentaci dat, kde objekt nebo povrch scény je trojúhelník.

5.1 Implementační prostředí

Pro zobrazování dat je použita knihovna OpenGL, která slouží k tvorbě 2D i 3D grafických aplikací. Používá se nejen při vytváření her, ale i k vizualizaci vědeckých poznatků nebo v aplikacích virtuální reality. OpenGL lze použít na všech široce rozšířených platformách. Aktuální verze je OpenGL 4.4 představená v srpnu 2013. Novější verze OpenGL jsou úzce spjaty s programováním na GPU v jazyce GLSL.

Jazyk GLSL stylem odpovídá programování v jazyku C, kde jednotlivé příkazy odpovídají instrukcím, které má grafická karta zpracovat. V dřívějších verzích OpenGL nebylo možné přistupovat k GPU jinak než nastavením stavů OpenGL a tak ovlivnit výsledné zobrazení. Proto s OpenGL 2.0 přichází jazyk GLSL, který umožňuje programátorovi lépe využít GPU.

Existují tři základní programy pro GPU: program zpracovávající vrcholy geometrie (Vertex Shader), program ovlivňující geometrii objektů (Geometry Shader) a program pracující nad pixely obrazu (Fragment Shader). Programy mohou být kompilovány za chodu aplikace do jednoho programu, který, pokud

Texture	Format	Type
Depth	GL_DC24	GL_DC
Normal	GL_RG16F	GL_FLOAT
Diffuse	GL_RGBA8	GL_UNSIGNED_BYTE
Specular	GL_RGBA8	GL_UNSIGNED_BYTE
Optical props	GL_RGB16F	GL_FLOAT

Tabulka 5.1: Nastavení textur v G-bufferu. GL_DC je zkratkou pro GL_DEPTH_COMPONENT.

je aktivován, nahrazuje statický zobrazovací řetězec a sám zajišťuje zpracování geometrie.

5.2 Použité textury

Při implementaci nové metody bylo použito množství textur, které umožnily víceprůchodové vykreslování.

Jedny z použitých jsou textury tvořící G-buffer, pro přehled zaznamenány v tabulce 5.1). Pro normálu a optické vlastnosti bylo nutné použít typ GL_FLOAT, protože jejich hodnoty leží mimo rozsah [0,1].

Další textury byly použity při vrhání paprsků. Pro záznam každého druhu paprsků (pozice a směr) slouží dvě textury ve formátu GL_RGB32F. Pro rekurzivní paprsky (lomu a odlesku) je třeba přidat ještě další dvě textury stejného formátu, které umožní současné zapisování nových paprsků se čtením předchozích paprsků. Pro každý druh paprsku jsou průběžné výsledky ukládány do dalších textur, které jsou ve formátu GL_RGBA8. Všechny průběžné výsledky se nakonec propojí a uloží do textury formátu GL_RGBA8, která je zobrazená na výstup vykreslování.

5.3 Použité knihovny

Při implementaci byly použity různé knihovny a zdrojové soubory, které jsou shrnuty v následujících odstavcích.

Výchozím bodem mé implementace je Minimax aplikace poskytnutá Ing. Bittnerem Ph.D. Minimax umožňuje mimo jiné stavbu BVH za použití Surface-area heuristiky či zadávání parametrů bez nutnosti další kompilace programu.

Načítání objektů ve formátu .obj a materiálů v .mtl zajišťuje třída *ModelOBJ* převzatá z projektu [AL09]. K načítání textur slouží dvě knihovny: DevIL 1.7.8 a pro obrázky formátu PNG Libpng 1.6.10. Přístup k OpenGL zajišťují knihovny GLEW 7.0 a GLFW 2.7. Uživatelské rozhraní zprostředkovává knihovna AntTweakBar 1.13. Export výsledků měření do MS Excelu umožnila knihovna LibXL 3.5.4.

5.4 Přehled struktur tříd

Tato sekce se zabývá datovým složením struktur, které byly součástí některé z hierarchií.

Hierarchie obálek (BVH) je složena z vnitřních uzlů a listů. Každý z uzlů či listů BVH si uchovává své id, řeznou osu, hloubku, obklopující obálku a odkaz na rodiče. Také obsahuje index prvního a posledního trojúhelníku v globálním poli trojúhelníků, protože párování (viz 4.3.4) může být ukončeno v jakékoliv úrovni BVH. Vnitřní uzel obsahuje navíc odkaz na levého a pravého potomka. Datová reprezentace listu a vnitřního uzlu BVH vypadá následovně:

```
struct BvhNode{

  int id;
  char axis;              // řezná osa
  unsigned char depth;    // hloubka uzlu
  vec3 boxMin;            // minimální bod obálky AABB
  vec3 boxMax;            // maximální bod obálky AABB
  BvhNode *parent;        // ukazatel na rodiče uzlu

  /** index na první a poslední trojúhelník
   v globálním poli trojúhelníků */
  int first;
  int last;

};

struct BvhInterior : BvhNode{

  BvhNode *left;    // ukazatel na levého potomka
  BvhNode *right;   // ukazatel na pravého potomka
};
```

Hierarchie postavená nad obrazem (SPH) je složena z obdélníků, k jejichž reprezentaci stačí pouze dva dvousložkové vektory, minimální a maximální vrchol obdélníku:

```
struct Rectangle{

  vec2 min;   // minimální bod obdélníku
  vec2 max;   // maximální bod obdélníku
};
```

Potomky obdélníků není třeba ukládat, protože obrazovka je dělena vždy v obou osách najednou a jelikož jsou obdélníky uloženy v jednom poli, můžeme

k potomkům přistupovat dopočítáním indexů. Platí, že obdélník s indexem i má potomky na pozicích od $4 * i + 1$ po $4 * i + 4$.

Hierarchie párů (QPH) je tvořena páry, jejichž struktura je následující:

```
struct QueryPair{

  BvhNode *node;          // ukazatel na uzel BVH
  int ScreenPatchIndex; // index obdélníku v poli obdélníků
  int Depth;              // hloubka páru
  bool IsVisible;         // zda-li je pár viditelný
  int RenderedFrame;// snímek, v kterém byl pár vykreslen

  GiOcclusionQuery *query;// ukazatel na dotaz zastínění
  QueryPair *children[4]; // ukazatele na čtyři potomky páru
};
```

Počet definovaných ukazatelů na potomky páru je závislý na typu dělení. V případě, že byl pár dělen v prostoru obrazu jsou určeny čtyři páry. Při dělení uzlu BVH, jsou to pouze dva potomci páru. Jedná-li se o list, ani jeden z ukazatelů není definován.

5.5 Vývojové prostředí

Implementace byla vyvíjena v prostředí MS Visual Studio 2010. K ladění aplikace byl kromě vestavěného debuggeru VS 2010 použit GDebugger 5.8.1, který sloužil ke kontrole obsahu textur při víceprůchodovém vykreslování.

5.6 Uživatelské rozhraní

Uživatelské rozhraní (*Graphical User Interface*, dále GUI) vzniklo při použití knihovny AntTweakBar 1.13. Umožňuje pohyb po scéně, zobrazení textových informací, nastavení parametrů a provádění příkazů. Ukázka GUI je na obr. 5.1.

Při spuštění aplikace je scéna standardně vykreslována rasterizací. Je to však možno změnit a vykreslovat scénu vrháním primárních paprsků. GUI umožňuje vypínání či zapínání vykreslování stínů, odlesků a lomů světla.

Důležitou částí uživatelského rozhraní zaujímají hierarchie obálek (BVH) a hierarchie nad obrazem (SPH). Tyto hierarchie je možné za běhu aplikace přestavět a jejich strukturu zobrazit. Pro přehlednost je umožněno zobrazení pouze aktuálně nastavené úrovně hierarchie.

Při vývoji implementace vznikly ještě další funkce, které umožňují zobrazit pomocné textury a světlo. GUI zobrazuje také celkový čas vykreslení s časy jednotlivých vykreslovacích průchodů. Pro hierarchii párů zobrazuje počet emitovaných dotazů zastínění, generovaných a vykreslených párů atd.

Pomocí GUI lze také měnit pozici světla a parametry kamery (near plane, FoV apod.). Všechny změny v shaderech lze kompilovat za chodu aplikace.

Obrázek 5.1: Uživatelské rozhraní aplikace.

Kapitola 6

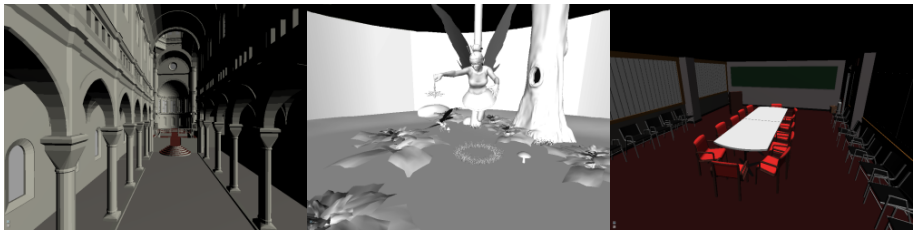
Výsledky

V předchozí kapitole byla popsána implementace nové metody vrhání paprsků, na kterou navazují výsledky práce získané testováním.

Nejprve jsou představeny testovací scény a hardware, na kterém se testy prováděly.

6.1 Testovací scény

Nová metoda byla testována na třech rozsáhlejších scénách: *Sibenik* (75 283 trojúhelníků), *Fairyforest* (174 117 trojúhelníků) a *Conference* (331 179 trojúhelníků) (viz 6.1). Scény byly staženy z volně dostupného zdroje – Morgan McGuire's Computer Graphics Archive [McG11].



Obrázek 6.1: Testovací scény. Zleva Sibenik (75 283 trojúhelníků), FairyForest (174 117 trojúhelníků) a Conference (331 179 trojúhelníků).

6.2 Testovací hardware

Testování bylo prováděno ve školní laboratoři ČVUT na počítači s grafickou kartou GeForce GTX 470. Tato karta podporuje OpenGL 4.3.0 a OpenCL 1.1 CUDA 2.0. Obsahuje 14 multiprocesorů a paměť má 1280 MB. Počítač je osa-

zen procesorem Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz. Operační paměť počítače byla 8192 MB.

6.3 Výkonnostní testy

Výkonnostní testy byly zaměřeny na efektivitu vrhání primárních paprsků, které je možné porovnat se sledováním paprsků na GPU [AL09]. Byly rovněž měřeny stínové paprsky, které však nemohly být porovnány s [AL09].

Testování bylo prováděno v rozlišení 1024x768 pro 50 snímků z jednoho reprezentativního pohledu; průměr je znázorněn na nadcházejících grafech. Pro každý pixel byl vrhnut jeden primární paprsek, což je dohromady 786432 paprsků.

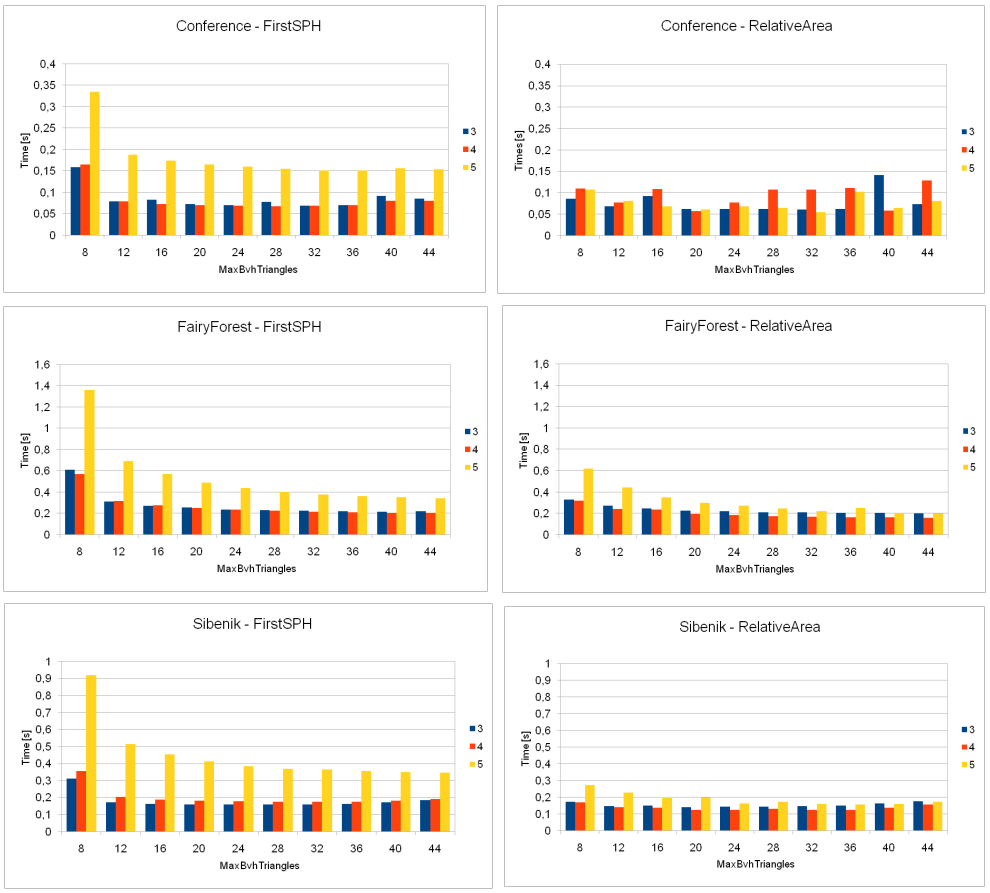
6.3.1 Efektivita implementace v závislosti na základních parametrech

Efektivita postavené hierarchie párů je závislá na různých parametrech – na stavbě hierarchií obálek (BVH) a obrazu (SPH), kritériích dělení hierarchie párů (QPH) a bodě, kdy je ukončeno dělení hierarchie párů (QPH).

K stavbě BVH byla použita Surface-area heuristika s maximálním počtem trojúhelníku v listu. SPH byla stavěna do zadané maximální hloubky. Scény byly testovány pro dvě kritéria dělení QPH popsaná v kap. 4.4.1. První kritérium (*FirstSPH*) je dělení v SPH, dokud se nedosáhne listů SPH. Poté následuje dělení v BVH. Druhé kritérium (*RelativeArea*) dělí na základě porovnání relativního povrchu obálky s relativním povrchem obdélníku. Dělení QPH je ukončeno, jakmile se dosáhne k listům obou párovaných hierarchií.

Testování bylo provedeno na scénách *Sibenik*, *FairyForest* a *Conference*. Grafy na obr. **??** znázorňují závislost času na maximálním počtu trojúhelníků v listě BVH pro různé hloubky hierarchie obrazu. Měření ukazuje, že dělící kritérium *RelativeArea* dosahuje pro všechny případy rychlejšího času vykreslení.

Ve scéně *Sibenik* je nejkratším časem vykreslení snímku čas 0.122s, kdy bylo max 20 trojúhelníků v listě BVH a SPH hloubky 4. Výsledný čas je složen z části dotazování se viditelnosti párů (0.05s) a vykreslení párů (0.07s). Pro zajímavost bylo emitováno přes 30 tisíc dotazů zastínění a vrženo přes 181 miliónů paprsků na obálky BVH. Ve scéně *FairyForest* dosáhlo vykreslování nejkratšího času 0.161s, přičemž bylo maximálně 44 trojúhelníků v listě BVH a hloubkou SPH 4. Poslední testovanou scénou byla *Conference*. Při jejím testování byl naměřen nejkratší čas 0,054s pro BVH s max 32 trojúhelníky v listě a hloubkou SPH 5.



Obrázek 6.2: Závislost času na maximálním počtu trojúhelníků v listě BVH pro různé hloubky SPH. Levý sloupec kritérium dělení *FirstSPH*. Pravý sloupec kritérium dělení *RelativeArea*.

6.3.2 Náročnost jednotlivých průchodů

Nová metoda se skládá z víceprůchodového vykreslování, které opakovaně využívá rasterizační řetězec. Největší časovou zátěží je párování BVH s SPH na základě výsledků dotazů zastínění. Ostatní zobrazovací průchody jako je rasterizace scény v prvním průchodě nebo vykreslení obdélníku přes celou obrazovku v druhém a závěrečném průchodě (viz kap. 4.3) jsou zanedbatelné (řádově tisíciny sekund) a výsledný čas nijak znatelně neovlivňují.

Scene	Rendered method	MRays /s	Time [ms]
Sibenik	Vrhání paprsků pomocí OQ	6.4	122.01
	Fermi speculative while while	217.2	3.62
	Kepler dynamic fetch	120.8	6.52
	Tesla persistent packet	111.1	7.09
	Tesla persistent speculative while while	139.6	5.63
	Tesla persistent while while	134.1	5.87
	–		
FairyForest	Vrhání paprsků pomocí OQ	4.9	161.61
	Fermi speculative while while	157.8	5.01
	Kepler dynamic fetch	93.3	8.43
	Tesla persistent packet	82.2	9.56
	Tesla persistent speculative while while	105.1	7.49
	Tesla persistent while while	93.7	8.40
	–		
Conference	Vrhání paprsků pomocí OQ	14.47	54.33
	Fermi speculative while while	260.0	3.03
	Kepler dynamic fetch	151.2	5.23
	Tesla persistent packet	138.56	5.67
	Tesla persistent speculative while while	183.4	4.01
	Tesla persistent while while	170.2	4.30

Tabulka 6.1: Srovnání metody Vrhání paprsků pomocí OQ s metodami GPU ray tracingu [AL09]. Měření pro primární paprsky.

6.3.3 Srovnání nové metody s implementací vrhání paprsků na GPU

Metoda vrhání paprsků na GPU [AL09] byla představena v roce 2009 jako nejrychlejší GPU ray tracer. Článek popisuje způsoby mapování elementárních operací ray tracingu (traverzace paprsku skrz akcelerační struktury a průnik s objekty) na rozšířené SIMT/SIMD stroje. Výsledkem je implementace 5 způsobů mapování, které jsou porovnány s novou metodou vrhání paprsků pomocí OQ. Porovnání je provedeno jen pro primární paprsky. Další efekty jako je ambient occlusion nebo diffuse interreflection, jsem z časových důvodů neimplementoval.

Tabulka 6.1 porovnává vrhání paprsků pomocí OQ s implementacemi vrhání paprsků na GPU. Vrhání paprsků na GPU dosahuje podstatně kratších časů k vykreslení scény. Ve scéně Sibenik je rozdíl 20 až 40 násobný. Ve scéně FairyForest 20 až 30 násobný a ve scéně Conference 10 až 18 násobný oproti vrhání paprsků pomocí OQ. Z toho můžeme usuzovat, že s rozsáhlostí scény bude klesat i časový rozdíl mezi porovnávanými metodami.

Scene	MRays/s	Time [s]
Sibenik (#75283)	5.4	0.14
FairyForest (#174117)	4.1	0.19
Conference (#331179)	15.5	0,05

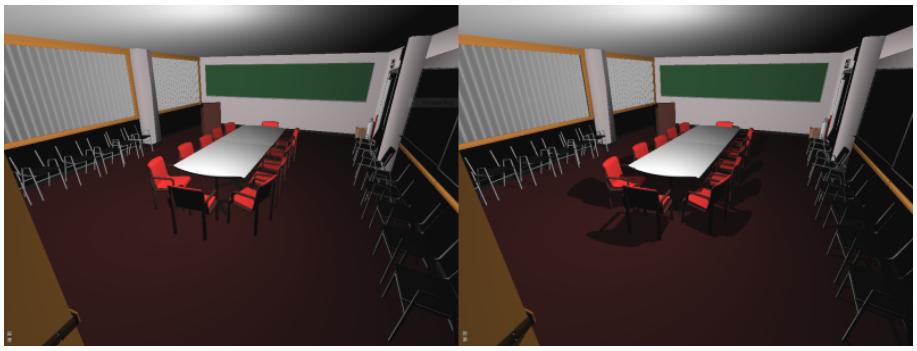
Tabulka 6.2: Nejkratší naměřený čas na výpočet ostrých stínů pro tři rozsáhlejší scény.

6.4 Výpočet různých osvětlovacích efektů

Kromě primárních paprsků byly implementovány další typy paprsků: stínové paprsky, zrcadlové paprsky a paprsky lomu světla.

6.4.1 Výpočet stínů

Nová metoda podporuje vykreslení ostrých stínů, které vznikají na odlehlých místech od bodového světla (viz obr. 6.3)



Obrázek 6.3: Scéna Conference bez stínů (vlevo) a se stíny (vpravo).

Měření byla soustředěna na čas vykreslení snímku a průměrný počet vrhnutých paprsků (v milionech) za sekundu. Výsledky měření jsou znázorněny na obr. 6.4, přičemž nejlepší výsledky jsou zaznamenány v tabulce 6.2. Nejlépe dopadlo testování pro scénu Conference, která je z testovaných scén nejrozsáhlejší, ale výpočet stínů je pro ni nejrychlejší.

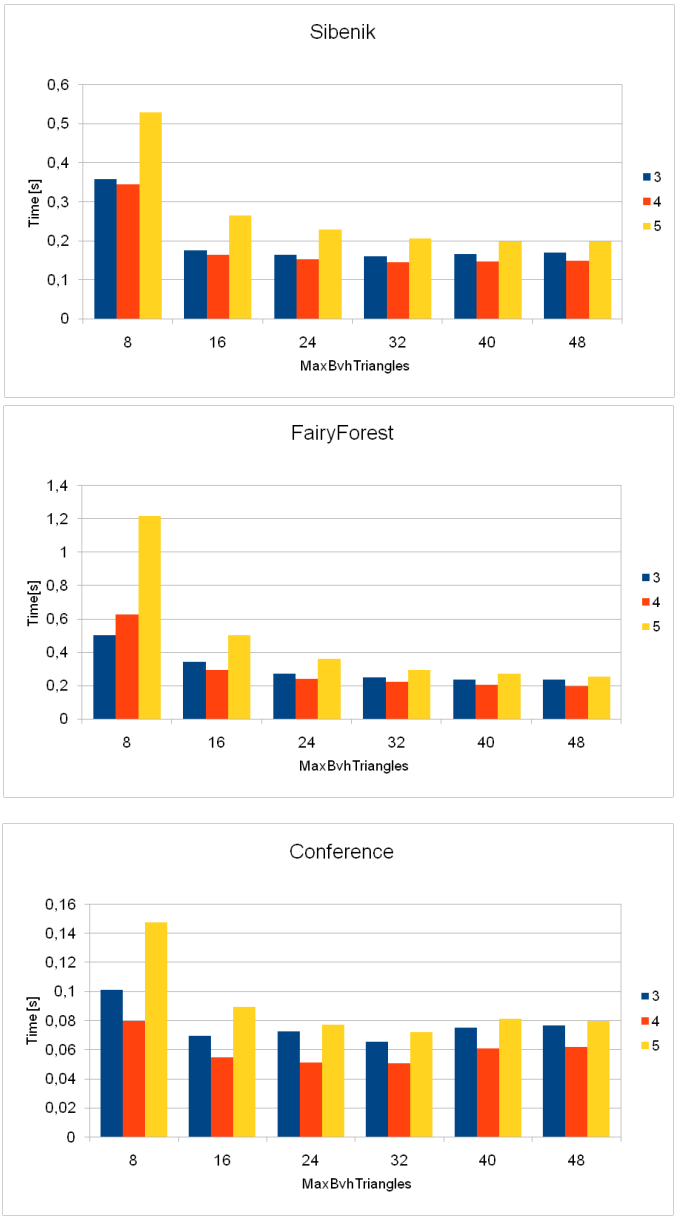
6.4.2 Zrcadlový odraz a lom světla

Výpočet zrcadlového odrazu resp. lomu světla se projeví pouze ve scénách s lesklými resp. poloprůhlednými povrchy. Scény FairyForest, Sibenik a Conference jsou pro tyto účely nevhodné. Obsahují pouze lesklé povrchy, jejichž

konstanty jsou pro ray tracing špatně nastavené. Vytvořil jsem proto jednoduchou scénu *FriendlyScene*, na které je možné ukázat odlesky a lom světla skrz skleněný povrch. Měření času nebylo pro tuto scénu provedeno, jsou jen ukázány výsledky rekurzivního sledování paprsků.

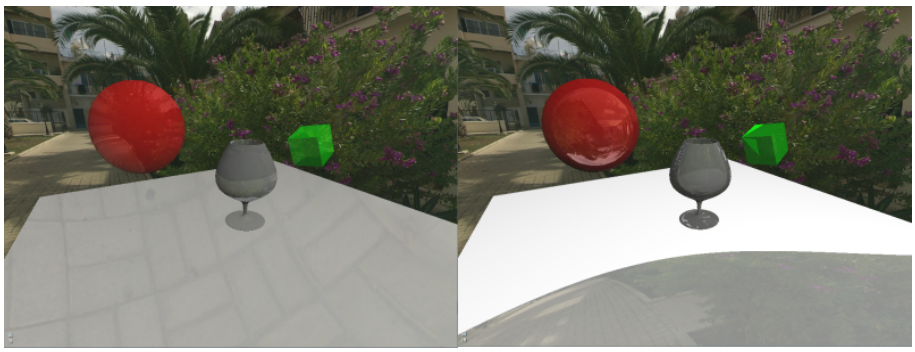
Výsledek sledování zrcadlového paprsku je ukázán na obr. 6.5.

Při sledování paprsku lomu rozlišujeme dva případy: paprsek protíná jen přední stěnu objektu a paprsek protíná obě stěny objekty (viz obr. 6.6). V druhém případě vznikají zatím artefakty, které se mi nepodařilo odstranit.



Obrázek 6.4: Výpočet stínů. Závislost času na maximálním počtu trojúhelníku v listě BVH pro různé hloubky SPH.

Obrázek 6.5: Výsledek sledování zrcadlového paprsku.



Obrázek 6.6: Láme-li se paprsek pouze při protnutí přední stěny (vlevo) a při protnutí přední i zadní stěny objektu (vpravo.

Kapitola 7

Závěr

V této práci byla popsána a implementována nová metoda výpočtu efektů globálního osvětlení v rasterizačním řetězci, která k urychlení výpočtů osvětlení využívá dotazy zastínění. Metoda byla testována na třech rozsáhlejších scénách a porovnána s existující implementací GPU ray tracingu [AL09].

Metoda ukazuje nový způsob vrhání paprsků v rasterizačním řetězci. Na základě dotazů zastínění propojuje hierarchii obálek s hierarchií paprsků v prostoru obrazu, přičemž k zrychlení využívá časové koherence po sobě jdoucích snímků. Metoda umožňuje rekurzivní sledování paprsků, které je implementováno pro zrcadlové paprsky a paprsky lomu. Metoda rovněž představuje zobecnění stínových technik, kdy dotazy zastínění identifikují oblasti, kde je obraz potencionálně zastíněn bez nutnosti konstrukce stínového objemu.

Nová metoda je v porovnání s GPU ray tracingem [AL09] časově mnohem náročnější. Při vrhání primárních paprsků dosahuje 10krát až 30krát delších časů k vykreslení snímku scény. Globální efekty – Ambient Occlusion a Diffuse Interreflection – jsem z důvodu dlouhého vývoje neimplementoval a neporovnal.

Implementace nové metody může být v budoucnu rozšířena o další globální efekty a optimalizace. Značné zrychlení pravděpodobně přinese vykreslení geometrie po skupinách, přičemž všechna geometrie náležící k jedné množině párů může být efektivně shromažďována a vykreslena jako jediný celek.

Literatura

[AL09] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 145–149, New York, NY, USA, 2009. ACM.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph.*, 11(2):242–248, July 1977.

[dG06] Bram de Greve. Reflections and refractions in ray tracing. `http://graphics.stanford.edu/courses/cs148-10-summer/docs/2006--degreve--reflection_refraction.pdf`, 2006.

[DWS+88] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. *SIGGRAPH Comput. Graph.*, 22(4):21–30, June 1988.

[FW79] J. D. Foley and Turner Whitted. An improved illumination model for shaded display, 1979.

[LP09] Nan Liu and Ming-Yong Pang. Shadow mapping algorithms: A complete survey. In *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*, pages 1–5, Jan 2009.

[McG11] Morgan McGuire. Computer graphics archive, August 2011.

[PF05] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[RAH07] David Roger, Ulf Assarsson, and Nicolas Holzschuch. Whitted ray-tracing for dynamic scenes using a ray-space hierarchy on the gpu. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, EGSR'07, pages 99–110, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[SAGC+12] Thales Luis Sabino, Paulo Andrade, Esteban Walter Gonzales Clua, Anselmo Montenegro, and Paulo Pagliosa. A hybrid gpu rasterized and ray traced rendering pipeline for real time rendering of per pixel effects. In *Proceedings of the 11th International Conference on Entertainment Computing*, pages 292–305. Springer-Verlag, 2012.

[SFD09] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 7–13, New York, NY, USA, 2009. ACM.

[Web04] Reflections and refractions in ray tracing. `http://www.flipcode.com/archives/reflection_transmission.pdf`, stav k dubnu 2014, 2004.

[Web07] The cg tutorial. `http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter09.html`, stav k březnu 2013, 2007.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, August 1978.

[ZB11] A. Zerari and M.C. Babahenini. Shadow volume in real-time rendering. In *Communications, Computing and Control Applications (CCCA), 2011 International Conference on*, pages 1–6, March 2011.

[ŽBSF04] Žára Beneš Sochor Felkel. *Moderní počítačová grafika*. Computer Press, 2004.

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

GPU Graphical Processor Unit

CPU Computer Processor Unit

BVH Bounding volume hierarchy

SPH Screen patch hierarchy

QPH QueryPair hierarchy

AABB Axis aligned bounding box

SAH Surface-area heuristic

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF