ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Navrátil** Jméno: **Michal** Osobní číslo: **420961**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Optimalizace akceleračních datových struktur pro fotorealistické zobrazování

Název diplomové práce anglicky:

Optimizing Acceleration Data Structures for Photorealistic Rendering

Pokyny pro vypracování:

Prostudujte existující metody pro akceleraci metod založených na sledování paprsků. Soustřeďte se na metody stavby a traverzace hierarchií obalových těles (BVH) a proveďte jejich rešerši. Zmapujte implementaci BVH v otevřeném softwaru Physically Based rendering Toolkit (PBRT). Rozšiřte stávající implementaci o podporu BVH s vyšší aritou (4-8). Realizujte pohledově závislou optimalizaci BVH založenou na kompaktaci podstromů a optimalizaci zaměřenou na rychlejší zpracování stínových paprsků. Proveďte důkladnou analýzu vytvořené implementace na nejméně pěti testovacích scénách a vyhodnoťte přínos vytvořených optimalizací.

Seznam doporučené literatury:

[1] Pharr, Matt, Humphreys, Greg, and Hanrahan, Pat. Physically Based Rendering 3rd edition. Morgan Kaufmann, 2016.
[2] Gu, Yan, Yong He, and Guy E. Blelloch. Ray Specialized Contraction on Bounding Volume Hierarchies. Computer Graphics Forum. Vol. 34. No. 7. 2015.
[3] Nah, J.-H. and Manocha, D. SATO: Surface Area Traversal Order for Shadow Ray Tracing. Computer Graphics Forum, 33: 167-177. 2014.
[4] Hendrich, J., Meister, D. and Bittner, J. Parallel BVH Construction using Progressive Hierarchical Refinement. Computer Graphics Forum, 36: 487-494, 2017.
[5] Shinji Ogaki, Alexandre Derouet-Jourdan, An N-ary BVH Child Node Sorting Technique for Occlusion Tests, Journal of Computer Graphics Techniques (JCGT), vol. 5, no. 2, 22-37, 2016.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Bittner, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **15.02.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

doc. Ing. Jiří Bittner, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

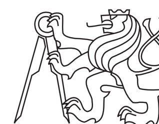
.

Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Diplomová práce

Optimalizace akceleračních datových struktur pro fotorealistické zobrazování

Bc. Michal Navrátil

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

23. května 2018

Poděkování

Rád bych zde poděkoval doc. Jiřímu Bittnerovi za ochotu, odborné rady a laskavost při vedení této práce. Dále bych rád poděkoval své rodině za jejich 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. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 23. května 2018 ....

České vysoké učení technické v Praze
Fakulta elektrotechnická
© 2018 Michal Navrátil. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě elektrotechnické. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Navrátil, Michal. *Optimalizace akceleračních datových struktur pro fotorealistické zobrazování.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2018.

Abstrakt

Práce pojednává o metodách optimalizace akceleračních datových struktur hierarchie obálek a jejich použití při sledování paprsku. Na základě analýzy použitého softwaru PBRT je navržena množina metod kombinující pohledově závislé optimalizace, nebo optimalizace podle geometrie objektů ve scéně společně s nově navrženou metodou průchodu hierarchie s využitím nahlížecí tabulky. Navržené metody jsou implementovány v jazyce C++ za použití softwaru PBRT. Výsledkem práce je snížení počtu prováděných testů průsečíku s paprskem až o 54 % a celkové snížení potřebného času syntézy obrazu v průměru o 23 %.

Klíčová slova sledování paprsku, hierarchie obalových těles, distribuce paprsků, PBRT, počítačová grafika, optimalizace datových struktur

Abstract

This thesis discusses optimization methods of bounding volume hierarchy and their usage in ray tracing algorithms. Based on the analysis of the used PBRT software, a number of methods combining a view dependent optimization or optimization based on the scene geometry are proposed together with the newly designed hierarchy traversation technique using pre-calculated lookup table. The proposed methods are implemented in C++ language with the PBRT

software. The result of this work is reduction of ray intersection tests with up to 54 % decrease and reduction in total rendering time by an average of 23 %.

Keywords ray tracing, bounding volume hierarchy, ray distribution, PBRT, computer graphics, optimization of data structures

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza a návrh **5**
2.1 Metody sledování paprsku 5
2.2 Akcelerační datové struktury 10
2.3 Hierarchie obalových těles 11
2.4 PBRT 14
2.5 Detekce úzkého hrdla výpočtu v PBRT 19

3 Realizace **25**
3.1 BVH s vyšší aritou 25
3.2 Optimalizace podle plochy 31
3.3 Pohledově závislá optimalizace 32
3.4 Určení pořadí průchodu pro primární a odražené paprsky . . . 38
3.5 Výsledky 45
3.6 Implementace a změny v PBRT 56

Závěr **59**

Literatura **61**

A Seznam použitých zkratek **63**

B Obsah přiloženého CD **65**

Seznam obrázků

1.1 Ukázka vykreslené scény s komplexním vzorem tvořených kaustik. Scéna byla vykreslena za použití metody syntézy obrazu BDPT s 8192 vzorky na pixel. 4

2.1 Princip algoritmu podle Whitteda získávající příchozí světlo pouze za pomoci přímého osvětlení, perfektního lomu a odrazu. 8

2.2 Ukázka náročné scény pro použití metody sledování cest. Ve scéně se nachází jedno plošné světlo osvětlující malý prostor na stropě. Cesty přispívající k osvětlení (příklad paprsku reprezentovaného čárkovaně) tak mohou být pouze ty, které zasáhnou zvýrazněný osvětlený prostor, nebo vnitřní část ohraničující světlo. 10

2.3 Hierarchie abstraktní třídy *Primitive* v PBRT. 15

2.4 Ukázka přestavby binární hierarchie ze stromové struktury do lineárního pole (převzato z [PJH16]). 16

2.5 Nejdelší fáze výpočtu podle metody syntézy obrazu s poměrem času k celkové době běhu programu. 22

2.6 Poměr celkového času dotazováním do akcelerační struktury při hledání nejbližšího průsečíku s paprskem, nebo dotazy viditelnosti označené příponou *P*. Testy označené *Accelerator* udávají testování průsečíku paprsku s obalovým tělesem během traverzace BVH, zbylé metody pak test průsečíku s konkrétním primitivem scény. . 23

3.1 Postupná kontrakce dvou vnitřních uzlů s důsledkem redukce dvou testů průchodu pro paprsky traverzujících skrze oba uzly. 26

3.2 Ukázka reprezentativních směrů k předpočítání pořadí průchodu synů při traverzaci BVH. Pro 4 a 13 směrů paprsků společně se směry symetrie při průchodu v opačným pořadí v modrém poli. 40

3.3 Rozložení uzlů s příslušnými reprezentativními grafy. Synové uzlu leží v listech s klíčem představujícím jejich pozici (ofset v poli uzlů hierarchie od pozice prvního syna) a použité dělící osy v vnitřních uzlech stromu . . . 42
3.4 Ukázka výstupů pro scény *Conference room*, *Buddha* a *Crown* za použití 256 vzorků na pixel a metodou syntézy obrazu *path*. . . . 46
3.5 Ukázka výstupů pro scény *San Miguel* a *Ecosystem* za použití 256 vzorků na pixel a metodou syntézy obrazu *path*. . . . 47
3.6 Porovnání metod určujících pořadí traverzace řazením a nahlížecí tabulkou oproti původní implementaci nad binárním stromem hierarchie z PBRT. V levém grafu je poměr celkového času výpočtu každé z metod. V pravém poměr počtu provedených testů průsečíku paprsku s obalovým tělesem a primitivem v průběhu dotazů k nalezení nejbližšího průsečíku. . . . 51
3.7 Poměr celkového času výpočtu metod *1Phase* a *2Phase* za použití různého počtu vzorků na pixel. . . . 52
3.8 Poměr celkového času výpočtu metod *SAO* a *1Phase* s použitím jedné nahlížecí tabulky oproti použití dvojice nahlížecích tabulek k určení pořadí průchodu synů při dotazu o nejbližší průsečík s paprskem. . . . 53
3.9 Porovnání metod určujících pořadí traverzace při hledání nejbližšího průsečíku za pomoci nahlížecí tabulky. Varianta *dirLUT* využívá předpočítaná pořadí pro 8 směrů paprsku, *LUT* představuje předpočítaná pořadí z konstrukce s jednou nahlížecí tabulkou. V levém grafu je poměr celkového času výpočtu, v pravém poměr počtu provedených testů průsečíku paprsku s obalovým tělesem a primitivem. . . . 54
3.10 Porovnání celkového času výpočtu metod *SAO* a *1Phase* vůči neupravenému běhu programu s metodou syntézy obrazu *path* a stavbou BVH pomocí *SAH*. Naměřená data jsou porovnána pro různé počty vzorků na pixel s hodnotami z alespoň čtyř různých měření pro každou z použitých metod. . . . 55
3.11 Porovnání počtu provedených testů průsečíku s paprskem pro metody *SAO* a *1Phase* za použití 128 vzorků na pixel, metody syntézy *path* a stavbou BVH metodou *SAH*. Poměry jsou uvedeny vůči původní implementaci PBRT. . . . 56
3.12 Porovnání relativního času stráveného ve fázích hledání průsečíku s paprskem pro metody *SAO* a *1Phase* za použití 128 vzorků na pixel, metody syntézy *path* a stavbou BVH metodou *SAH*. Poměry jsou uvedeny vůči původní implementaci PBRT. . . . 57

Seznam tabulek

2.1 Přehled scén s odpovídající BVH postavenou podle zvolené metody konstrukce. 18

2.2 Porovnání výkonu testovaných struktur BVH pro různé metody stavby a algoritmy vykreslování. Pro výpočet SAH cost jsme použili $c_T = 3$ a $c_I = 2$. Hodnoty *diff* označují testy během dotazů na nejbližší průsečík a *shadow* dotazy zastínění. Procenta uvádí poměr času stráveného jednotlivými testy vůči celkovému času syntézy obrazu. 20

2.3 Porovnání výkonu testovaných struktur BVH pro různé metody stavby a algoritmy vykreslování. Pro výpočet SAH cost jsme použili $c_T = 3$ a $c_I = 2$. Hodnoty *diff* označují testy během dotazů na nejbližší průsečík a *shadow* dotazy zastínění. Procenta uvádí poměr času stráveného jednotlivými testy vůči celkovému času syntézy obrazu. 21

3.1 Pořadí průchodu pro syny vnitřních uzlů hierarchie s ohledem na jejich rozložení a průchod paprsku vůči použitým dělícím osám. . . . 43

3.2 Porovnání metod *sort*, *nearest* a LUT určujících pořadí traverzace synů pro primární a odražené paprsky. Pro stínové paprsky je pořadí u všech metod určeno stejně, seřazením podle velikosti obalových těles synů. Naměřená data jsou zprůměrována z 16 měření a 4 metod syntézy obrazu s 64 vzorky na pixel. Počty testů jsou v milionech s *diff* udávající počet hledání nejbližšího průsečíku a *shadow* počet testů zastínění. Poměry srovnávají metody oproti původní implementaci z PBRT. 50

3.3 Časy strávené ve fázi hledání průsečíku během traverzace BVH a testování jednotlivých primitiv. Naměřená data jsou pro 128 vzorků na pixel. 56

3.4 Detailní výsledky měření pro různé metody optimalizace společně s původní variantou programu. Počty testů průsečíku paprsku s obálkami i primitivy jsou uvedeny v milionech, množství uzlů podléhajících kontrakci v tisících s poměrem vůči celkovému počtu uzlů BVH. 58

Úvod

Syntéza obrazu (renderování) je tvorba reálného obrazu ze vstupních dat nejčastěji 3D scény. Uplatnění nachází například během vytváření animací pro počítačové hry, filmy a při zobrazování různých simulací fyzikálních jevů. Získává také výrazné pozornosti v průmyslu a architektuře vizualizací prototypů výrobků a staveb. Zatímco i drobné změny designu mohou přinést výrazné komplikace v jejich aplikaci na fyzický objekt, využitím počítačové grafiky lze změny vizualizovat obvykle za zlomek času i nákladů.

K docílení fotorealistického výsledku je zapotřebí značný výpočetní výkon a celý proces může pro jeden obraz trvat i několik hodin, během kterého se do scény vrhá běžně i několik miliard paprsků. Právě sledované paprsky určují výsledný vzhled scény přenášením světelné energie, popřípadě zjišťují zastínění mezi dvojicí bodů.

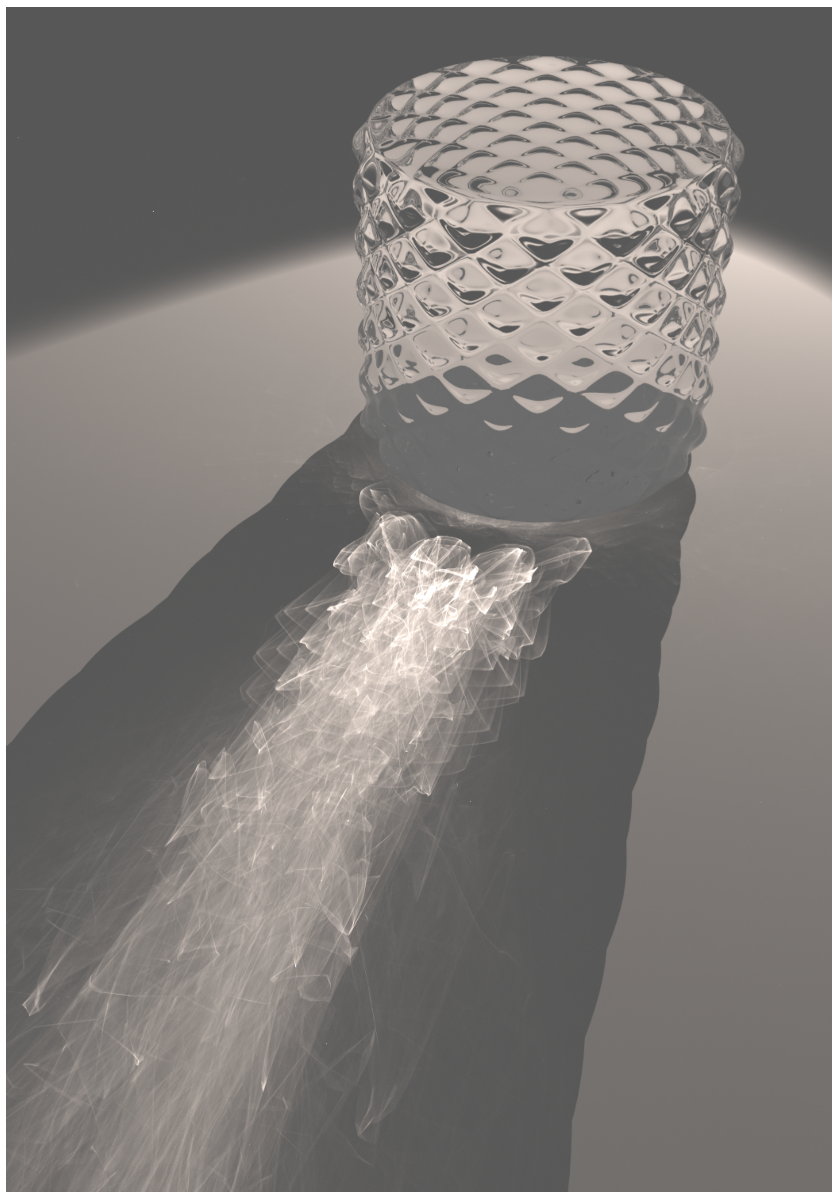
Práce pojednává o metodách optimalizace akceleračních datových struktur hierarchie obálek a jejich použití během sledování paprsku se zaměřením na implementaci v softwaru PBRT. V první části prozkoumáme existující metody softwaru, včetně detekce nejdelších fáze výpočtu, na základě kterých v kapitole 3 navrhneme a otestujeme několik možných řešení optimalizace.

Kapitola 1

Cíl práce

Syntéza obrazu je stále se rozšiřující odvětví počítačové grafiky napodobující reálný svět. S rostoucími požadavky na realističnost však stoupá i výpočetní náročnost a tedy i potřebný čas pro vykreslení každého obrazu.

Cílem této práce je urychlit proces syntézy obrazu optimalizací akcelerační datové struktury hierarchie obalových těles. Pro každý obraz je provedena její přestavba na základě výběru optimalizační metody závislé na pohledu kamery, nebo na geometrii objektů ve scéně.



Obrázek 1.1: Ukázka vykreslené scény s komplexním vzorem tvořených kaustik. Scéna byla vykreslena za použití metody syntézy obrazu BDPT s 8192 vzorky na pixel.

Kapitola 2

Analýza a návrh

V této kapitole se nejprve zabýváme teoretickém základem celé práce s popisem základních vlastností zobrazovacích softwarů společně s několika konkrétními metodami syntézy obrazu (viz sekce 2.1). Dále pak navážeme použitím akceleračních datových struktur (viz sekce 2.2) a detailnějším popisem použité hierarchie obalových těles, včetně příslušných metod její stavby (viz sekce 2.3). Následuje sekce 2.4 zaměřená na použitý software PBRT, jeho základní vlasti a postup pro měření výsledků. V závěru kapitoly nakonec provedeme analýzu naměřených dat základní verze PBRT a detekci úzkého hrdla celého výpočtu syntézy obrazu (viz sekce 2.5).

2.1 Metody sledování paprsku

Téměř všechny fotorealistické renderovací softwary jsou založené na metodách sledování paprsku, jehož princip je ve skutečnosti velice jednoduchý: spočívá ve sledování trasy paprsku světla skrze scénu, ve které paprsek naráží a interaguje s objekty prostředí. Princip této techniky má počátek ve tvorbě čoček, když v 19. století Carl Friedrich Gauß skrze ně ručně sledoval šíření paprsků. Hlavní rozdíl pro většinu metod syntézy obrazu ale je, že se neposílají paprsky ze světel jak by se očekávalo, ale směrem od pozorovatele (kamery).

Paprsek se promítne skrz pomyslný pixel v rovině obrazovky směrem do scény, ve které se snaží získat informaci o množství světelné energie, kterou danému pixelu přináší. Podstatnou částí tohoto sledování paprsku je nalezení průsečíku s jeho nejbližším tělesem. Zde na základě osvětlovacího modelu paprsek získává svou energii podle normály k tělese a jeho materiálu, popřípadě z dalších sekundárních paprsků. Kromě samotného průniku s objektem, paprsek může měnit svou světelnou energii v závislosti na opticky aktivním prostředí ve kterém se šíří, například během průchodu poloprůhledných objektů.

2.1.1 Základní vlastnosti softwarů pro realistickou syntézu obrazu

Ačkoli je možné napsat metodu sledováni paprsku mnoha způsoby, veškeré aplikace mají několik základních objektů a nezbytných událostí společných:

- Kamera: Model kamery určující pohled do scény a její zaznamenání.
- Průsečík paprsku s objektem: Přesné určení pozice průsečíku paprsku s geometrickým objektem ve scéně. Obvykle se navíc určují další vlastnosti v místě průniku, jako například povrchová normála nebo materiál.
- Světelné zdroje: Zdroje osvětlení scény, jejich pozice a způsob distribuce světelné energie ve scéně.
- Viditelnost a zastínění: Test osvětlení určité pozice specifickým světlem, tedy přispívá-li k osvětlení bodu tak, že mezi samotnou pozicí bodu a světlem neleží zastiňující objekt.
- Rozptylování na povrchu: Popis vzhledu každého objektu, jejich povrchu a také chování světla při interakci s nimi.
- Nepřímé osvětlení: Jelikož se světlo může šířit dále po odrazu či průchodu s objektem, bývá nutné pro přesné zaznamenání tohoto jevu sledovat další paprsky s počátkem v daném bodě.
- Propagace paprsku: Vliv prostředí jako je mlha a atmosféra na světlo podél paprsku. na rozdíl od vakua, kde světelná energie zůstává konstantní po celou délku paprsku.

2.1.2 Distribuce světla

Po nalezení pozice průsečíku s objektem získáváme bod k osvícení a další informace o geometrii v okolí tohoto bodu. Naším výsledným cílem je pak zjistit množství světla, které opouští tento bod ve směru pozorovatele (kamery). K tomu je třeba zjistit, kolik světelné energie dopadá na pozici daného bodu.

Hrají zde zásadní úlohu prostorové úhly, jelikož výpočet osvětlení bodu se vypočítá jako světlo přicházející skrze kouli, která tento bod obklopuje. Často navíc chceme zjisti množství světelné energie, která připadá na diferenciální oblast obklopující bod průsečíku. Za předpokladu, že má bodové světlo zářivý tok Φ a vyzařuje ve všech směrech stejně, je intenzita ozáření, tedy zářivý tok (výkon) dopadající na jednotku plochy, na jednotkové kouli obklopující světlo rovna $\Phi/(4\pi)$. Je pak zřejmé, že při srovnání dvojice různě velkých koulí, bude intenzita ozáření na větší kouli menší, jelikož je energie rozdistribuována přes větší plochu. Množství toku dopadajícího na plochu koule poloměru r proporčně odpovídá zlomku $1/r^2$. Navíc lze ukázat, že pro diferenciální plochu dA nakloněnou o úhel θ vůči vektoru spojujícího světlo a bod

na povrchu, je množství zářivého toku připadající na tuto plochu poměrově odpovídající *cosθ*. Sloučením je pak diferenciální intenzita ozáření dE (*differential irradiance*), jakožto

$$dE = \frac{\Phi cos\theta}{4\pi r^2}.$$

Scény s více světly se pak řeší jednoduše, jelikož osvětlení je lineární. Je možné proto vypočítat příspěvek přímého osvětlení pro každé světlo zvlášť a výsledek dostat jejich sečtením [PJH16].

Nepřímé osvětlení získáme na základě zobrazovací rovnice, jejíž výchozím kritériem je zachování energie, tedy zář (radiance) opouštějící bod povrchu musí být někde odražena nebo absorbována, jedná se tak o popis ustáleného stavu ve scéně. Obecně je množství světla dopadajícího do kamery z daného bodu získáno jako vyzářené světlo v daném bodě, jedná-li se o světelný zdroj, společně s odraženým světlem ze všech možných směrů. Předpokládejme pozorovaný bod x na povrchu nějakého objektu. Hledaná celková záře L opouštějící tento bod x ve směru ω_o určíme podle zobrazovací rovnice

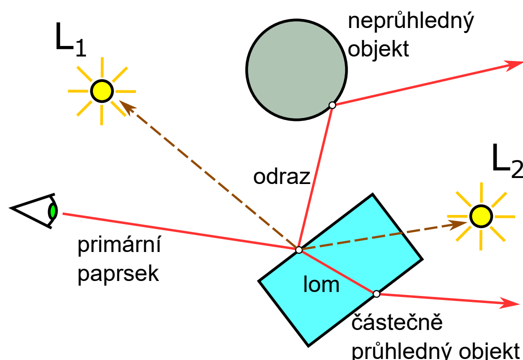
$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f(x, \omega_o, \omega_i) L_i(x, \omega_i) cos\theta_i d\omega_i, \tag{2.1}$$

kde $L_e(x, \omega_o)$ je záře vyzářená z x ve směru ω_o, L_i a ω_i přicházející záře a směr a θ_i úhel sevřený směrem ω_i a normálovým vektorem k povrchu. V uvedeném vztahu je $f(x, \omega_o, \omega_i)$ dvousměrová odrazová distribuční funkce (BRDF) s integrací přes polokouli Ω, je však možné použít i dvousměrovou rozptylovou distribuční funkci (BSDF - Bidirectional scattering distribution function) a integraci provádět pro všechny možné směry na celé kouli okolo x. Řešení tohoto integrálního vztahu analyticky je až na úplně nejmenší scény nemožné a musí se proto hledat aproximační řešení, které získáme na základě dále popsaných metod syntézy obrazu.

2.1.3 Whittedův algoritmus sledování paprsku

Metoda syntézy obrazu na základě algoritmus podle Whitteda [Whi80] zjednodušuje integrální výpočet (2.1) ignorováním příchozího světla z většiny směrů a bere v potaz pouze příspěvky $L_i(x, \omega_i)$ ze zdrojů světla a pro perfektní odraz a lom paprsku (viz obr. 2.1). Výslednou rovnicí je tak pouze suma přes malý počet směrů.

Oproti původní variantě algoritmu je použito malé, ale významné rozšíření za účelem získání aproximace pro lesklé (glossy) povrchy. Kromě perfektního odrazu a lomu je navíc sledováno větší množství rekurzivních paprsků v náhodném směru ω_i blízkému zrcadlovému odrazu. Váha jednotlivých příspěvků těchto náhodných paprsků je získána vyhodnocením BRDF $f(x, \omega_o, \omega_i)$. Výsledek této modifikace zachycuje všechny možné odrazy světla mezi objekty a vede k realistických obrazům.



Obrázek 2.1: Princip algoritmu podle Whitteda získávající příchozí světlo pouze za pomoci přímého osvětlení, perfektního lomu a odrazu.

2.1.4 Metoda přímého osvětlení

Jak již napovídá název, metoda bere v potaz pouze přímé osvětlení. Během výpočtu tak počítáme pouze světlo dopadající na stínovaný bod přímo ze zdrojů světla a ignoruje se nepřímé osvětlení z objektů, které sami nejsou světelným zdrojem s výjimkou základního zrcadlového odrazu.

S využitím této metody se běžně používají dvě strategie odhadu výstupní záře ve zvoleném místě a v daném směru. První strategie prochází všechna světla ve scéně s odpovídajícím počtem vzorků na každé z nich. Výsledná zář je pak získána sečtením všech příspěvků ze vzorků. Druhá strategie vybírá pouze jediný vzorek z náhodně zvoleného světla.

V závislosti na scéně mají oba přístupy své uplatnění. Například při vyhodnocování efektu hloubky ostrosti za použití velkého množství vzorků pro redukci nadbytečného šumu, může být vhodnější strategie získávání příspěvků z jednoho náhodného vzorku, oproti tomu při použití malého počtu vzorků na pixel může tato možnost vykazovat celkového šumu výrazně více.

2.1.5 Sledování cest

Sledování cest (Path tracing) je první obecně navržený nevychýlený algoritmus metody Monte Carlo pro přenos světla v počítačové grafice [KK86]. Algoritmus opakovaně generuje cesty paprsku, na základě rozptylové události, začínajících od kamery a končících ve zdroji světla ve scéně.

Kompaktní zápis vyhodnocované rovnice lze zapsat jako

$$L(p_0 \to p_1) = \sum_{n=1}^{\infty} P(\overline{p}_n),$$

kde p_0 je počáteční vrchol cesty na kameře, p_1 první průsečík s geometrií a $P(\overline{p}_n)$ udává množství záře rozptýlené na cestě $\overline{p}_n$ s $n+1$ vrcholy, $\overline{p}_n = p_0, p_1, \ldots, p_n$ [PJH16].

Počínaje prvním průsečíkem paprsku s geometrií ve scéně jsou postupně generovány další body na sledované cestě paprsku na základě vzorku z BSDF u poslední získané pozice a nalezeném nejbližším průsečíku v daném směru.

Možnou změnou implementace je vyhodnocování členu $P(\overline{p}_i)$ se znovupoužitím bodů z předcházející cesty $\overline{p}_{i-1}$. Až na poslední vrchol ležící na zdroji světla je znovu použito všech zbývajících $i-1$ vrcholů. Výsledkem je potřeba nalezení pouze jediného nejbližšího průsečíku pro další sledovaný paprsek, namísto i v případě celé nové cesty. Tento přístup ve výsledku snižuje kvalitu přidáním korelace příspěvků $P(\overline{p}_n)$, přesto je používán díky výraznému snížení počtu sledovaných paprsků.

Jako ukončovací kritérium je použita metoda ruské rulety, aby bylo možné přestat sledovat paprsky s menším, než minimálním příspěvkem, společně s možností omezení na maximální hloubku rekurze za pomoci vstupního parametru.

Na rozdíl od předešlých metod *direct lighting* a *whitted* může být zapotřebí i několik stovek až tisíců vzorků na pixel k docílení kvalitního výstupu bez charakteristického šumu této metody.

2.1.6 BDPT

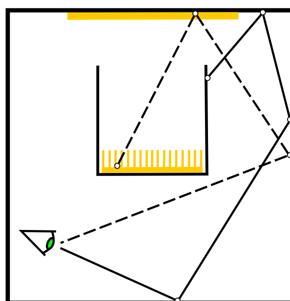
U některých scén může být sledování paprsků pouze z kamery pomocí metody sledování cest neefektivní (viz obr. 2.2), s většinou paprsků přinášejících pouze malý, nebo dokonce žádný příspěvek do celkového osvětlení scény. Příkladem podmínek takové scény je umístění jediného světla do částečně otevřeného objektu, díky kterému je většina scény osvětlena pouze nepřímo skrze malý otvor. Možným řešením takto specifických scén je konstrukce sledované cesty z kamery zároveň s druhou cestou ze zdroje světla, mezi kterými jsou posílány paprsky pro test viditelnosti. Výsledkem je dvousměrové sledování cest, tedy metoda BDPT (Bidirectional Path Tracing).

Vrcholy obou cest jsou získány podobně jako u metody *path*, tedy počínaje prvním průsečíkem s geometrií ve scéně je na základě vzorku z BSDF poslán další paprsek, jehož nejbližší průsečík tvoří další vrchol cesty a na kterém se proces opakuje. Pro cestu začínající ve světle je postup stejný, pouze je použita adjungovaná BSDF.

Výsledné cesty $\overline{p}$ přispívající k celkovému osvětlení lze získat propojením dvojice vrcholů každé z cest. Pro cestu z kamery délky t tvořenou vrcholy $p_0, p_1, \ldots, p_{t-1}$ a cestu ze světla $q_0, q_1, \ldots, q_{s-1}$ délky s, tak výsledná cesta sledovaná od světla vypadá jako

$$\overline{p} = q_0, q_1, \ldots, q_{s'-1}, p_{t'-1}, \ldots, p_0,$$

kde $s' \leq s$ a $t' \leq t$. Pokud jsou vrcholy $q_{s'-1}$ a $p_{t'-1}$ vzájemně nezastíněné, pak příspěvek cesty lze nalézt vyhodnocením BSDF na spojovaných vrcholech.



Obrázek 2.2: Ukázka náročné scény pro použití metody sledování cest. Ve scéně se nachází jedno plošné světlo osvětlující malý prostor na stropě. Cesty přispívající k osvětlení (příklad paprsku reprezentovaného čárkovaně) tak mohou být pouze ty, které zasáhnou zvýrazněný osvětlený prostor, nebo vnitřní část ohraničující světlo.

2.2 Akcelerační datové struktury

K získání světelné energie přinášené ze směru posílaného paprsku je, nehledě na použité metodě syntézy obrazu, potřeba umět určit jeho interakci s objekty scény. Jedním z nejdůležitějších testů je nalezení nejbližšího průsečíku, který nám určí první bod zásahu daným paprskem. Je zřejmé, že pro naivní implementaci by bylo zapotřebí projít a otestovat všechny objekty ve scéně, čímž by se aplikace stala i pro středně velké scény nepoužitelnou. Z důvodu zrychlení celého testu se proto používají akcelerační datové struktury.

Cílem těchto struktur je především provádět test průsečíku pouze u objektů, které leží blízko sledovaného paprsku, resp. schopnost rozhodnou které objekty jistě zasaženy nebudou a test s nimi ani neprovádět. Důraz je tedy kladen na získání výsledku v co nejkratším čase, musí být ale stejný, jako u testování všech objektů samostatně. Právě hledání nejbližšího průsečíku, nebo jen existence průsečíku mezi dvěma body pro testy zastíní, je jedna z nejčastěji volaných funkcí programu, celková efektivita této části je proto velice důležitá a výrazně se projeví na celkovém času syntézy obrazu.

Akcelerační datové struktury lze rozdělit na základě jejich přístupu na:

- Prostorové dělení: Typ akceleračních datových struktur dělící prostor scény na nepřekrývající se oblasti. Každá z výsledných oblastí si pamatuje v ní ležící objekty scény a v případě vstupu paprsku jsou obsažené objekty testovány na průsečík. Objekty navíc mohou ležet ve více jak jedné oblasti, v závislosti na implementaci tak může pro jeden paprsek docházet zbytečně k opakovanému testování stejného objektu. Příklady tohoto typu struktur jsou *k*-d strom, octree (oktalový strom) a uniformní mřížka.

- Objektové dělení: Hierarchický přístup dělení objektů ve scéně na blízké shluky. Části hierarchie se mohou překrývat, objekty se v ní ale nacházejí vždy pouze jednou. Typickým reprezentantem tohoto způsobu dělení je hierarchie obalových těles, kterou v této práci využíváme a budeme se s ní blíže zabývat v následující sekci 2.3.

2.3 Hierarchie obalových těles

BVH je, jak již název napovídá, založeno na stavbě hierarchie obálek využívající obálky těles ve scéně. Samotný tvar obálky může být vcelku rozmanitý (koule, kvádr, rovnoběžnostěn), princip ale zůstává stejný - neprotne-li paprsek obálku, tak je jisté že nezasáhl ani tělesa v ní obsažená. Nejčastěji používaným druhem obálky je osově zarovnaný kvádr, neboli *AABB* (Axis Aligned Bounding Box), s kterými je samotný test na průsečík velice rychlý a zabírá oproti komplikovanějším strukturám relativně málo paměti - ve $3D$ celkem 24B.

Stavbu BVH je možné provádět metodou zdola-nahoru, jak ukázali např. Walter a kol. [WBKP08] za použití navrženého lokálně-řazeného algoritmu. Většina používaných metod ale využívá postupu shora-dolů, jehož princip stavby hierarchie je založen na metodě objektového dělení, která progresivně dělí objekty scény na menší disjunktní množiny objektů. Veškerá primitiva jsou uložena v listech hierarchie a každý uzel obsahuje minimální obálku obsahující veškerá primitiva pod ním. Jak již bylo zmíněno, základní vlastností tohoto dělení je, že každé primitivum se objeví v celé hierarchii právě jednou. Zároveň se jedná o úplný binární strom, tedy každý vnitřní uzel má právě dva syny. Listy dále mohou obsahovat více primitiv.

2.3.1 Postup stavby BVH

Základní metoda stavby, běžně považovaná za tzv. zlatý standard, je metoda *SAH* [GS87], podle *Surface Area Heuristic*, blíže popsaná v podsekci 2.3.4. Alternativou je metoda *HLBVH* s využitím řazení primitiv podél Mortony křivky [GPM11], pro kterou stavba může být podstatně rychlejší, obecně ale postavená struktura není tak efektivní na vyhodnocování dotazů do ní. Další dvě metody jsou založeny na prostorovém a objektovém mediánu, pojmenované *Middle* a *Equal*. Jsou sice méně výpočetně náročné na stavbu, obecně ale stavějí méně efektivní stromy hierarchie.

Až na metodu *HLBVH* se stavba provádí pomocí rekurzivní funkce rozdělující primitiva mezi levého a pravého potomka uzlu dokud není dosaženo ukončující kritérium, tedy namá-li obálka centroidů (středů obálek) zbývajících primitiv nulovou plochu, nebo nevejdou-li se primitiva do listu s definovanou velikostí. Dělení začíná v kořeni obsahujícím veškerá primitiva scény. Osa ve které se bude dělení provádět je vybrána ta z os, ve které má projekce

centroidů na jednotlivé osy největší rozpětí, samotná pozice pomyslné dělící roviny pak záleží na vybrané metodě stavby BVH.

2.3.2 Prostorový medián

Jednoduchá dělící metoda označená obvykle jako *Middle*, spočte střed obálky centroidů podél dělící osy a rozdělí primitiva do pravého či levého syna v závislosti na pozici centroidu ve vybrané ose vůči spočtenému středu. Pokud se během této metody nepodaří rozdělit primitiva do dvou skupin, jednou z možností je vytvořit list se všemi zbylými primitivy, jehož průchod je ale pro velký počet značně neefektivní, jelikož dotazováním na nejbližší průsečík je třeba otestovat všechna z nich. Alternativou je v takovém případě provést opětovné dělení pomocí objektového mediánu, která jistě zbylá primitiva dále rozdělí.

2.3.3 Objektový medián

Metoda stavby nazvaná *Equal*, nebo také jako *EqualCounts*. Rozdělí primitiva do dvou stejně velkých množin, kde v jedné mají všechny centroidy menší pozici ve vybrané dělící ose než ve množině druhé. Každé z těchto množin je vytvořen uzel hierarchie umístěný jako levý, resp. pravý syn uzlu obalujícího celou množinu vstupních primitiv. na oba syny se dále volá rekurzivně stejný postup, dokud není docíleno ukončující kritérium a uzel zůstane listem obsahujícím vstupní primitiva.

2.3.4 SAH

Dělení využívá již zmíněné metody SAH, jejíž cílem je nalezení optimálního dělení primitiv do levého a pravého podstromu, popřípadě určit zda-li by bylo výhodnější ze zbývajících primitiv vytvořit list. Při dotazování na nejbližší průsečík paprsku s primitivem je pro list třeba otestovat všechna jeho primitiva, zatímco pro vnitřní uzly je testování jednotlivých primitiv v podstromech ovlivněno pravděpodobností zásahu synů a pokračování traverzace. Při určování rozdělení tak je třeba brát v potaz cenu tetování průsečíku se všemi primitivy, nebo zvýšení ceny o test s vnitřním uzlem, ale pravděpodobností předčasného ukončení před samotným testováním rozdělených primitiv.

K docílení co nejlepšího dělení na tomto principu je třeba minimalizovat cenový model, pro který se cena pro list spočte jako

$$\sum_{i=1}^{N} t_{isect}(i),$$

kde $t_{isect}(i)$ je čas potřebný pro výpočet průsečíku s i-tým primitivem, nebo pro rozdělení primitiv do množin A a B jako

$$c(A, B) = t_{trav} + p_A \sum_{i=1}^{N_A} t_{isect}(a_i) + p_B \sum_{i=1}^{N_B} t_{isect}(b_i),$$

kde t_{trav} je čas potřebný pro traverzaci vnitřním uzlem, p_A a p_B pravděpodobnosti pokračování paprsku příslušným synem a N_A a N_B velikosti odpovídajících množin. k usnadnění výpočtu je běžně učiněn zjednodušující předpoklad, že $t_{isect}(i)$ je stejný pro všechna primitiva a jeho cena pro výpočet nastavena např. na 8 a cena traverzace na 1.

Volba rozdělení primitiv do množin A a B je kritická s přímým vlivem na cenu podstromů počtem primitiv a pravděpodobností p_A a p_B plochou příslušných obálek. Aby nebylo třeba testovat všechna možná rozdělení, používá se metoda zvaná *binning* [HHS06]. Primitiva jsou umístěna do konečného počtu disjunktních, vzájemně navazujících přihrádek na zvolené dělící ose a výpočet rozdělení je proveden na jejich základě jakožto shluků primitiv. Pro velký počet primitiv se tak výrazně redukuje potřebný počet testovaných rozdělení. Primitiva spadají do přihrádek na základě pozice jejich centroidu a reprezentativní obalové těleso přihrádky je vypočteno sjednocením obálek obsažených primitiv.

2.3.5 HLBVH

Tato metoda vznikla díky problémům paralelizace výše zmíněných metod, kde je třeba obvykle sekvenčně postavit několik prvních úrovní hierarchie, než je možné využít výpočetní sílu paralelizace a jednotlivá vlákna spustit na předpřipravené podstromy.

Lineární hierarchie obalových těles (LBVH) řeší tento problém pomyslnou záměnou konstrukce na řadící algoritmus. Jelikož ale není jednoznačná funkce pro seřazení vícerozměrných dat, používá se zde *Mortonův kód* (Mortonova Z-křivka), jenž mapuje blízké body v n rozměrech na blízké body podél $1D$ přímky. Po seřazení primitiv tak prostorově blízké shluky leží na sousedních pozicích v seřazeném poli.

Konečná konstrukce HLBVH (*hierarchical linear bounding volume hierarchy*) [GPM11] nejprve vytvoří ze shluků (množiny s podobným Mortonovým kódem) spodní část výsledné hierarchie jakožto samostatné podstromy. Ty jsou pak spojovány ve vyšších úrovních hierarchie metodou *SAH* do výsledného BVH.

Metoda *HLBVH* počítá během výpočtu pouze s centroidy řazených primitiv, což je kritické pro výkon který HLBVH nabízí, na druhou stranu ale vůbec nepočítá s velikostí jednotlivých primitiv v prostoru, což znamená, že ani výsledná struktura nebere v potaz variace ve velikostech jednotlivých primitiv, na rozdíl například od stavby metodou SAH.

2.4 PBRT

PBRT pracuje na základě technik objektově-orientovaného programování. Pro důležité entity scény jsou vytvořeny abstraktní třídy tak, že každá entita definuje své rozhraní a každý objekt dané entity musí toto rozhraní implementovat (např. abstraktní třída *Shape* definuje rozhraní s funkcí pro test průsečíku s ním, tedy každý geometrický objekt, jakožto dědící z této třídy, musí tuto funkci také podporovat). Účelem tohoto dělení je, že převážná část práce s PBRT je tvořena interakcí s dostupným rozhraním těchto abstraktních tříd. Díky tomu není nutné se zabývat, kterým konkrétním typem je daný objekt. Systém tak navíc umožňuje vcelku snadnou možnost rozšíření o další typy, pouhým rozšířením o dědičnou třídu podporující požadované rozhraní.

Základní abstraktní třídy, které systém využívá jsou:

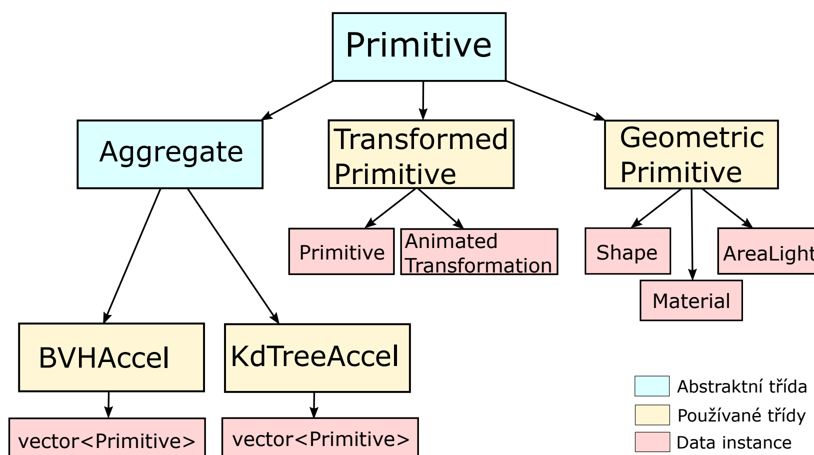
- Shape
- Primitive
- Camera
- Sampler
- Filter
- Material
- Texture
- Medium
- Light
- Integrator

Každý geometrický objekt ve scéně je reprezentován třídou *Primitive*, která v sobě obsahuje *Shape* specifikující její tvar a $Material$ specifikující její vzhled. Veškerá geometrická primitiva jsou obsaženy v jednom souhrnném *Primitivu* (*Aggregate* primitivum), to je speciální v tom, že v sobě drží reference na další primitiva, ale udržuje stejné rozhraní a nejeví se proto navenek nijak odlišně. Aby ale nedocházelo k nadbytečnému počtu testů na dotaz o průsečík s paprskem, udržuje obsažená primitiva v akcelerační struktuře (pro nás zmíněné BVH), díky niž se počet dotazů výrazně redukuje. Příklad použití této hierarchie primitiv viz obr. 2.3.

Samotný proces syntézy obrazu PBRT lze koncepčně rozdělit na dvě fáze:

1. Zpracování vstupního souboru

   Samotný soubor specifikuje jednotlivé tvary tvořící scénu, jejich materiál, světla co je osvětlují a umístění kamery. Zároveň se zde definují veškeré parametry volaných funkcí napříč systémem.



Obrázek 2.3: Hierarchie abstraktní třídy *Primitive* v PBRT.

2. Vykreslení scény

   Hlavní vykreslovací smyčka procesu. Během níž se posílá význačné množství paprsků do scény pro určení množství světla dopadajícího na virtuální film (projekční rovinu).

2.4.1 Hierarchie obalových těles v PBRT

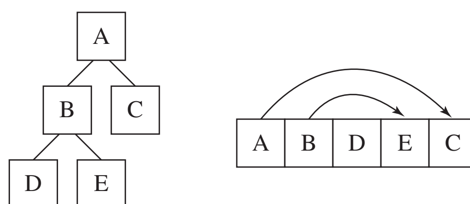
Stavba hierarchie obalových těles v PBRT je prováděna na základě teoretického základu uvedeného v předešlé sekci a podporuje všechny čtyři uvedené metody stavby, včetně drobných zlepšení, jako např. změna dělící metody na objektový medián v případě neúspěšného rozdělení primitiv stavbou využívající prostorový medián (viz podsekce 2.3.2). Výraznější specifikaci je třeba uvést pouze pro metodu *SAH* (viz podsekce 2.3.4), pro kterou je během celé stavby počet použitých přihrádek metodou *binning* konstantní (12). Důsledkem je výrazné zrychlení výpočtu oproti postupu testujícího všechna možná rozdělení s výslednou hierarchií, která se ukazuje téměř stejně efektivní [PJH16]. Metoda se navíc pro malý počet primitiv (méně, nebo rovno čtyřem) přepíná do dělení podle objektového mediánu, jelikož výpočetní náročnost ku zlepšení se již pro takto malé skupiny tolik nevyplácí.

Počet primitiv na list je nastaven na čtyři.

2.4.2 Konečná úprava hierarchie

Výsledkem stavby je binární strom, ve kterém každý vnitřní uzel udržuje ukazatel na své syny a každý list referenci na jedno či více primitiv. Nakonec je tento strom ještě převeden do kompaktnější lineární verze, která již nepoužívá

ukazatele a uzly jsou zarovnány na $32B$ paměti. Uzly jsou navíc uloženy v paměti tak, aby levý syn každého vnitřního uzlu ležel na následující pozici v poli a rodič si díky tomu pamatuje pouze index na pravého syna (viz obr. 2.4). Dalším důležitým efektem je, že pole primitiv celé scény je během této fáze přeuspořádáno tak, že primitiva odkazovaná z jednoho listu nyní obsazují sousední pozice v tomto poli. Díky tomu je kromě efektivity při čtení navíc možné si pro primitiva v listu pamatovat pouze dvě hodnoty: index do pole primitiv a počet primitiv.



Obrázek 2.4: Ukázka přestavby binární hierarchie ze stromové struktury do lineárního pole (převzato z [PJH16]).

2.4.3 Traverzace

Traverzace v PBRT je velice jednoduchá, nevyužívá rekurze a jejím cílem je co nejmenší nutná manipulace s daty. Každé vlákno navíc využívá pro traverzaci vlastní zásobník reprezentovaný jednoduchým polem celočíselných hodnot s indexy procházených uzlů. Během traverzace vnitřního uzlu se nejprve provede test průsečíku paprsku s obalovým tělesem, který určí, zda-li je uzel paprskem zasažen a muže se v něm nacházet zlepšující řešení, tedy bližší než doposud nalezený nejbližší průsečík. V případě, že paprsek nezasáhl obalové těleso, nebo se v uzlu nemůže nacházet lepší řešení, pokračuje traverzace dalším uzlem z vrcholu zásobníku, nebo končí, je-li prázdný. Při úspěšném pokračování do vnitřního uzlu se do zásobníku ukládá pouze vzdálenější ze synů a výpočet rovnou pokračuje bližším. Tímto způsobem se tak využívá toho, že pokud nalezneme průsečík již v bližším synovi, pravděpodobně nebude třeba vzdálenější uzel vůbec procházet, jelikož v něm nebude možné nalézt bližší průsečík.

Traverzace pro stínové paprsky je téměř identická, jediným rozdílem je předčasné ukončení po nalezení prvního průsečíku, namísto hledání nejbližšího.

2.4.4 Měření

Oproti původní implementace v PBRT jsme provedli změnu v metodě syntézy obrazu BDPT (viz podsekce 2.1.6) a způsobu testování viditelnosti mezi dvojicí bodů z vygenerovaných cest. Zatímco v původní verzi dochází k hledání

nejbližšího průsečíku, v našem případě provádíme pouze test zastínění, bez ohledu na průhlednost zasaženého objektu a opticky aktivní prostředí. Ačkoli je tento postup fyzikálně méně přesný, přináší do měření testy pro stínové paprsky, na které (s lehkým předbíháním do výsledků analýzy) jsou zaměřeny některé z optimalizačních metod představené v části realizace této práce. Jelikož v žádné z testovaných scén není použito opticky aktivní prostředí není rozdíl této změny tak výrazný, u většiny použitých scén navíc nedochází k žádné změně ve výstupním obrazu.

Dále u metody přímého osvětlení (viz podsekce 2.1.4) s námi zvoleným cílem na fotorealističnost výsledného obrazu využíváme strategii průchodu všech světel, což je důvodem výrazně delších časů u některých scén oproti ostatním metodám syntézy obrazu.

Samotné měření dále proběhlo jak na množství scén dostupných společně s PBRT, tak i na dalších scénách pro tento software upravených s přehledem počtu primitiv a odpovídajících BVH na základě zvolené metody konstrukce v tabulce 2.1. Základní informace o průběhu vykreslení scény a výsledná statistika byla již implementována v základu softwaru, pro podrobnější analýzu však bylo třeba přesněji identifikovat zdroje hodnot a přidat další podstatné údaje. K vypisované statistice proto byly navíc přidány další rozšiřující informace jako například:

- Čas konstrukce BVH.
- Počet použitých typů primitiv.
- SAH cost [JMV13] s cenou $c_T = 3$ a $c_I = 2$.
- Počet uzlů BVH a jejich typ.

Ačkoli je výpis počtu uzlů BVH již součástí PBRT, jedná se pouze o počet unikátních uzlů skutečně uložených v paměti. Kvůli použití instancování objektů jsme proto přidali výpis celkového počtu zahrnujícího i rozdílné instance, který lépe reprezentuje složitost konkrétní scény. Byl zde navíc přidán i další typ uzlu nazvaný *hybrid*, který je sice v BVH použitý jako list, jedno či více z něj odkazovaných primitiv je ale opět akcelerační struktura jenž sama obsahuje další hierarchii. Tento způsob řešení je velice užitečný v úspoře místa, jelikož si stačí pro instanci pamatovat jediný reprezentativní uzel, který si kromě ukazatele na samostatně vytvořenou hierarchii instancovaného objektu pouze pamatuje použité transformace a není tak třeba duplikovat veškeré uzly. Jediná nevýhoda tohoto řešení nastala při vykreslování scény *San Miguel* s BVH stavěného metodou *HLBVH*, kde tak tyto hybridní listy překračovaly maximální možný počet primitiv 2^{16} dovolující velikostí čítače odkazovaných primitiv v listu.

Scene	#primitives total	build method	SAH cost	BVH node types Total	Interior	Leaf	Hybrid
Conference	123651	equal	296.25	244397	122198	122199	0
		middle	157.27	240023	120011	120012	0
		hlbvh	133.31	84211	42105	42106	0
		sah	128.18	197933	98966	98967	0
Buddha	1087720	equal	71.01	2164659	1082329	1082330	0
		middle	22.66	2150127	1075063	1075064	0
		hlbvh	22.82	771665	385832	385833	0
		sah	16.25	1812473	906236	906237	0
Crown	3540215	equal	83.53	7034407	3517203	3517204	0
		middle	55.50	6970597	3485298	3485299	0
		hlbvh	42.41	446573	223286	223287	0
		sah	34.85	5738807	2869403	2869404	0
San Miguel	10318776	equal	130.86	20369909	10184954	10184956	69039
		middle	63.44	20126503	10063251	10063256	69036
		sah	54.94	15708401	7854200	7862013	61228
Ecosystem	19042013	equal	281.48	37989659	18994829	18994862	4785
		middle	198.64	37841095	18920547	18920612	4753
		hlbvh	197.37	18849225	9424612	9426355	3075
		sah	175.10	26100209	13050104	13050548	4374

Tabulka 2.1: Přehled scén s odpovídající BVH postavenou podle zvolené metody konstrukce.

2.4.5 Profilování

Dalším zdrojem informací bylo vestavěné profilování PBRT, které využívá vestavěných signálů pro operační systémy Linux. Pokud je tato možnost dostupná a zapnutá, tak se během výpočtu periodicky naplánují dočasná přerušení, která zjišťují stav, ve kterém se právě aplikace nachází pomocí nastavovaných značek při vstupu a výstupu do částí (funkcí) programu. Samotné přerušení pak na jejich základě pouze zapříčiní inkrementaci odpovídajícího čítače, které se anulují s další scénou. Jelikož jsou čítače a nastavování značek pouze pro primární části programu, celkové množství práce pro profilování by tak mělo být co nejnižší a dát dostatečně přesný výstup ve formě seznamu funkcí s indikátorem v nich stráveného času a poměru vůči celému běhu programu. Výstupní procenta a časy jsou za celkový běh programu, tedy včetně načítání vstupního souboru a tvorbě struktury scény.

Výstupní data z profilování se nakonec ukázala jako velice užitečné, především z důvodu určování úzkého hrdla výpočtu napříč scénami s různou charakteristikou. Celkové zpomalení v důsledku profilování se pro 64 běhů programu na různých scénách a vstupních parametrech ukázalo jako 3.67% se směrodatnou odchylkou 6.76%, jedná se tedy o přijatelně malé zpomalení.

2.4.6 Kombinace scén a vstupních parametrů

PBRT bez významných úprav nedokáže vykreslit některé kombinace scén a zvolených metod, jako například již zmíněný problém ve scéně *San Miguel* s metodou stavby *HLBVH*, kde jsou použity hybridní listy udržující instance a překračující tak maximální povolený počet primitiv.

Dalším problémem pro vykreslení se ukázalo použití syntézy obrazu metodou *directlighting* strategií *UniformSampleAll* pro scény *Conference* a *Crown*. Třída *sampler* má za úkol tvorbu n-dimenzionálního vektoru vzorků v $[0, 1)^n$, která se provádí pro každý vzorek na pixel s různou hodnotou n na základě použité vykreslovací metody. Namísto generování každého vzorku zvlášť, rovnoměrně z intervalu $[0, 1)$, se zde používá více sofistikovaná metoda generování většího množství vzorků. Vzorky navíc obecně nejsou reprezentovány explicitně, nebo přímo uloženy, ale jsou generovány inkrementálně podle potřeby zvoleného algoritmu syntézy obrazu. Zádrhel nastává pro metodu *directlighting*, kde pro každý vzorek na pixel je potřeba získat kromě náhodných čísel pro kameru (např. pozice vzorku uvnitř zpracovávaného pixelu) také různý počet vzorků pro každé světlo, které vzorky vyžaduje. Při volbě strategie procházející všechna světla tak je překročena maximální nastavená dimenze pro vzorkování (námi použitý vzorkovač *Halton* s omezením dimenze rovné 1000). Důsledkem je předčasné ukončení programu, což je důvodem chybějících dat v tabulce měřeních 2.2.

Konfigurace pro testování:

- CPU: 2× Intel Xeon E5-2620 2.4 GHz, celkem 32 vláken
- RAM: 64 GB
- OS: Ubuntu 14.10
- Kompilátor: gcc verze 4.9.1
- Veškerá rozlišení obrázků: 1024x1024 pixelů
- Vzorků na pixel: 16, 64, 128
- Sampler: Halton

2.5 Detekce úzkého hrdla výpočtu v PBRT

Stručný souhrn naměřené statistiky na pětici scén je vynesen do dvojice tabulek 2.2 a 2.3. Naměřená data jsou získána za použití, až na zmíněné výpisy, neupravené binární hierarchie obalových těles, která je součástí PBRT. Doba stavby BVH metodou *HLBVH* je výrazně rychlejší než pro ostatní metody. Ve srovnání rychlostí pak vychází nejlépe metoda *SAH* s naopak nejdelší dobou stavby způsobenou jak obtížností paralelizace, tak hlavně díky potřebným výpočtům navíc.

	BVH build method	SAH cost [-]	trace speed [Mrays/s]	build time [s]	render time [s]	#ray-box tests				#ray-primitive tests			
						diff		shadow		diff		shadow	
						[M]	[%]	[M]	[%]	[M]	[%]	[M]	[%]
Conference room			#triangles 124k			#instances 1							
BDPT	equal	296	5.68	0.096	892.6	251406	34.19	248360	30.28	6955	5.51	4122	1.64
	middle	157	8.85	0.155	572.6	126135	27.65	114394	22.52	6911	7.79	4199	2.38
	HLBVH	133	8.10	0.023	627.6	102522	25.48	91019	20.8	7855	8.49	4867	2.86
	SAH	128	8.91	0.224	572.6	100291	25.28	95466	20.79	8259	8.99	5200	3.11
path	equal	296	6.17	0.093	361.5	123948	42.12	60743	19.19	3421	7.53	1080	1.07
	middle	157	7.62	0.083	292.5	61877	33.35	27695	14.06	3384	10.65	1093	1.68
	HLBVH	133	9.18	0.022	243.5	50374	31.26	22199	12.24	3885	12.27	1265	2.15
	SAH	128	8.50	0.228	263.5	49036	29.83	23339	12.64	4105	12.86	1360	2.39
Buddha			#triangles 1087k			#instances 1							
BDPT	equal	71	12.61	1.211	114.5	35947	22.71	13761	9.99	902	4.53	314	0.81
	middle	22	8.13	2	177.5	17941	15.31	7600	7.56	944	4.98	327	1.03
	HLBVH	22	13.18	0.167	109.5	19482	18.38	8462	8.89	1751	6.29	726	1.8
	SAH	16	13.55	2.289	106.5	15228	15.45	6683	7.62	1023	4.97	377	1.23
directligh.	equal	71	8.48	2.651	88.5	22607	26.38	6733	9.4	575	5.72	174	1.03
	middle	22	16.86	0.896	44.5	11793	18.9	3960	7.97	600	6.32	181	1.08
	HLBVH	22	15.47	0.224	48.5	12648	21.02	4391	8.87	1099	8.06	384	2.03
	SAH	16	17.25	2.289	43.5	10141	18.08	3496	7.16	652	6.66	207	1.28
path	equal	71	6.28	2.65	177.5	50043	35.71	10833	8.71	1263	6.1	287	0.83
	middle	22	6.32	0.897	176.5	26966	29.33	6361	7.55	1305	7.42	300	1.12
	HLBVH	22	11.79	0.171	94.5	29885	31.58	7116	8.08	2697	9.56	661	1.75
	SAH	16	13.85	2.292	80.5	23170	27.07	5683	7.24	1445	7.64	345	1.27
whitted	equal	71	21.02	1.212	29.5	10751	19.57	6733	15.56	310	5.83	174	1.64
	middle	22	24.32	0.892	25.5	5520	12.51	3960	12.81	329	6.64	181	1.79
	HLBVH	22	22.97	0.195	27	5665	13.12	4391	15.13	518	6.93	384	3.26
	SAH	16	24.32	2.29	25.5	4797	11.81	3496	12.63	344	6.39	207	2.1
Crown			#triangles 3540k			#instances 1							
BDPT	equal	83	8.19	3.355	862.6	166606	24.34	148308	20.92	5103	2.62	4384	1.52
	middle	55	10.97	3.325	642.6	87495	18.38	71214	15.32	5081	3.27	4358	1.95
	HLBVH	42	9.48	0.608	742.6	65826	18.29	54511	14.96	20886	8.77	14058	5.18
	SAH	34	11.44	19.795	617.1	65666	16.12	55762	14.03	5344	3.57	4524	2.05
path	equal	83	8.52	3.284	402.5	102708	28.33	71569	18.45	3331	4.35	2605	1.93
	middle	55	8.87	3.282	385.6	52528	21.18	33819	13.54	3314	5.16	2591	2.34
	HLBVH	42	8.49	0.721	402.5	40714	20.1	25838	12.41	16026	11.73	8562	5.36
	SAH	34	9.93	7.736	345.5	40756	20.67	27585	12.6	3548	6.08	2695	2.58
whitted	equal	83	6.67	3.291	435.6	45160	9.64	220121	49.41	1081	1.13	8004	5.05
	middle	55	8.81	3.37	327.5	18594	7.23	98541	37.85	1079	1.87	7960	7.45
	HLBVH	42	9.46	0.851	304.5	14991	5.76	76034	31.5	4469	3.47	29911	19.2
	SAH	34	10.07	8.086	288.5	14991	6.24	82043	34.72	1169	1.94	8419	8.13

Tabulka 2.2: Porovnání výkonu testovaných struktur BVH pro různé metody stavby a algoritmy vykreslování. Pro výpočet SAH cost jsme použili $c_T = 3$ a $c_I = 2$. Hodnoty *diff* označují testy během dotazů na nejbližší průsečík a *shadow* dotazy zastínění. Procenta uvádí poměr času stráveného jednotlivými testy vůči celkovému času syntézy obrazu.

Souhrnné informace z profilování a časově nejvíce významné fáze z celého výpočtu jsou patrné v obr. 2.5. Konkrétní vykreslené fáze jsou zde pouze ty, které pro některou ze scén samostatně zabírají alespoň 5% z celkového času, nebo po sloučení do logických celků.

Například pro scénu *Buddha* nemalou část celkového běhu programu zabírá tvorba scény a načítání ze vstupního souboru v průměrnou okolo 5%, i přes množství měření s 16, 64 a 128 vzorky na pixel. Příčinou je zde především tvorba mipmap a jejich trilineární interpolace. Podobně pak u scény *San Miguel*, zde je však příčinou samotné načítání textur a jejich zpracování. Další podstatnou částí ve které se tráví čas výpočtu je vzorkování světel. Namísto

	BVH	SAH	trace	build	render	#ray-box tests				#ray-primitive tests			
	build	cost	speed	time	time	diff		shadow		diff		shadow	
	method	[-]	[Mrays/s]	[s]	[s]	[M]	[%]	[M]	[%]	[M]	[%]	[M]	[%]
San Miguel			#triangles 10319k			#instances 69k							
BDPT	equal	130	5.99	2.015	637.6	242917	49.07	69885	14.43	7672	7.11	2370	1.47
BDPT	middle	63	7.24	2.084	527.6	140077	41.91	41053	12.55	7750	9.09	2380	1.84
BDPT	SAH	54	7.92	5.148	507.6	117838	39.87	34354	11.9	8934	10.12	2643	2.05
directl.	equal	130	6.29	2.048	5467.7	1427420	32.72	1577170	24.41	49333	5.18	51980	2.43
directl.	middle	63	7.62	2.046	4517.2	850240	28.61	908474	20.07	50424	6.71	51433	3.06
directl.	SAH	54	8.09	5.239	4487.7	714901	27.44	754739	18.86	57899	7.64	57297	3.52
path	equal	130	4.71	2.01	932.6	360677	59.57	56226	9.12	11771	8.51	2016	1.04
path	middle	63	5.69	3.483	772.1	212110	51.52	34221	8.37	11903	10.89	2069	1.27
path	SAH	54	6.55	9.808	712.6	177700	49.13	28274	7.94	13737	12.36	2240	1.42
whitted	equal	130	10.40	2.178	156.5	67081	24.98	75575	30.82	1583	3.2	2397	2.91
whitted	middle	63	14.04	2.044	115.5	37286	20.42	42837	26.47	1599	4.47	2362	3.87
whitted	SAH	54	15.49	5.309	109.5	31791	18.54	35826	24.04	1874	4.86	2651	4.61
Ecosystem			#triangles 19042k			#instances 4.8k							
BDPT	equal	281	10.41	1.49	363.5	140694	49.62	41127	14.3	5767	6.33	1742	1.25
BDPT	middle	198	9.67	0.731	391.6	106796	48.03	31408	13.83	5748	7.17	1754	1.51
BDPT	HLBVH	197	13.32	0.5	405.6	97104	47.29	27672	13.23	9815	9.51	2714	1.9
BDPT	SAH	175	16.97	3.912	246.5	94587	45.53	28177	13.13	6623	8.02	1983	1.68
directligh.	equal	281	17.05	0.83	90.5	56742	49.35	14266	13.84	2021	5.59	571	1.33
directligh.	middle	198	10.12	0.761	152.5	42747	45.48	10854	13.36	2003	6.64	575	1.43
directligh.	HLBVH	197	26.71	0.435	82.5	38480	44.6	9335	12.32	3526	8.6	890	1.75
directligh.	SAH	175	22.58	1.975	75.5	38012	43.66	9688	12.5	2327	6.99	658	1.52
path	equal	281	10.15	0.876	368.5	158817	58.33	35759	12.38	6611	7.43	1517	1.14
path	middle	198	12.22	1.645	306.5	121581	54.62	27513	11.18	6580	7.87	1528	1.25
path	HLBVH	197	15.25	0.427	366.6	109774	55.23	24130	10.98	11251	10.29	2376	1.63
path	SAH	175	13.47	3.956	312.5	107745	53.72	24524	11.25	7646	8.78	1746	1.45
whitted	equal	281	20.27	0.851	65.5	44361	46.16	14266	19.37	1513	5.59	571	1.7
whitted	middle	198	23.92	0.685	55.5	33228	42.44	10854	18.31	1498	6.71	575	1.74
whitted	HLBVH	197	31.22	0.472	59.5	30037	42.57	9335	16.92	2668	8.46	890	2.54
whitted	SAH	175	26.68	2.827	54.5	29624	41.05	9688	17.64	1735	6.84	658	2.22

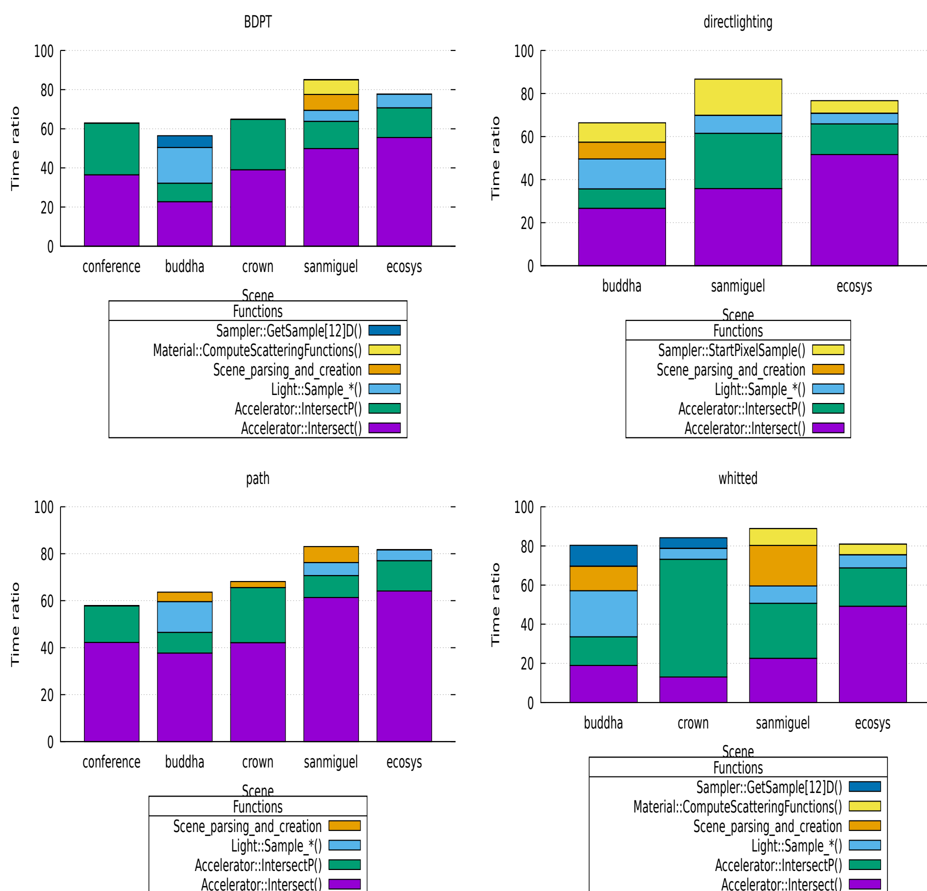
Tabulka 2.3: Porovnání výkonu testovaných struktur BVH pro různé metody stavby a algoritmy vykreslování. Pro výpočet SAH cost jsme použili $c_T = 3$ a $c_I = 2$. Hodnoty *diff* označují testy během dotazů na nejbližší průsečík a *shadow* dotazy zastínění. Procenta uvádí poměr času stráveného jednotlivými testy vůči celkovému času syntézy obrazu.

distribuce vzorků za pomoci BSDF, je použit přístup podle zdrojů světla ve scéně. Během postupu je tak možné pro vzorek vybrat pouze jeden ze směrů, ve kterých je potenciálně světlo viditelné. Funkce zabývající se těmito náležitostmi jsou označeny jako *Light::Sample*.

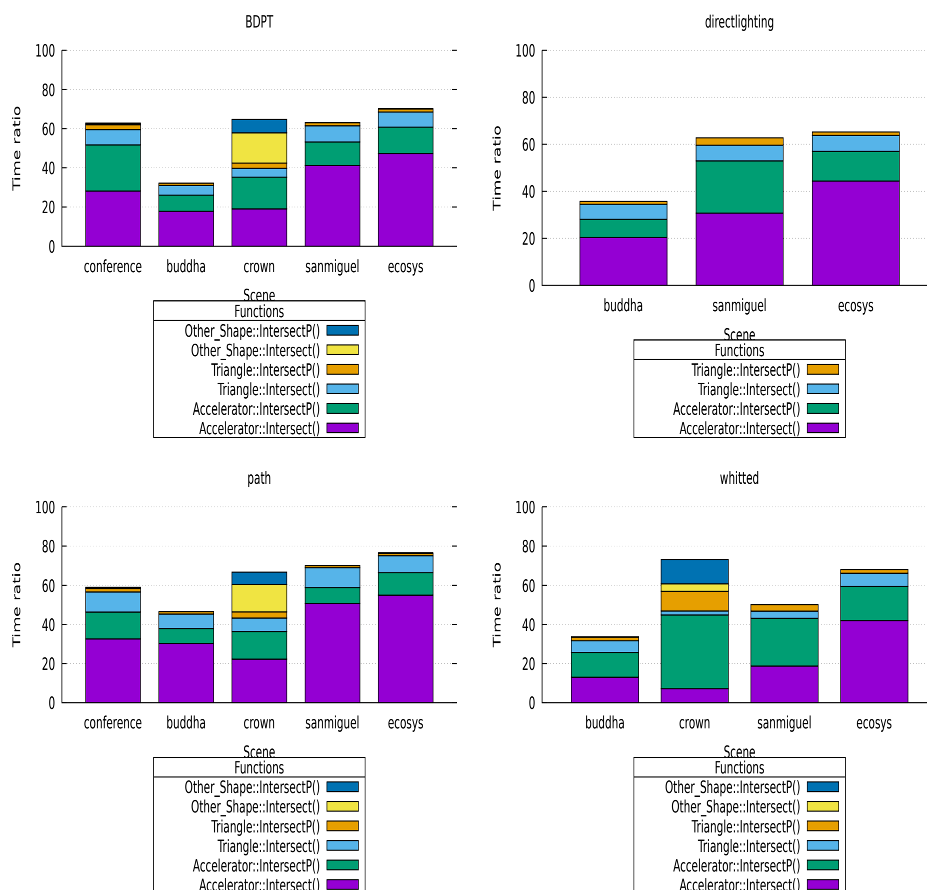
Viditelně nejvíce významnou částí se ukázalo hledání průsečíků, pro které je strávený čas, opět vůči celkové době výpočtu, vynesen navíc zvlášť do grafů na obr. 2.6. Celá tato fáze je dále rozdělena do několika funkcí, podle toho, zda se jedná o hledání průsečíku během traverzace BVH, nebo již výpočet u konkrétního primitiva scény. Je zde patrné, že většina času se tráví samotnou traverzací akcelerační struktury.

Ačkoli pro některé scény a zvolené metody určité fáze jako například práce s texturami (mipmapping), vzorkování a výpočet rozptylovací funkce zabírají podstatnou část celkového času výpočtu, napříč všemi scénami ale zůstává jednou z nejdelších částí hledání průsečíku s paprskem. Ze získaných dat pro 64 i 128 vzorků na pixel se dokonce projevilo dotazování do akcelerační struktury a hledání průsečíku jako nejdelší část výpočtu. Akcelerační struktura BVH

se tak ukázala jako vhodný kandidát pro optimalizaci za účelem celkového zrychlení.



Obrázek 2.5: Nejdelší fáze výpočtu podle metody syntézy obrazu s poměrem času k celkové době běhu programu.



Obrázek 2.6: Poměr celkového času dotazováním do akcelerační struktury při hledání nejbližšího průsečíku s paprskem, nebo dotazy viditelnosti označené příponou *P*. Testy označené *Accelerator* udávají testování průsečíku paprsku s obalovým tělesem během traverzace BVH, zbylé metody pak test průsečíku s konkrétním primitivem scény.

Kapitola 3

Realizace

Optimalizaci akcelerační datové struktury BVH provádíme na základě několika metod a jejich kombinací. Zatímco některé metody směřují na celkové zrychlení vyhodnocování dotazů do akcelerační struktury, jiné jsou zaměřeny pouze na zpracování stínových paprsků, popřípadě paprsků hledajících nejbližší průsečík.

Nejprve popíšeme rozšíření binární hierarchie na n-ární za použití přestavby kontrakcí (viz sekce 3.1) a odpovídající změnou v paměti uzlů hierarchie s ohledem na jejich výslednou velikost (podsekce 3.1.2). Následně pak změnu traverzace n-ární hierarchie za použití dvojice metod využívající řazení synů vůči sledovanému paprsku a změnu práce se zásobníkem (podsekce 3.1.3).

V sekci 3.2 dále popíšeme optimalizaci BVH na základě plochy s odpovídající změnou během kontrakce (podsekce 3.2.1) a seřazením potomků uzlů za účelem rychlejšího zpracování stínových paprsků (podsekce 3.2.2).

Další část je určena pro pohledově závislou optimalizaci (sekce 3.3), ve které dochází ke změně běhu programu rozšířením o fázi směru dat (viz podsekce 3.3.2), ve které jsou získána statistická data paprsků ve scéně. Na základě získaného odhadu distribuce paprsků se dále provádí upravená přestavba BVH kontrakcí se změněnou cenovou funkcí (podsekce 3.3.3) a přeuspořádání potomků (podsekce 3.3.4).

V předposlední sekci 3.4 se budeme zabývat nově navrženou metodou určování průchodu synů během traverzace na základě předpočítaných hodnot v uzlu a směru sledovaného paprsku s využitím nahlížecí tabulky.

Na závěr pak porovnáme jednotlivé metody a jejich vhodné kombinace s možnými modifikacemi (sekce 3.5).

3.1 BVH s vyšší aritou

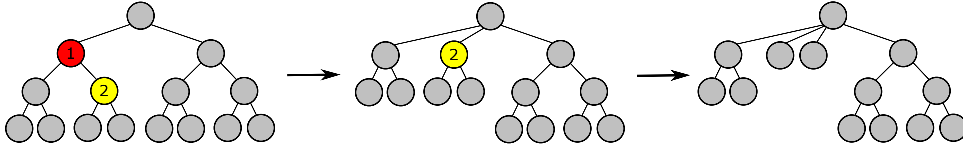
Počáteční část optimalizace spočívá v přestavbě původní binární hierarchie na n-ární pro redukci testů průsečíku paprsku s obalovým tělesem snížením počtu vnitřních uzlů. I když je možné provést konstrukci n-ární BVH přímo

z primitiv, budeme vycházet z binární hierarchie postavené jednou ze zmíněných metod, které jsou součástí PBRT. Tato počáteční binární hierarchie je následně přestavěna podle zvolené metody optimalizace:

- Podle plochy obalových těles: k přestavbě dochází okamžitě po dokončení stavby binární hierarchie na základě plochy obalových těles (viz sekce 3.2.2).
- s ohledem na distribuci paprsků ve scéně: binární BVH je nejprve použita pro sběr dat a teprve následně, na jejich základě, přestavěna na n-ární (viz sekce 3.3).

Nehledě na datech řídících přestavbu v obou případech k ní dochází pomocí kontrakce vnitřních uzlů (def. 1), definované v [GHB15]. Ukázka postupné kontrakce dvou uzlů viz obr. 3.1.

Definice 1. Operace kontrakce pro vnitřní uzel BVH je přesunutí všech jeho synů do rodiče a odstranění tohoto uzlu ze stromu.



Obrázek 3.1: Postupná kontrakce dvou vnitřních uzlů s důsledkem redukce dvou testů průchodu pro paprsky traverzujících skrze oba uzly.

3.1.1 Kontrakce

Namísto maximálního plnění vnitřních uzlů postupujeme za pomoci cenové funkce $\delta(N)$, která nám vrací cenu při kontrakci uzlu N oproti zanechání BVH v původním stavu. Dojde-li k průchodu paprsku daným uzlem, je třeba zaplatit cenu $(1 + n_{N.child}) * C_B$, kde $n_{N.child}$ je počet potomků uzlu N a C_B cena testu průsečíku paprsku s obalovým tělesem. V opačném případě dochází k prořezání stromu s cenou jednoho testu průsečíku s obalovým tělesem a žádný z potomků není dále traverzován. Pro pravděpodobnost α_N průchod uzlu N tak dostáváme

$$
\begin{aligned}
\delta(N) &= n_{N.child} C_B - \ (\alpha_N (1 + n_{N.child}) + \ (1 - \alpha_N)) C_B \\
&= n_{N.child} C_B - \ (\alpha_N * n_{N.child} + 1) C_B \\
&= \ (n_{N.child} - \alpha_N * n_{N.child} - 1) C_B \\
&= \ ((1 - \alpha_N) n_{N.child} - 1) C_B .
\end{aligned}
$$

Operace kontrakce je výhodná, pokud výsledná cenová funkce splňuje $\delta(N) < 0$, což lze rozepsat do tvaru

$$\alpha_N > 1 - \frac{1}{n_{N.child}}.$$

Cílená přestavba se pak provádí na základě algoritmu kaskádové kontrakce (viz alg. 1) pracujícím na principu hladového přístupu. Algoritmus spouštíme shora-dolu, kde postupně provádí kontrakci vybraných uzlů na základě definované cenové funkce. U každého vnitřního uzlu si udržujeme množinu kandidátů S pro kontrakci, počínaje dvojicí původních synů zpracovávaného uzlu. Dokud pak není porušeno kritérium kontrakce (*ContractionCriterion*), tedy nerovná-li se velikost množina S maximálního počtu synů pro uzel, nebo kontrakce jakéhokoli z uzlů $s \in S$ již není výhodná, vyjme se v každé iteraci jeden z kandidátů vybraný funkcí *Select* na kterého se provede kontrakce a jeho synové se přidají do množiny S. V opačném případě se stanou z kandidátů potomci zpracovávaného uzlu a rekurzivně nimi přestavba pokračuje. Výběr uzlu funkcí *Select* je prováděn na základě cenové funkce. Nejen, že kontrakce vybraného uzlu musí být výhodná (existence takového uzlu je kontrolována jako součást kritéria kontrakce), ale s omezením na aritu BVH chceme, aby byla co nejvýhodnější. Výběr tak provádíme na základě prioritní fronty množiny kandidátu S, podle hodnot pravděpodobnosti průchodu α_N daným uzlem N a kontrolu přínosu kontrakce podle rovnice 3.1.

```
1  Function BVHContract(BVHNode N) is
2      if N.isLeaf then
3          return;
4      else
5          S ← {N.left, N.right};
6          while ContractionCriterion(S) do
7              s ← Select(S);
8              S ← S - {s} + {s.left, s.right};
9          end
10         N.children ← S;
11         foreach s ∈ S do
12             BVHContract(s);
13         end
14     end
15 end
```

Algoritmus 1: Pseudokód přestavby BVH pomocí kontrakce vnitřních uzlů. Kritérium kontrakce (*ContractionCriterion*) ukončuje zpracování dalších synů na základě cenové funkce $\delta(N)$ a maximálního počtu synů na uzel.

Průběh přestavby provádíme shora-dolu, i když je možné využít i postup zdola-nahoru. Hlavním důvodem je optimalizace paměti na základě prostorové lokality, bez nutnosti další přestavby hierarchie (jedná-li se o konečnou

úpravu). Zdrojem je uložení potomků zpracovaného uzlu do souvislého bloku paměti. Během výpočtu se již zpracovaná část nemění a každý z uzlů je navštíven pouze jednou, stejně tak je pouze jednou vypočtena i příslušná pravděpodobnost průchodu α. Bez omezení na počet synů a s opatrnou implementací má kontrakce BVH lineární časovou složitost vůči počtu navštívených synů, jelikož je každý z uzlů navštíven pouze jednou. Pokud uzel N splňuje $\delta_N < 0$, dochází k jeho kontrakci rovnou, v opačném případě je zařazen na další pozici syna zpracovávaného uzlu do výstupního pole a není znovu kontrolován. Pokud se omezíme na maximální počet synů a využijeme zmíněnou prioritní frontu ve které v průběhu zpracování uzlu udržujeme množinu kandidátů S. Dokud není porušena podmínka kontrakce, během každé iterace dochází k jednomu vyjmutí maxima určující uzel podléhající kontrakci a následné vložení obou jeho synů do fronty. Pro každou kontrakci je tak třeba v uzlu povést navíc až $\log(|S|-1)+\log(|S|)+\log(|S|+1)$ operací vzhledem k velikosti množiny S na počátku iterace. Jelikož je navíc vždy $|S| \geq 2$, lze tento počet shora omezit hodnotou $3\log(|S|)$. Pro celý běh algoritmu je tak nutné provést až

$$\sum_{i=1}^{H_{in}} C_v + \sum_{i=1}^{H_{out}} \sum_{k=1}^{K_i} 3 * \log(k+1)$$

operací, kde H_{in} udává počet uzlů přestavované hierarchie, H_{out} počet uzlů výstupní hierarchie, C_v počet operací pro výpočet cenové funkce uzlu a K_i počet kontrakcí provedených v uzlu i. Přepočteme-li dále počet operací k určení horního odhadu tak, že počet kontrakcí každého výstupního uzlu bude roven jejich maximu přes všechny uzly, tedy $K_{max} = \max\{K_i : i \ \in H_{out}\}$ a počet operací během kontrakce bude určen velikostí výsledné množiny S, získáváme

$$\begin{aligned}\sum_{i=1}^{H_{in}} C_v + \sum_{i=1}^{H_{out}} \sum_{k=1}^{K_i} 3 * \log(k+1)) &< \sum_{i=1}^{H_{in}} C_v + \sum_{i=1}^{H_{out}} \sum_{k=1}^{K_{max}} 3 * \log(K_{max}+1) \\ &< \sum_{i=1}^{H_{in}} C_v + \sum_{i=1}^{H_{out}} K_{max} * 3 * \log(K_{max}+1).\end{aligned} \tag{3.1}$$

Omezíme-li dále maximální počet potomků konstantou výrazně menší než celkový počet uzlů a jelikož $H_{in} \geq H_{out}$, získáváme asymptotickou složitost našeho horního odhadu

$$\begin{aligned}\sum_{i=1}^{H_{in}} C_v + \sum_{i=1}^{H_{out}} K_{max} * 3 * \log(K_{max}+1) &\in O(H_{in}C_v + 6H_{out}K_{max}) \\ &\in O(H_{in} + H_{out}) \\ &\in O(H_{in}).\end{aligned}$$

Asymptotická složitost s omezením arity BVH a použitím prioritní fronty je tak stále lineární vůči počtu uzlů přestavované hierarchie. Ve zbytku celého

textu se dále omezujeme na přestavbu s výslednou aritou rovnou čtyřem. Použití většího počtu synů na uzel je možné, tuto hodnotu jsme ale zvolili z důvodu používání nahlížecí tabulky pro určení pořadí traverzace synů v závislosti na směru paprsku (viz sekce 3.4.2) jejíž velikost roste rychlostí faktoriálu s maximálním počtem potomků na uzel.

Další výhodou postupu shora-dolu je možnost předčasného ukončení přestavby. Optimalizující pouze několik vrchních úrovní, nebo počtu přestavěných uzlů zaručující tak rychlou modifikaci. V našem případě ale tuto možnosti nevyužíváme a ačkoli uzly blízko listů již nemají takový vliv na zrychlení, přesto ke zlepšení dochází. Zpomalení v důsledku delší optimalizace se tak pro větší scény, nebo více vzorků na pixel, překoná. Dalším důvodem je kombinace s optimalizací měnící pořadí synů postupující zdola-nahoru, která tak může v listech rovnou navázat.

3.1.2 Změna struktury uzlu BVH

Při změně na n-ární BVH je třeba modifikovat i její uzly. Zatímco chceme zachovat velikost 32 bajtů kvůli zarovnání v cache paměti, musíme být schopni stále rozhodnout, jedná-li se list nebo vnitřní uzel a umět přesně určit odkazovaná primitiva nebo potomky.

V případě listů nedochází k žádné změně přestavbou na n-ární BVH, odkazováno je již i větší množství primitiv a není tak potřeba další změny provádět. Vnitřní uzly jsou naopak přizpůsobeny práci s binární hierarchií (viz sekce 2.4.2), udržující pouze ofset (posun od počáteční pozice) na pravého syna a jsou následování v poli uzlů synem levým. Jelikož není z hlediska paměťové kapacity pro více-ární hierarchie možné uložit, až na prvního syna, ofset na každého z potomků zvlášť, využijeme stejného způsobu, jakého využívají listy hierarchie. Původní ofset na pravého syna použijeme jako ofset na prvního z potomků a do uzlu přidáme hodnotu určující jejich počet. Kvůli omezení maximálního počtu potomků navíc můžeme využít pouze hodnotu s malým rozsahem, využijeme proto za tímto účelem nevyužitou hodnotu 8 bitů, která sloužila pouze k zarovnání celkové velikosti uzlu na 32 bajtů. Provedeným způsobem kontrakce je navíc splněna podmínka, aby odkazovaní pomoci leželi v souvislé paměti, tedy vždy na následujícím ofsetu od předešlého sourozence.

3.1.3 Traverzace pro n-ární BVH

Obecně sledování paprsku i pro n-ární hierarchie probíhá na podobném principu jako prohledávání do hloubky s využitím zásobníku, na který se ukládají potomci, kterými výpočet rovnou nepokračuje a mohou stále obsahovat hledané řešení. Pro každý procházený uzel N tak můžeme na zásobník uložit až $n_{N.child} - 1$ uzlů, kde $n_{N.child}$ je počet potomků uzlu N. Zásobník opět

reprezentujeme pomocí jednorozměrného pole, oproti binární variantě je ale úměrně větší aritě použité BVH, aby nemohlo dojít k přetečení.

V případě, že v n-ární hierarchii dojde k zásahu paprsku s potomky uzlu, dochází k jejich seřazení podle vzdálenosti průsečíků a k následnému umístění na zásobník traverzace tak, aby na vrcholu zůstal ten nejbližší [Áfr13]. Řazení i ukládání provádíme pouze pro uzly, kteří mohou obsahovat lepší než doposud nalezené řešení. Stejně jako u binární varianty navíc nejbližší z uzlů na zásobník neukládáme, ale pokračujeme jím traverzaci. I přes redukci počtu dále zpracovávaných uzlů, představuje seřazení výrazné režijní náklady.

Alternativní možností snižující cenu výpočtu pořadí, ale s neoptimálním pořadím průchodu je použití pouze částečného řazení. V průběhu zpracování procházeného uzlu si pamatujeme pouze nejbližší z testovaných synů, kterým následně bude traverzace pokračovat. Ostatní potomky rovnou ukládáme na zásobník, může-li se v nich nacházet lepší než dosavadní řešení. V případě že testovaný syn může řešení obsahovat a je blíž než dosavadní nejbližší syn, zapamatujeme si jej a není-li první, tak předešlý uložíme na zásobník, pokud je ale vzdálenější, uložíme jej na zásobník rovnou bez dalšího řazení. Ve výsledku tak vždy nejprve procházíme nejbližším ze synů ve směru paprsku a díky odkládání uložení dočasně nejbližšího uzlu, může před jeho výměnou za bližší dojít k částečnému uspořádání dalších potomků. V případě, že nalezneme nejbližší ze synů jako první, nedochází u zbytku sourozenců k žádné optimalizaci jejich pořadí. Výjimkou je pouze nezařazení uzlů jistě neobsahujících řešení. Může tak dojít k traverzaci v opačném směru, než by bylo optimální, eliminující tím jakékoli prořezávání stromu hierarchie. Ačkoli je tato metoda výpočetně méně náročná, celkový počet traverzací během výpočtu obvykle vzroste a i přes menší počet operací v každém uzlu, může pro některé ze scén dojít ke zpomalení oproti metodě řadící všechny syny.

Způsob testování existence řešení před uložením na zásobník má své výhody, hlavně ve způsobu práce s pamětí:

- Princip lokality

  Testovaní potomci jsou uloženy v souvislém bloku paměti, v závislosti na velikosti řádku cache paměti se tak snižuje počet cache-miss přístupů využitím principu prostorové lokality.

- Použití zásobníku

  na zásobník jsou ukládáni pouze uzly, kteří mohou obsahovat lepší než doposud nalezené řešení. Celkově se tak redukuje počet ukládaných uzlů na zásobník a velikost přepisované paměti.

U metod určujících pořadí pro průchod potomků uzlu nevyužívajících řazení podle vzdálenosti (viz sekce 3.4) přesto provádíme test průsečíku paprsku s obalovým tělesem až po vyjmutí uzlu ze zásobníku. Dochází tak ke většímu počtu prořezávání stromu a tedy menšímu počtu nutných testů průsečíku paprsku s obalovým tělesem. Důvodem je aktualizovaná hodnota hledané

nejbližší vzdálenosti průsečíku paprsku s primitivem z již prošlých sousedů testovaného uzlu.

3.2 Optimalizace podle plochy

Běžnou technikou pro určení pravděpodobnosti průchodu synem je poměr jeho velikosti s velikostí rodiče [WBB08]. na tomto základě lze velice rychle provést jak přestavbu binární BVH na n-ární [GHB15], tak i určit pořadí traverzace synů pro stínové paprsky [NM14, ODJ16].

V následujících podsekcích popíšeme nejprve metody kontrakce podle plochy (SATC), následovanou určením pořadí pro traverzaci stínovými paprsky podle plochy (SATO). Obě metody jsou založeny na předpokladu, že paprsky jsou nekonečně dlouhé a jejich distribuce je zcela náhodná. Jelikož známe velikost všech uzlů okamžitě po dokončení stavby zdrojové binární hierarchie a není třeba dalších vstupních parametrů, dochází k oběma optimalizacím rovnou a současně. V průběhu zpracování uzlu N se nejprve provede kontrakce potomků vybíraných z prioritní fronty na základě cenové funkce $\delta(N)$. Následuje uložení získané množiny synů S do výsledného pole hierarchie seřazených na základě použité cenové funkce a ve stejném pořadí pak pokračuje rekurze přestavby shora-dolu. Výsledkem je uložení v paměti optimální pro průchod do hloubky. Nebude-li použit jiný způsob optimalizace určující pořadí synů, budeme takto seřazené potomky dále používat jako standard ve zbytku textu při měření počtu traverzací a průsečíků s primitivy pro stínové paprsky.

3.2.1 Kontrakce podle plochy (SATC)

Pro přestavbu hierarchie na n-ární pouze na základě plochy uzlů využijeme metodu kontrakce nazvanou Surface-Area guided Tree Contraction (SATC) [GHB15] s definicí cenové funkce na základě poměru plochy obalového tělesa syna a rodičovského uzlu. Parametry použité pro algoritmus 1 jsou:

$$
\begin{aligned}
\alpha_N &= \frac{SA(N)}{SA(N.parent)} \\
Select(S) &= s_{max} \qquad \text{s.t. } s_{max} \in s \ \text{ and } \alpha_{s_{max}} \geq \alpha_s \ \forall s \ \in s \\
ContractionCriterion(S) &= \begin{cases} True & \text{if } \delta(s_{max}) > 0 \text{ and } \textit{not } N.parent.full \\ False & otherwise \end{cases},
\end{aligned}
$$

kde $SA(N)$ je plocha obalového tělesa uzlu N a $N.parent.full$ značka, zda-li obsahuje otec N již maximální počet potomků, což je ekvivalentní porovnání velikosti množiny S s aritou výsledné BVH.

Jelikož metoda postupuje shora-dolů, není konečná hodnota $n_{N.child}$ během přestavby známa a při postupu zpracování potomků v nedefinovaném

pořadí tak nelze jednoznačně určit má-li dojít ke kontrakci či nikoli. Naším řešením se stává opět již použitá prioritní fronta. Během kontrakce je jisté, že uzly zařazené místo otce jistě nemohou mít plochu větší než jejich rodič, jinak by došlo k porušení definice obalového tělesa. V důsledku tak hodnota pravděpodobnosti $\alpha_{Q.top}$ uzlu na vrcholu prioritní fronty Q může pouze klesat. Stejně tak pouze roste počet synů zpracovávaného uzlu, jelikož se během kontrakce synové mohou pouze přidávat, čímž roste i hodnota $1 - \frac{1}{n_{N.child}}$ pravé strany rovnice (3.1) určující prospěšnost kontrakce. Není proto možné, aby byla porušena podmínka $\delta(N) < 0$ v důsledku kontrakce dalšího z kandidátů pro uzly, které již byly zpracovány.

3.2.2 Traverzace podle plochy (SATO)

K určení pořadí traverzace pro stínové paprsky na základě plochy využijeme techniku Surface Area Traversal Order (SATO) [NM14], která přiděluje prioritu pro průchod synům s větší plochou k docílení rychlejšího nalezení zastínění. Tato metoda navíc nepotřebuje žádné informace z podstromů a je výpočetně velice nenáročná.

Pro binární akcelerační struktury je pořadí určeno jako

$$TO = \begin{cases} Left\ first & \text{if } SA(l) \geq SA(r) \\ Right\ first & otherwise \end{cases}.$$

Rozšířením na více potomků je v tomto případě jednoduché, jedná se pouze o seřazení podle plochy. Uzly jsou následně přeuspořádány podle získaného pořadí v poli uzlů hierarchie. Během traverzace se stínovými paprsky tak stačí procházet syny podle jejich uložení v paměti.

3.3 Pohledově závislá optimalizace

Jak již bylo popsáno u stavby BVH (sekce 2.3.1), využíváme celkem čtyři metody konstrukce, nabízející určitý kompromis mezi časem stavby, paralelizací a výslednou kvalitou hierarchie. Všechny však mají společné, že předpokládají rovnoměrné rozdělení paprsků ve scéně. Při stavbě tak zohledňují pouze geometrii objektů.

V této sekci se zabýváme optimalizací akcelerační struktury na základě distribuce paprsků ve scéně s jejíž ohledem lze kvalita výsledné hierarchie výrazně zlepšit. Rozdělení paprsků je závislé jak na samotné metodě syntézy obrazu a jejím nastavením, tak především na pohledu kamery a rozmístění světel. Téměř každá změna ve scéně tak má za následek nutnou přestavbu hierarchie pro zohlednění změny rozdělení.

Základem několika z dříve navržených metod, byla optimalizace *SAH*, pro stavbu akcelerační struktury s ohledem na nerovnoměrné rozdělení paprsků

za pomoci analytického přístupu [HM08, FFD09]. Další modifikací za účelem získání přesné cenové metriky byla fáze sběru dat z řídce rozmístěných vzorků, popřípadě u sekvencí z předešlých snímků, začleňující distribuci paprsků do konstrukce BVH [BH09, FLF12]. U obou metod ale dochází k přestavbě celé hierarchie, eliminující možný benefit jinak optimalizované konstrukce. Metoda z které zde vycházíme [GHB15] začíná s libovolnou binární hierarchií obalových těles, kterou na základě sběru dat pomocí souboru řídce rozmístěných vzorkových paprsků rekonstruuje na více-ární hierarchii kontrakcí některých vnitřních uzlů. Během sběru dat si každý uzel udržuje čítač počtu zásahů paprskem, na jejichž základě je kontrakce prováděna.

Další optimalizace je zaměřena na určení pořadí traverzace pro stínové paprsky. Dříve navržená metoda Surface area traversal order (SATO) [NM14] (popsána v sekci 3.2) udává pořadí čistě podle velikosti plochy uzlů a primitiv. Ačkoli je velice rychlá a bez problémů rozšiřitelná na n-ární ($n > 2$) hierarchie, nebere vůbec v potaz distribuci paprsků ve scéně. Nemusí tak docházet ke zrychlení pro scény s velkými primitivy, která nevrhají stín. Zmíněná kontrakce často navštívených uzlů [GHB15] přináší v tomto směru zrychlení, řazením synů uzlu podle počtu zásahů, neuvažuje ale ceny vůči jednotlivým podstromům. Námi použitým řešením je tak řazení podle cenové funkce na základě pravděpodobnosti zásahu řazených synů i pravděpodobnosti zásahu primitiv v příslušných podstromech [ODJ16]. Metoda je vhodná nehledě na aritu BVH a využívá stejného způsobu sběru dat pro určení pořadí, jako je tomu u kontrakce pro přestavbu hierarchie. Je tak možné využít stejné hodnoty získané během fáze sběru dat pro kontrakci a současně určení pořadí.

3.3.1 Algoritmus a implementace

Potřeba statistiky ze vzorkových paprsků přináší nutnou změnu v postupu programu. Společně se sběrem dat je nutné přidat fázi optimalizace hierarchie, která je až po dokončení následována syntézou obrazu pro zbytek paprsků. Výsledný postup tak vypadá:

1. Stavba binární BVH na základě zvolené metody konstrukce.
2. Sledování malého množství paprsků se sběrem dat inkrementací odpovídajících čítačů pro každý uzel i primitivum.

3.a. Aplikace jednofázové optimalizace 3.3.5: přestavba kontrakcí společně s přeuspořádáním.

3.b. Dvoufázová optimalizace 3.3.6:

   a) Přestavba hierarchie kontrakcí.
   b) Sběr dat s použitím n-ární hierarchie.
   c) Přeuspořádání synů a primitiv.

4. Dokončení syntézy obrazu.

Krok 3 je vybrán v závislosti na vybraném způsobu optimalizace, bude-li proveden sběr dat dvakrát s kontrakcí a přeuspořádáním zvlášť nebo pouze jedno a úpravy se provedou současně.

Čas strávený strávený ve krocích 3.*a*, resp. 3.*b*.*a* a 3.*b*.*c* je téměř zanedbatelný, oproti tomu sběr dat v krocích 2 a 3.*b*.*b* přináší množství operací navíc, způsobených především inkrementací čítačů navštívení. Aby navíc nebyla vykreslená část provedena nadarmo, je zakomponována do konečného výsledku. Kromě práce s čítači, tvorby a následné změny třídy řídící vzorkování tak nedochází sběrem dat k výraznému množství operací navíc.

Režijní náklady spojené s čítači nejsou zanedbatelné, používáme proto více funkcí hledání zastínění. Během výpočtu je používána varianta podle cíle vnitřní proměnné ukazatele na funkci, který je nastavován po stavbě a provedení optimalizace hierarchie. I když jsou, až na inkrementaci čítačů, funkce identické, eliminuje se tím kontrola fáze výpočtu s každým testem.

3.3.2 Sběr dat

K získání představy o rozdělení paprsků ve scéně dochází ve fázi sběru dat k ukládání počtu zásahů způsobených vzorkovými paprsky. Může být zřejmé, že pro lepší výsledky je vhodné ukládat nejen zásahy s uzly hierarchie, ale také s primitivy. Pro přesné výsledky je zapotřebí při inkrementaci čítačů použití atomických operací, to však může přinášet pro některé multiprocesorové systémy značné režijní náklady navíc. Jak ale ukázali Gu a kol. [GHB15] pouze přibližný odhad je pro tento postup dostačující a odebrání atomicity zvýší výkon bez vlivu na kvalitu výstupní BVH. Ačkoli by bylo z hlediska přístupů do paměti výhodnější udržovat každý čítač přímo v uzlu, nezbývá na něj místo. Pro výchozí uzly použité v binární hierarchii máme k dispozici 8 bitů nevyužité paměti a 16 bitů udávajících počet odkazovaných primitiv použitých pouze jako značka ne-listu pro vnitřní uzly. Při uložení čítačů do volné paměti uzlu by přesto mohlo dojít k jejich přetečení v závislosti na počtu sledovaných paprsků, nehledě na uzly použité pro n-ární BVH, které nemají k dispozici žádnou volnou paměť. Čítače proto ukládáme zvlášť do samostatného pole.

Obě z metod optimalizace jsou zaměřeny na zrychlení pro dotazy se stínovými paprsky, z tohoto důvodu provádíme inkrementaci čítačů zásahu pouze během testů zastínění. V případě provedení kontrakce na základě dat získaných i z primárních a odražených paprsků, nebo pouze z nich, dochází ke snížení počtu jejich traverzací o až 0.9 %, ve stejném důsledku ale dojde ke zvýšení počtu traverzací pro stínové paprsky o průměrně 4.8 %. Jelikož pro jiné než stínové paprsky používáme optimalizaci s předpočítaným pořadím průchodu (viz sekce 3.4, snížení je důsledkem pouze menšího počtu traverzovaných uzlů (počet testů průsečíku s primitivy zůstává stejný). Pro stínové

paprsky je ale pořadí průchodu ovlivněno nevhodnými daty a vede tak k neoptimálnímu pořadí a znatelně horším výsledkům.

Během sběru dat je možné, že pro běžný způsob traverzace nebude nalezen potenciálně často zastiňující objekt uzavřený mezi dalšími primitivy. Testování všech primitiv by tento problém vyřešilo, je ale výpočetně příliš náročné. Navrženým řešením je zamíchání synů před jejich uložením na zásobník [ODJ16]. Jsou tím přidány další režijní náklady, pro některé případy ale může dojít k redukci traverzačních kroků.

Zamíchání spoléhá na náhodných číslech, jejichž generování může být výrazně náročná operace. Použití standardní verze generování pseudonáhodného čísla v jazyce `C++` za pomoci funkce `std::rand()` je velice pomalé a pro naše účely zcela nepoužitelné. Součástí PBRT je již implementovaná třída PCG generátoru pseudonáhodných čísel [O'N14], který nejen úspěšně prochází řadou rigorózních statistických testů, ale hlavně disponuje i vysokou rychlostí, což nám umožňuje praktické použití náhodného zamíchání.

3.3.3 Kontrakce podle distribuce paprsků ve scéně

Pro druhý způsob přestavbu BVH kontrakcí řízený podle distribuce paprsků ve scéně (Ray-Distribution Tree Contraction - RDTC) používáme pro každý uzel čítač udávající počet průchodů uzlem vzorkovými paprsky. Na základě těchto statistických dat uložených v každém z uzlů, resp. na odpovídajících pozicích v poli čítačů, určujeme zda-li je kontrakce výhodná a popřípadě který ze synů kontrakci podlehne. Samotné hodnoty jsou získány z předcházející fáze sběru dat a postup přestavby opět odpovídá postupu alg. 1 s parametry definovanými jako:

$$\alpha_N = \frac{hitCount(N)}{hitCount(N.parent)}$$

$$Select(S) = s_{max} \quad \text{s.t. } s_{max} \in s \text{ and } \alpha_{s_{max}} \geq \alpha_s \; \forall s \in s$$

$$ContractionCriterion(S) = \begin{cases} True & \text{if } \delta(s_{max}) > 0 \text{ and } \textit{not } N.parent.full \\ False & otherwise \end{cases},$$

kde $hitCount(N)$ je počet průchodů uzlu N během fáze sběru dat a $N.parent.full$ značka, zda-li obsahuje otec N již maximální počet potomků. Stejně jako v předešlých metodách používáme k výběru uzlu podléhajícího kontrakci prioritní frontu řazenou podle hodnot pravděpodobnosti α.

Řízení kontrakce podle čítačů zásahu přináší značně přesnější řešení nevyváženosti stromu hierarchie s ohledem na dotazování do ní. Upřednostněním kontrakce v uzlech, které jsou více navštěvovány a tedy i důležitějších z hlediska distribuce paprsků, zatímco méně důležité jsou spíše zanedbány.

V případě, že nebudeme dále provádět optimalizaci zaměřenou na průchod se stínovými parpsky, poskytuje tato metoda sama dobré řešení. V každém

uzlu už totiž známe jeho přibližnou pravděpodobnost průchodu stínovým paprskem. Během traverzace tak stačí postupovat od největších hodnot čítačů počtu průchodů.

3.3.4 Seřazení synů pro testy zastínění

V případě testů zastínění stačí zjistit, zda-li se na trase paprsku vyskytuje objekt pro který dojde k absorpci sledovaného paprsku. V případě zásahu takového objektu lze traverzaci rovnou zastavit. Je tak zřejmé, že právě pořadí traverzovaných uzlů výrazně ovlivňuje celkovou cenu testu a lze ji redukovat přeuspořádáním synů. V předešlých metodách využíváme řazení na základě velikosti, které správně předpokládá větší počet zastínění pro velké objekty, neuvažuje ale s distribucí paprsků ve scéně a může tak dojít k prioritnímu průchodu objektů, u kterých dochází pouze k malému počtu zásahů. Metoda kontrakce z předchozí sekce tento problém řeší na základě získané statistiky sledování vzorkových paprsků, nebere ale v potaz hodnoty v podstromech a zda-li v nich k zastínění skutečně došlo, nebo sledovaný paprsek pokračoval do dalšího z potomků.

Při dotazování na nejbližší průsečík s paprskem nám změna pořadí synů může výsledek ovlivnit. Pokud zrovna není pro průchod použito řazení ve směru paprsku, tedy řazení podle vzdálenosti, musíme přeuspořádání brát v potaz. Řešením se dále zabýváme v sekci 3.4.2.3.

Cena testu zastínění lze rekurzivně určit jako

$$C_N = \begin{cases} \sum_{i=1}^{n_{N.child}} \left(C_B + \alpha_i C_i \prod_{j=1}^{i-1}(1-\beta_j)\right) & \text{pro vnitřní uzly} \\ n_{N.primitiv} C_P & \text{pro listy} \end{cases}, \tag{3.2}$$

kde C_i je cena traverzace i-tého syna N, β_i pravděpodobnost zásahu primitiva paprskem v podstromě i-tého syna, $n_{N.primitiv}$ je počet primitiv pro list N a C_P cena testu průsečíku paprsku s primitivem (zjednodušena předpokladem stejné ceny, nehledě na typ primitiva). Jak ukázali S. Ogaki a A. Derouet-Jourdan [ODJ16], seřazení synů podle

$$score(i) = \frac{\beta_i}{\alpha_i C_i} \tag{3.3}$$

minimalizuje celkovou cenu. K určení obou pravděpodobností α_i a β_i využijeme statistiku získanou vzorkovými paprsky jako

$$\begin{aligned} \alpha_i &= hitCount(i)/hitCount(N), \\ \beta_i &= O_i/hitCount(N). \end{aligned}$$

Hodnota O_i udává počet paprsků, pro které došlo k průsečíku s primitivem v podstromu i.

Možnou alternativou, která nepotřebuje tolik paměti je nepoužití čítačů pro uzly hierarchie a odhadnout pravděpodobnost traverzace každého ze synů

konstantou. s β_i podle počtu zásahů primitiv v podstromu syna i vůči celému podstromu otce vzorkovými paprsky tak získáváme

$$\alpha_i = hitCount(i)/hitCount(N),$$
$$\beta_i = O_i / \sum_{j=1}^{n_{N.child}} O_j.$$

Volbou $\alpha_i = 1.0$ je možné provést optimalizaci se zaměřením na nejhorší případ, kdy paprsek vždy zasáhne všechny syny. Tento přístup má ale nižší přesnost a v důsledku i výkon.

3.3.5 Jednofázová optimalizace

K získání maximálního zlepšení pro traverzaci stínových paprsků zkombinujeme obě pohledově závislé metody dohromady. I když jsou obě zaměřeny na snížení počtu nutných traverzací, každá redukuje jiný zdroj způsobující zbytečný výpočet navíc díky čemuž je možné oba způsoby efektivně zkombinovat bez vzájemně negativního vlivu. Obecně přestavba struktury na n-ární přináší menší počet traverzačních kroků, zvolená metoda kontrakce ale navíc upřednostňuje části podstromu s velkým počtem traverzací a výrazně tak snižuje hloubku pro často navštěvované části hierarchie. Na druhou stranu metoda určení pořadí traverzace synů má za cíl zaručit pokračování do částí podstromu kde skutečně dochází k nalezení průsečíku paprsku s primitivem a naopak oddalující traverzaci synů, ve kterých k nalezení průsečíku obvykle nedochází.

Z důvodu rychlejší optimalizace provádíme obě metody naráz. Využijeme k tomu rekurzivní postup kontrakce směrem shora-dolu, během které jsou uzly přestavovány do nového pole hierarchie. Po zpracování všech synů výsledného uzlu namísto návratu z rekurze ještě provedeme jejich přeuspořádání na základě jejich seřazení podle vypočteného skóre (3.3). Pseudokód algoritmu viz alg. 2.

Abychom nemuseli určovat pořadí průchodu synů pro stínové paprsky za běhu programu a zároveň si tak nemuseli pamatovat skóre každého uzlu, provedeme zmíněné přeuspořádání synů v poli hierarchie. V důsledku se tím ale poruší ideální uložení v paměti, které by odpovídalo stejnému uspořádání jako je průchod prohledávání do hloubky. Jelikož při přeuspořádání potomků postupujeme zdola-nahoru a podstromy synů již musejí být zpracovány, není tak možné cílené uspořádání zachovat. Na závěr proto provedeme ještě jednu rychlou přestavbu zachovávající změněné pořadí synů, ale s přesunutím jejich podstromů na správná místa v paměti.

3.3.6 Dvoufázová optimalizace

Na rozdíl od jednofázové optimalizace provádíme sběr dat ke každé z použitých metod zvlášť. Po prvním sběru dat provedeme pouze kontrakci uzlů s pořadím

```
input  : Přestavovaný uzel N
output: Počet průsečíků s primitivy v podstromě uzlu N
output: Výsledná traverzační cena uzlu N
1  Function int Optimize(BVHNode N, out float cost) is
2      if N.isLeaf then
3          cost = n_{N.primitive};
4          return SumPrimitiveHits(N);
5      else
6          primitiveHits = 0;
7          S ← BVHContract(N);
8          foreach s ∈ S do
9              primitiveHits += Optimize(s, s.cost);
10             s.score = CalculateScore(s, N);
11         end
12         SortChildren(N, S);
13         cost = 1 + CalculateCost(S, primitiveHits);
14         return primitiveHits;
15     end
16 end
```

Algoritmus 2: Pseudokód jednofázové optimalizace podle distribuce paprsků ve scéně. Funkce $SumPrimitiveHits()$ vrací součet zásahů primitiv vstupního listu, $CalculateScore()$ vypočte skóre syna podle rov. (3.3), na základě kterého jsou následně seřazeni funkcí $SortChildren()$. Výstupní cena traverzace zpracovaného uzlu je nakonec vypočtena funkcí $CalculateScore()$ podle rovnice (3.2) s výsledkem zvýšeným o 1 reprezentují cenu testu průsečíku s obalovým tělesem.

synů pro průchod stínových paprsků podle hodnot čítačů průchodu. s touto přestavěnou n-ární BVH je následována druhá fáze sběru dat, ze které jsou získaná data použita ke konečnému přeuspořádání uzlů i primitiv.

Prvotní část s kontrakcí probíhá zcela stejně, včetně získávání statistických dat paprsků. V důsledku je rozložení stromu hierarchie téměř identické s jednofázovou verzí a stejně tak celkový počet traverzačních kroků během hledání nejbližšího průsečíku s paprskem při použití stejné metody určení pořadí průchodu. Drobné změny v měření jsou způsobeny převážně zamícháním vkládaných uzlů na zásobník během traverzace vzorkových paprsků.

3.4 Určení pořadí průchodu pro primární a odražené paprsky

Pro n-ární hierarchie nastává další problém a to určení pořadí průchodu synů vnitřních uzlů.

Ačkoli je možné pořadí průchodu pevně určit již při stavbě hierarchie a s využitím různých heuristik docílit redukce počtu traverzačních kroků (viz pře-

dešlé sekce 3.2 a 3.3), jedná se o optimalizaci vhodnou pouze pro dotazování se stínovými paprsky. Pro dotazování na nejbližší průsečík totiž nemusí být toto pořadí vůbec optimální vzhledem k právě sledovanému paprsku, resp. je vhodné pouze pro určité rozmezí směrů, kde pořadí průchodu odpovídá vzdálenostem zásahů s obalovými tělesy jednotlivých synů (nezasáhnutí synové mají vzdálenost nastavenu na t_{max}). Obecně tak nedochází tolik k prořezávání hierarchie a je třeba procházet větší množství synů, jelikož se v nich stále může nacházet bližší průsečík s paprskem než dosavadní nalezený.

Jednou z možných metod se tak rovnou nabízí řazení podle vzdálenosti k obalovému tělesu. Ačkoli je tato metoda ideální pro redukci celkového počtu traverzačních kroků, je výpočetně náročnější a celkový čas výrazně stoupne.

Jako možná řešení jsme navrhli dvě metody, jejichž základem je využívání předpočítaných pořadí průchodu uzlu vhledem ke sledovanému paprsku. V obou variantách mohou být pořadí, kromě zrcadlové symetrie, vzájemně zcela nezávislá a libovolná. Pro 4-ární BVH tak dostáváme pro každé pořadí $4! = 24$ možností, což je možné zakódovat do 8 bitů vzhledem k jednotlivým ofsetům z intervalu $\{0, \dots, 3\}$, nebo do 5 bitů, nechceme-li si pamatovat pořadí přímo v uzlu, ale získávat ho například indexem do jednoduché tabulky předpřipravených lexikografických uspořádání.

3.4.1 Předpočítané pořadí pro směr paprsku

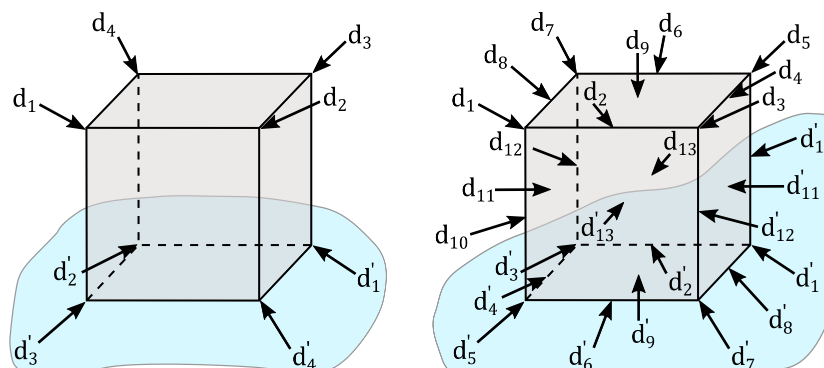
První metoda nazvaná *dirLUT* spočívá v předpočítání pořadí vzhledem ke konečnému množství směrů paprsků pro každý uzel. Během traverzace pro primární a odražené paprsky se pak spočte do jakého z předpočítaných směrů právě sledovaný paprsek spadá a určí tak které z pořadí bude použito. Pro konkrétní paprsek je směr a tedy i index použitého pořadí konstantní podél celé délky paprsku. K jeho výpočtu proto dochází pouze jednou na každý dotaz, stejně tak se pouze jednou určí, zda-li se jedná o zmíněný zrcadlový směr k nějakému předpočítanému a v takovém případě se opačným pořadím řídí celá traverzace, bez zbytečných kontrol u každého navštíveného uzlu.

3.4.1.1 8 směrů paprsku

Při použití osmi předpočítaných směrů (viz obr. 3.2) je třeba v každém uzlu využít alespoň $4 * 5 = 20$ bitů paměti pro udržení všech pořadí.

Každý uzel zachováváme na velikosti 32 bajtů kvůli zarovnávání ve vyrovnávací paměti. S obsahem pro vnitřní uzly využívajících 24 bajtů pro obalové těleso, $4B$ index na prvního syna v poli uzlů hierarchie a minimálně 2 bity pro počet synů. Zůstává tak k dispozici 30 bitů v každém uzlu a využijeme-li celých 8 bitů pro počet synů, kvůli snadnější práci s ním, zůstane tak volných pouze 24 bitů.

Ačkoli by se předpočítaná pořadí do uzlu vešla, je výpočetně výhodnější využít pro každý směr zarovnanou hodnotu na 8 bitů a umožnit tak práci



Obrázek 3.2: Ukázka reprezentativních směrů k předpočítání pořadí průchodu synů při traverzaci BVH. Pro 4 a 13 směrů paprsků společně se směry symetrie při průchodu v opačným pořadí v modrém poli.

s nimi bez nutnosti použití operací jako bitový posun a maskování. Celkem tedy použijeme $4*8 = 32$ bitů, čímž bychom ale přesáhli požadovanou velikost uzlu. Jelikož ale využíváme pouze 5 z 8 bitů indexem do tabulky uspořádání, není tak problém tabulku zvětšit. Přesuneme proto do ní hodnotu počtu synů a index v uzlech tak bude ukazovat nejprve na počet synů následovaný jejich seřazením.

Tabulku je navíc potřeba rozšířit pro uzly, které nejsou zcela zaplněny a mohou mít pouze tři, popřípadě dva syny. Celkem tak tabulka bude mít $4! + 3! + 2! = 32$ řádků. Samotná pořadí je pro 4-ární hierarchii možné uložit do jediného bajtu tak, že by ofsetům pro jednotlivé syny vždy odpovídala dvojice bitů. Ofsety přesto uložíme zvlášť do samostatných hodnot pro jejich snazší získávání. Jelikož se bude k této tabulce přistupovat často, bude s velkou pravděpodobností udržována během celého výpočtu v cache paměti. Z tohoto důvodu chceme, aby byla co nejmenší. Přesto však tabulku zmíněným způsobem rozšíříme, jelikož úspora eliminací nutných operací k získání konkrétní hodnoty je ve výsledku výhodnější i se zvýšením možných cache miss přístupů zmenšením zbylého prostoru.

Konečná modifikace je posun celé tabulky o jeden prvek. Výsledkem tak bude, že každé z pořadí bude mít index alespoň jedna a jednoduše tak lze rozlišit vnitřní uzel od listu kontrolou libovolného z pořadí, resp. části paměti nepoužívané listem pro jiné vnitřní proměnné a připadající pro vnitřní uzel některému z pořadí.

Celková velikost tabulky bude

$$5 * (4! + 3! + 2!) + 1 = 161$$

bajtů. Lze tak uložit do hodnot pořadí v uzlu rovnou index na první prvek (počet synů), namísto indexu na pomyslný řádek s požadovaným pořadím čímž

se zbavíme nutného násobení s každým přístupem.

Jelikož není neobvyklé, že by paprsek zcela minul obalové těleso některého ze synů, použijeme pro určení pořadí namísto něj vzdálenost od rohu obalového tělesa otce ke středům obálek synů. Vybraný roh odpovídá nejbližšímu k bodu skrze který by reprezentativní paprsek pro předpočítávaný směr pronikl do obalového tělesa otce, kdyby byl posunutý tak, že by mířil do jejího středu. Otestováno bylo také řazení podle minimální vzdálenosti zvoleného rohu obálky otce k obálkám synů, v průměru však vycházelo o 4.3 % (1.31 % až 7.88 %) traverzačních kroků více, oproti řazení se středy obalových těles synů.

3.4.1.2 Zvýšení přesnoti

Pro přesnější určení pořadí zvýšením předpočítaných směrů např. na 26 (13 se zrcadlovou symetrií, viz obr. 3.2) již není možné uchovat pro každý směr pořadí přímo v uzlu, jelikož by bylo zapotřebí alespoň 75 bitů paměti, zatímco máme k dispozici pouze zmíněných 32 bitů, včetně prostoru pro počet synů.

Řešením by tak mohl být odkaz do tabulky, kde by každý sloupec odpovídal jednomu z předpočítaných směrů paprsku a řádek konkrétní kombinaci seřazení k jednotlivým směrům. Jelikož jsou ale na sobě předpočítaná pořadí nezávislá, vyhledávací tabulka by tak pro tento případ obsahovala 24^{13} řádků, což je nejen jistě více jak celkový počet uzlů, ale i mnohonásobně více, než by se vešlo do paměti počítače.

Chceme-li zachovat velikost uzlu 32 bajtů, není tak možné docílit větší přesnosti zvýšením předpočítavaných směrů paprsku.

3.4.2 Předpočítání pořadí z konstrukce hierarchie

Druhá navržená metoda využívá způsob, jakým byla použitá 4-ární hierarchie zkonstruována s využitím výchozího binárního BVH.

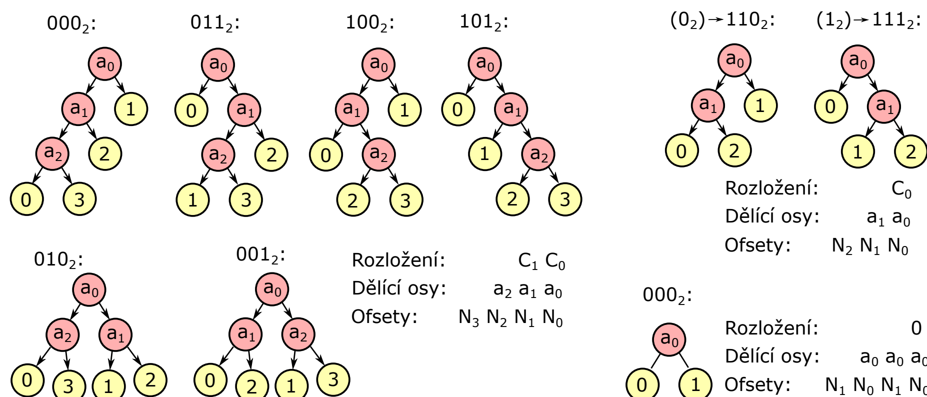
Každý vnitřní uzel přestavované binární hierarchie dělí obsažená primitiva do levého a pravého syna na základě výběru dělící osy, ve které má projekce centroidů na jednotlivé osy největší rozpětí, a její pozice, tedy rozdělení množiny primitiv do synů, podle zvolené metody stavby. Během traverzace pak lze pouze ze směru paprsku ve vybrané dělící ose určit, který ze synů by mělo být výhodnější procházet dříve. Jedná se o velice rychlý test a ačkoli nemusí vždy dát správné pořadí, rozhodli jsme se stejného principu využít i v naší metodě.

Dělící osy pro uzel výsledné 4-ární hierarchie získáváme během jeho stavby. Jednu dělící osu má vždy, již jako vstupní binární uzel. Další pak získává během kontrakce některého ze synů, zkopírováním z mazaného uzlu. Může proto obsahovat až tři dělící osy $\{a_0, \ldots, a_2\}$.

Kromě jejich zapamatování je ale nutné umět stále určit, na které z nynějších synů se konkrétní osa vztahuje. Jelikož uzel podléhající kontrakci může být libovolný ze synů, musíme si zapamatovat i další hodnotu, pomocí které

by jsme mohli toto rozložení určit. Tuto hodnotu (*node layout*) získáme opět během kontrakce, konkrétně ofsetem syna, který kontrakci podlehl s tím, že pro vstupní binární uzel má levý syn ofset 0 a pravý 1. Pro snazší implementaci během rozšiřování uzlu ofsety synů nepřečíslujeme, ale pouze nahradíme kontraktovaný uzel jeho levým potomkem a pravého umístíme na první nevyužitý ofset.

Výsledný uzel můžeme reprezentovat jako plný binární strom, kde vnitřní uzly odpovídají dělícím osám a listy synům. Během traverzace tohoto uzlu tak pomyslně dochází k průchodu tímto stromem, kde pro vnitřní uzly pokračuje průchod do levého potomka, je-li směr sledovaného paprsku v dané ose kladný, popřípadě do pravého potomka je-li záporný. Pořadí pro zbývající syny pak lze určit stejně, jako by tomu bylo u průchodu do hloubky (*DFS*). Hodnoty rozložení s příslušnými stromy uzlu jsou vyneseny do obr. 3.3 s odpovídajícím získaným pořadím vůči průchodu stromem v tabulce 3.1.



Obrázek 3.3: Rozložení uzlů s příslušnými reprezentativními grafy. Synové uzlu leží v listech s klíčem představujícím jejich pozici (ofset v poli uzlů hierarchie od pozice prvního syna) a použité dělící osy v vnitřních uzlech stromu

Oproti metodě řazení tento způsob potřebuje pouze minimální množství operací navíc. Osy kopírujeme z výchozích binárních uzlů, rozložení uzlu získáme během kontrakce a pouze konečné přeuspořádání (blíže popsáno v podsekci 3.4.2.3) je třeba spočítat. Pro získání přeuspořádání proto pracujeme s n-ticí hodnot namísto pouhých ofsetů, kde jedna z hodnot nám udává původní pozici syna. Potřebný výsledek je tak možné získat pouze lineárním průchodem výsledného pole n-tic u každého uzlu.

3.4.2.1 Paměť uzlu

Stejně jako bylo popsáno v předešlé metodě chceme uchovat uzly hierarchie na 32 bajtech paměti, zároveň ale použijeme celých 8 bitů pro hodnotu počtu synů, čímž nám zůstává pouze 24 bitů k dispozici.

Rozložení	Průchod vůči uspořádané množině vnitřních uzlů (a_2, a_1, a_0)							
	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
0_2	0,3,2,1	1,0,3,2	2,0,3,1	1,2,0,3	3,0,2,1	1,3,0,2	2,3,0,1	1,2,3,0
1_2	0,3,1,2	1,2,0,3	0,3,2,1	2,1,0,3	3,0,1,2	1,2,3,0	3,0,2,1	2,1,3,0
10_2	0,2,1,3	1,3,0,2	2,0,1,3	1,3,2,0	0,2,3,1	3,1,0,2	2,0,3,1	3,1,2,0
11_2	0,1,3,2	1,3,2,0	0,2,1,3	2,1,3,0	0,3,1,2	3,1,2,0	0,2,3,1	2,3,1,0
100_2	0,2,3,1	1,0,2,3	2,3,0,1	1,2,3,0	0,3,2,1	1,0,3,2	3,2,0,1	1,3,2,0
101_2	0,1,2,3	1,2,3,0	0,2,3,1	2,3,1,0	0,1,3,2	1,3,2,0	0,3,2,1	3,2,1,0
110_2	0,2,1	1,0,2	2,0,1	1,2,0	0,2,1	1,0,2	2,0,1	1,2,0
111_2	0,1,2	1,2,0	0,2,1	2,1,0	0,1,2	1,2,0	0,2,1	2,1,0

Tabulka 3.1: Pořadí průchodu pro syny vnitřních uzlů hierarchie s ohledem na jejich rozložení a průchod paprsku vůči použitým dělícím osám.

Pro každý uzel navíc budeme potřebovat:

- 3 bity pro rozložení uzlu
- 5 bitů na dělící osy
- 8 bitů pro přeuspořádání synů

Všem třem množinám hodnot proto přidělíme 8 bitů, čímž splníme limit velikosti uzlu. Konkrétní hodnoty pak budou vždy uloženy od nejméně důležitých bitů podle potřebné velikosti. Výjimkou je hodnota pro dělící osy, které nabývají hodnot mezi 0 až 2 a ačkoli by mohlo být výhodnější uložit je vždy po dvojici bitů s jejich získáváním pomocí bitového posunu a konjunkce, zvolíme pro jejich uložení vzorec $9 * a_2 + 3 * a_1 + a_0$, kde a_2, resp. i a_1, jsou rovny nule, má-li uzel pouze tři, resp. dva syny. Důvodem je použití této hodnoty jako indexu do nahlížecích tabulek, ušetříme v nich tak prostor menším rozsahem pro přímou indexaci.

3.4.2.2 Předpočítání pořadí

Jelikož by bylo výpočetně náročné vypočítávat pořadí průchodu z hodnot rozložení uzlu, dělících os a směru paprsku při každém průniku paprsku s uzlem, předpočítáme si pořadí předem do tabulky.

Získávané pořadí bude záležet na rozložení uzlu a samozřejmě na směru paprsku. Možných rozložení uzlu je $2 * 3 = 6$ pro uzel se všemi čtyřmi syny, 2 pro uzel s třemi syny a pouze jediné pro uzel s dvěma syny. Díky tomu, že budeme těmito hodnotami přistupovat do přepočítané tabulky a potřebujeme unikátní index pro různá rozložení, přenastavíme hodnoty v uzlech s třemi syny z 0_2 na 110_2 a 1_2 na 111_2. Uzel se dvěma syny ale necháme stejný (0_2) a namísto toho zopakujeme jeho dělící osu do všech tří pozic a ofsety synů zkopírujeme na zbylé pozice odpovídající synům 2 a 3, čímž se tento typ

uzlů během traverzace bude chovat jako plný uzel s pouze dvěma možnými pořadími. Ušetříme tím místo pro předpočítané hodnoty jednoho z rozložení.

Dělící osy v uzlu a jeho rozložení jsou nezávislá, nelze proto předpočítávat tabulku vzhledem ke směru paprsku, ale je třeba ji určit vzhledem ke konkrétnímu průchodu zmíněným stromem uzlu. Druhým indexem do tabulky pořadí tak bude až trojice hodnot 0 nebo 1 určujících průchod do levého či pravého syna vzhledem ke směru paprsku a použité dělící ose v uzlu.

Stejně jako v předchozí metodě používáme pro každý ofset v tabulce samostatnou hodnotu, ačkoli by bylo možné vložit celé pořadí do jedné 8 bitové hodnoty. Výsledná tabulka tak má velikost

$$(6 + 2) * 2^3 * 4 = 256$$

bajtů.

Možnou nevýhodou je nutnost počítání průchodu stromem uzlu ze směru paprsku a použitých dělících os pro získání indexu do tabulky průchodu. Navrhli jsme proto druhou nahlížecí tabulku eliminující tento výpočet, rozšířenou ke každému rozložení uzlu o možné kombinace dělících os, zvětšující ji tak celkem 27 krát. Uplatňujeme navíc zmíněnou zrcadlovou symetrii pořadí průchodu pro paprsky v opačném směru, resp. pro paprsky procházející strom uzlu v opačném pořadí. Velikost konečné tabulky (v kódu nazvané *masterLUT*) tak činí

$$\frac{(6 + 2) * 27 * 2^3 * 4}{2} = 3456$$

bajtů.

3.4.2.3 Změna pořadí synů přeuspořádáním

V případě, že dojde k přeuspořádání synů v uzlu při optimalizaci pro stínové paprsky, zneplatní se tím předpočítané pořadí v nahlížecí tabulce. Z tohoto důvodu je třeba si v uzlu pamatovat, které pozici během konstrukce náleží jaký ofset v lineárním poli hierarchie od pozice prvního syna. Kdyby nedošlo k žádnému přeuspřádání, ofset do pole uzlů hierarchie by odpovídal pozici při konstrukci.

K přeuspořádání dochází již během konstrukce jako část optimalizace pro stínové paprsky. Stejně jako výběr uzlu podléhajícího kontrakci, pokračuje rekurze konstrukce hierarchie v pořadí stejné prioritní fronty. Pokud pak nejsou uzly opět přeuspořádány z nasbíraných dat (metoda optimalizace ze sekce 3.3.4), mají pořadí pro průchod stínových paprsků určené velikostí ploch jejich obalových těles (metoda optimalizace 3.2).

Z nedostatku místa v uzlu, uložíme přeuspořádané ofsety jako jednu 8 bitovou hodnotu tak, že směrem od nejméně důležitých bitů reprezentuje pozice po dvojicích bitů hodnoty pozic synů při konstrukci uzlu a hodnota udává ofset od pozice prvního syna do pole uzlů hierarchie. Během traverzace se tak při

získání požadovaného pořadí nejprve podíváme na odpovídající pozici v hodnotě pořadích, odkud získáme potřebnou hodnotu ofsetu uzlu pro průchod.

Jelikož je potřeba pro každé získání hodnoty ofsetu z hodnoty pořadí v uzlu provést alespoň jeden bitový posun a logickou konjunkci (viz alg. 3) a přitom se jedná pouze o jednoduchou permutaci, nahradíme celou hodnotu opět indexem do tabulky nazvanou *traveseOrderLUT*. Protože ale používáme hodnotu přeuspořádání k rozlišení mezi vnitřními uzly a listy, posuneme navíc tabulku o jednu hodnotu tak, aby indexy začínali od jedné. Rozlišování uzlů se tím, oproti verze bez tabulky, nezmění. Výsledná tabulka pro získávání konkrétních ofsetů pro procházené syny má tak velikost $4! * 4 + 1 = 97$ bajtů.

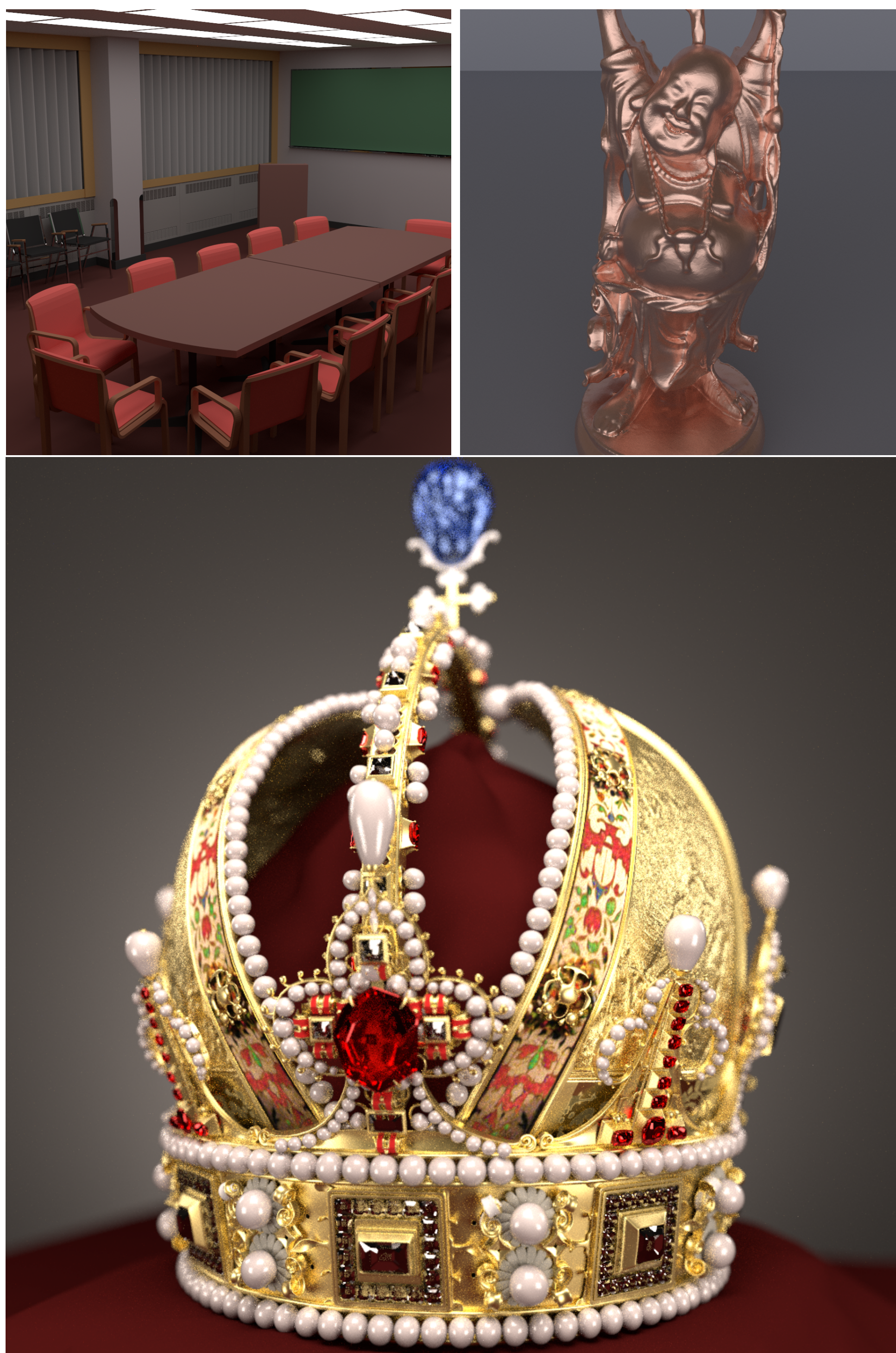
```
1  int childrenOffset;    // index na prvního syna v poli uzlů hierarchie
2  uint8_t childOrder; // přeuspořádání synů
3  int orderLUTindex; // index do tabulky podle směru paprsku a rozložení uzlu
4  uint8_t chilnNo;      // pozice získávaného syna
5  uint8_t nodeIndex; // výsledný index požadovaného syna
6  // Bez využití tabulky:
7  begin
8  |  nodeIndex = childrenOffset +
   |    (node.childOrder ≫ masterLUT[orderLUTindex + chilnNo]) & 0x3);
9  end
10 // S přístupem do tabulky:
11 begin
12 |  nodeIndex = childrenOffset +
   |    traveseOrderLUT[childOrder + masterLUT[orderLUTindex + chilnNo]];
13 end
```

Algoritmus 3: Srovnání metody s a bez nahlížecí tabulky po provedení permutace synů v uzlu. Výsledkem je indexu do pole uzlů hierarchie pro syna s pozicí vůči nepermutovanému pořadí podle vstupního parametru *childNo*. Metoda získávající pořadí z hodnoty uložené v uzlu, má hodnoty tabulky *masterLUT* vynásobeny předem dvěma, z důvodu posunu na příslušnou dvojici bitů. Operace ≫ představuje bitový posun doprava o počet pozic rovný pravé straně operátoru.

3.5 Výsledky

Měření zrychlení implementovaných metod jsme provedly na množině pěti různě velkých scén s porovnáním časů a počtů provedení optimalizovaných částí programu vůči původní variantě, která je součástí softwaru PBRT. Přehled scén s počty primitiv a velikostí výchozích BVH nad nimi postavených je umístěn v tabulce 2.1. Ukázka výsledných obrazů testovaných scén je vidět na obrazech 3.4 a 3.5.

Testování jsme provedli v několika částech. Každá se zaměřením na jednu konkrétní úlohu nebo metodu za účelem určení vhodných změn a alterna-



Obrázek 3.4: Ukázka výstupů pro scény *Conference room*, *Buddha* a *Crown* za použití 256 vzorků na pixel a metodou syntézy obrazu *path*.

Obrázek 3.5: Ukázka výstupů pro scény *San Miguel* a *Ecosystem* za použití 256 vzorků na pixel a metodou syntézy obrazu *path*.

tiv v jejím postupu. V první části (podsekce 3.5.1) se zaměříme na určení pořadí průchodu potomků během hledání nejbližšího průsečíku s paprskem a metody průchodu využívající řazení oproti navržené metodě s nahlížecí tabulkou. Všechny tři porovnávané postupy využívají přestavbu kontrakcí podle plochy na n-ární BVH společně s optimalizací průchodu stínových paprsků přeuspořádáním potomků také na základě plochy obálek. Kromě času jsou proto porovnávány pouze počty testů při hledání nejbližšího průsečíku, jelikož pro stínové paprsky jsou hodnoty stejné.

Následují dvě podsekce zaměřené čistě na metodu s nahlížecí tabulkou. V první (podsekci 3.5.2) porovnáváme způsoby předpočítání hledaného pořadí s variantami podle definovaných směrů paprsků a s využitím konstrukce BVH. V další podsekci 3.5.3 následuje porovnání varianty s jednou tabulkou a varianty určující pořadí přes dvojici tabulek. Obě z těchto částí jsou porovnány s použitím pohledově závislé optimalizace, tedy přestavbou BVH na n-ární i přeuspořádání potomků na základě distribuce paprsků ve scéně.

V závěrečné části 3.5.4 nakonec porovnáváme nejlepší z variant pohledově závislé optimalizace a optimalizace podle plochy společně s původní implementací využívající binární BVH.

Pro sjednocení označení různých metod a jejich kombinací nejprve uvedeme stručný přehled se shrnutím jejich postupu, počínaje metodami určujícími pořadí průchodu potomků při zpracování dotazů na nejbližší průsečík:

- sort: řazení potomků na základě vzdáleností průsečíku s jejich obalovým tělesem.

- nearest: pokračování průchodu nejbližším synem, zbytek potomků je pouze částečně seřazen v průběhu vkládání na zásobník traverzace.

- LUT: metoda předpočítání pořadí na základě konstrukce 3.4.2 s jednou nahlížecí tabulkou. Podle směru sledovaného paprsku a hodnot uložených v uzlu během konstrukce je z nahlížecí tabulky vybráno předpočítané pořadí průchodu potomků.

- dirLUT: během konstrukce se provede předpočítání pořadí pro definovanou množinu směrů paprsků. na základě sledovaného paprsku se následně vybírá konkrétní pořadí v každém z uzlů.

Konečné kombinace metod optimalizující průchod stínovými paprsky i dotazování na nejbližší průsečík:

- *SAO* (Surface Area Optimized)

  Kombinace dvojice metod SATC 3.2.1 a SATO 3.2.2 provádějící přestavbu binární hierarchie kontrakcí a následné přeuspořádání synů pro průchod stínovými paprsky na základě plochy obalových těles. K určení pořadí průchodu synů při hledání nejbližšího průsečíku je použita metoda LUT.

- *1Phase*

  Jednofázová, podhledově závislá optimalizace. Přestavba BVH i přeuspořádání potomků je prováděno na základě distribuce paprsků ve scéně získané z jedné fáze sběru dat. K určení pořadí průchodu potomků při hledání nejbližšího průsečíku s paprskem je také použita metoda LUT.

- Orig

  Nezměněná implementace z PBRT využívající binární BVH. Pořadí průchodu je pro stínové paprsky i dotazování na nejbližší průsečík určeno z dělící osy z konstrukce a směru paprsku v dané ose.

3.5.1 Výpočet pořadí za běhu programu

V případě určování pořadí průchodu při dotazu hledání nejbližšího průsečíku s paprskem je optimálním řešením projít syny odpředu-dozadu vzhledem ke směru paprsku, k čemuž se běžně používá řazení podle vzdálenosti průsečíku s obalovým tělesem. Kvůli značným výpočetním nákladů na řazení všech synů jsme navíc otestovali metodu částečného řazení nazvanou *nearest*, která jednou lineárně projde a testuje potomky pro nalezení nejbližšího z nich. Během tohoto průchodu může navíc docházet k posunutí dočasně nejbližšího z potomků více k vrcholu zásobníku, proto částečné řazení.

K odstranění režijních nákladů řazení jsme navrhli zmíněné metody pracující na základě nahlížecí tabulky (viz sekce 3.4), ze které lze podle směru sledovaného paprsku a hodnot předem uložených v uzlech určit téměř optimální pořadí průchodu jeho potomků. Výraznou změnou je navíc způsob testování existence lepšího řešení v uzlu, tedy test průsečíku paprsku s obalovým tělesem a porovnání vzdálenosti oproti nejbližšímu doposud nalezenému průsečíku paprsku s primitivem. Zatímco oba postupy používající řazení testují uzel před vložením na zásobník, jelikož je vzdálenost získávána zároveň s testem průsečíku, zbylé metody provádějí test až po jeho vyjmutí a tedy s možným blíže nalezeným průsečíkem, než v době vkládání na zásobník. Důsledkem je vyšší počet testů průsečíku pro metody řazení, i když se na počet traverzací jedná o optimální průchod.

Srovnání časů a počtu testů obou řadících metod společně s metodou *LUT* používající nahlížecí tabulku podle konstrukce je patrné z tabulky 3.2 a grafů 3.6. Porovnané metody dále využívají zmíněný postup pouze při hledání nejbližšího průsečíku s paprskem, pro testy zastínění je využita optimalizace podle plochy (viz sekce 3.2).

I přes časté ořezávání stromu BVH, je čas výpočtu obou metod výrazně horší, než u metody *LUT*. Pro metodu *nearest* sice dochází ke zlepšení u většiny testovaných scén ve srovnání s původní implementací z PBRT, přesto jsou režijní náklady navíc oproti předpočítané variantě s nahlížecí tabulkou výrazné, eliminující tak zisk menšího počtu traverzací. Průměrné hodnoty

Scene	Traverse order method	Relative runtime ratio	#ray-box tests [M]				#ray-primitive tests [M]			
			diff		shadow		diff		shadow	
			amount	ratio	amount	ratio	amount	ratio	amount	ratio
Conference	sort	1.20	43093	0.80	34569	0.81	2819	1.01	1399	0.97
	nearest	1.07	44963	0.83	34570	0.81	3294	1.18	1399	0.97
	LUT	1.00	42680	0.79	34570	0.81	2799	1.00	1399	0.97
Buddha	sort	0.92	8016	0.85	2035	0.63	493	1.00	107	0.66
	nearest	0.86	8659	0.92	2035	0.63	618	1.26	107	0.66
	LUT	0.81	8011	0.85	2035	0.63	492	1.00	107	0.66
Crown	sort	1.07	24891	0.83	33473	0.83	3036	1.03	4038	0.99
	nearest	0.94	26029	0.87	33473	0.83	3183	1.08	4038	0.99
	LUT	1.16	24079	0.81	33473	0.83	2935	1.00	4038	0.99
San Miguel	sort	1.12	168784	0.92	85431	0.56	9872	1.05	4971	0.66
	nearest	0.92	178695	0.98	89974	0.59	10523	1.12	5200	0.69
	LUT	0.81	156672	0.86	85431	0.56	9369	1.00	4971	0.66
Ecosystem	sort	1.81	35698	0.91	9095	0.89	2479	1.03	602	0.93
	nearest	1.25	38928	1.00	9095	0.89	2641	1.09	602	0.93
	LUT	0.84	35046	0.90	9095	0.89	2417	1.00	602	0.93

Tabulka 3.2: Porovnání metod *sort*, *nearest* a *LUT* určujících pořadí traverzace synů pro primární a odražené paprsky. Pro stínové paprsky je pořadí u všech metod určeno stejně, seřazením podle velikosti obalových těles synů. Naměřená data jsou zprůměrována z 16 měření a 4 metod syntézy obrazu s 64 vzorky na pixel. Počty testů jsou v milionech s *diff* udávající počet hledání nejbližšího průsečíku a *shadow* počet testů zastínění. Poměry srovnávají metody oproti původní implementaci z PBRT.

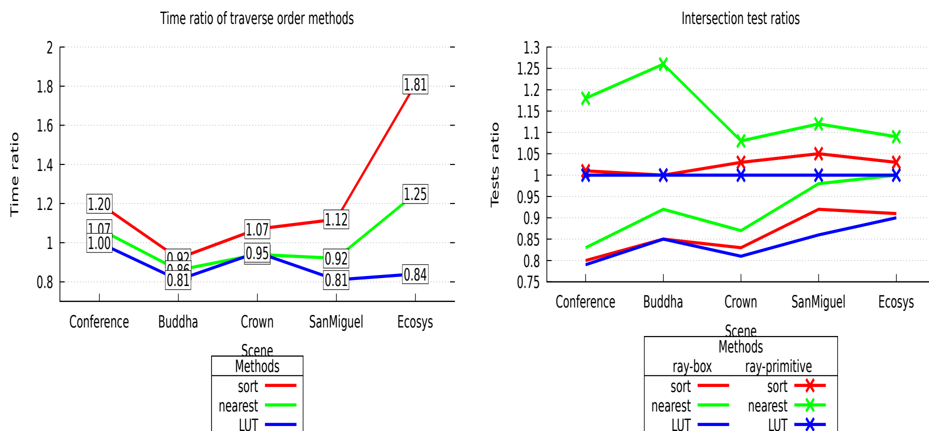
pro 16 měření s 64 vzorky na pixel jsou vepsány do tabulky 3.2 a vyneseny do grafů 3.6.

3.5.2 Jednofázová a dvoufázová optimalizace

Jak již bylo popsáno u dvoufázové metody 3.3.6, počet traverzací pro primární a odražené paprsky je mezi jednofázovou a dvoufázovou optimalizací identický, až na drobné změny způsobené randomizací hodnot a pořadí vkládaných uzlů na zásobník traverzace během sběru dat. V případě různého konečného proházení synů také nedochází ke změně. Důvodem je definované pořadí na základě zvolené metody průchodu, kterou je například řazení podle vzdálenosti průsečíku paprsku s obálkou, nebo získání pořadí z nahlížecí tabulky tak, aby traverzace mohla pokračovat co nejlépe vůči směru sledovaného paprsku.

Pro stínové paprsky ve srovnání s jednofázovou optimalizací ale, i přes přesnější způsob přeuspořádání synů, ke zlepšení nedochází. Důvodem jsou režijní náklady pro provedení druhé fáze sběru dat a samotný sběr za pomoci méně optimalizované BVH převyšují možný zisk tímto způsobem získaný.

Dalším důvodem srovnatelných výsledků obou metod je vysoká přesnost statistiky rozmístění paprsků ve scéně. Získaná posíláním velkého množství vzorkových paprsků, pro naši implementaci jeden na každý pixel, a provedením zamíchání pořadí průchodu synů.V případě sběru dat pouze z malého



Obrázek 3.6: Porovnání metod určujících pořadí traverzace řazením a nahlížecí tabulkou oproti původní implementaci nad binárním stromem hierarchie z PBRT. V levém grafu je poměr celkového času výpočtu každé z metod. V pravém poměr počtu provedených testů průsečíku paprsku s obalovým tělesem a primitivem v průběhu dotazů k nalezení nejbližšího průsečíku.

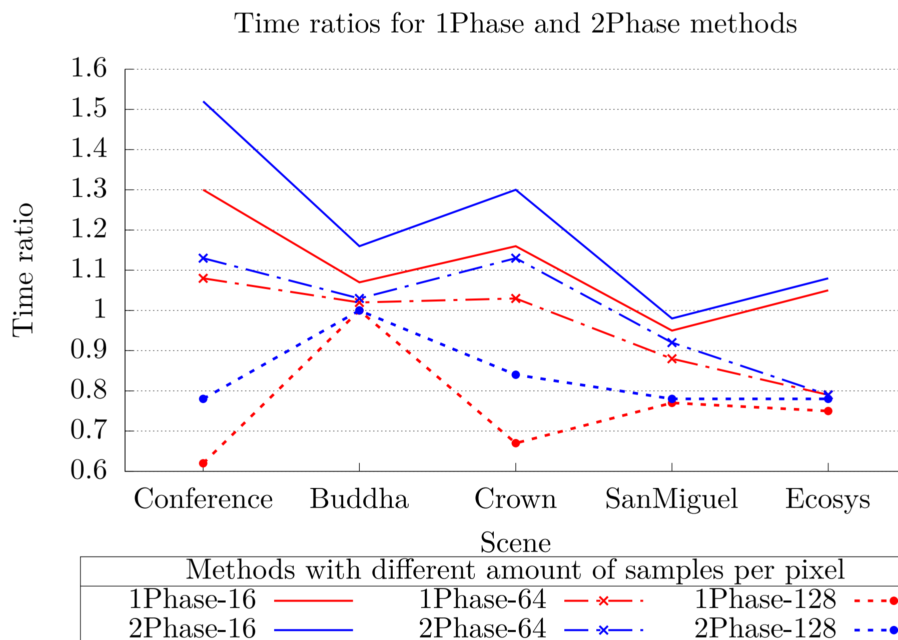
počtu paprsků (např. posláním jednoho na každý blok 16×16 pixelů), by pak opětovným sběrem u dvoufázové optimalizace mohlo oproti jednofázové dojít ke zlepšení.

Pro porovnání uvádíme pouze graf celkového času obou metod (viz obr. 3.7), jelikož počet testů s primitivy i obálkami pro stínové paprsky se i v nejvíce rozdílném měření liší o méně jak jedno procento.

3.5.3 Jedna a dvě nahlížecí tabulky

Abychom určili výhodu předpočítaného pořadí průchodu s ohledem na dělící osy z konstrukce uzlu v jediné tabulce oproti získávání pořadí z dvojice tabulek, provedli jsme porovnání obou variant na dvojici metod *SAO* a *1Phase* určujících pořadí průchodu synů se stínovými paprsky. Poměr celkového času výpočtu varianty s jednou tabulkou vůči variantě s dvěma tabulkami je vynesen do grafu 3.8.

Je patrné, že pro scény s větším počtem primitiv (na ose x seřazené od nejmenší po největší) začínají být oba způsoby srovnatelné, zatímco pro menší je výrazně výhodnější využít jednu nahlížecí tabulku. Pro větší počet primitiv a uzlů hierarchie se dá očekávat menší šance, že se data budou stále nacházet v cache paměti při jejich opětovném použití, například během vyzvednutí uzlu ze zásobníku traverzace. Celkově tak při použití jedné nahlížecí tabulky dochází k většímu počtu cache miss přístupů, ať už z důvodu menšího volného místa v důsledku její velikosti, tak i větší pravděpodobností nahrazení

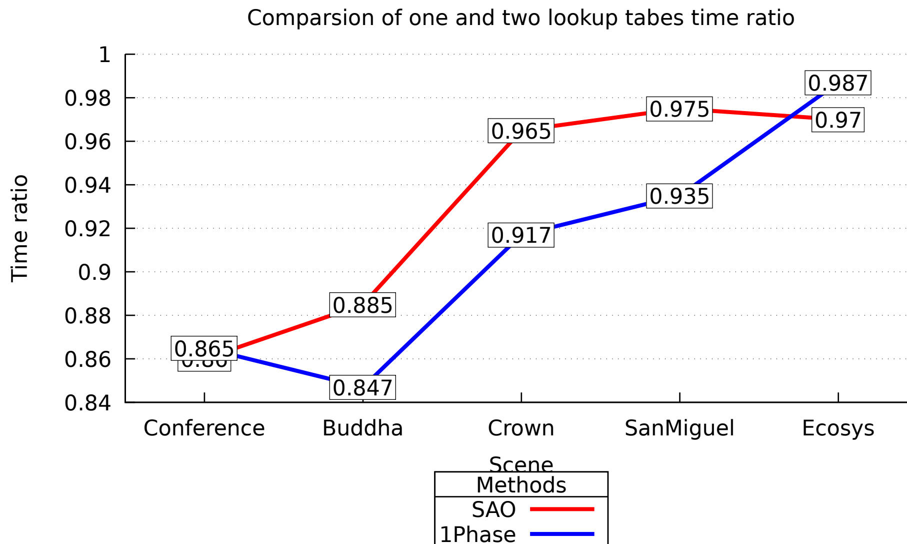


Obrázek 3.7: Poměr celkového času výpočtu metod *1Phase* a *2Phase* za použití různého počtu vzorků na pixel.

dat méně používané části tabulky a jejich opětovné získávání z pomalejšího úložiště při znovupoužití.

Varianta s jednou tabulkou nám dává pořadí průchodu rovnou z hodnoty indexu do ní, u varianty s dvěma tabulkami potřebujeme nejprve z první získat hodnotu určující průchod pomyslným stromem uzlu a s využitím této hodnoty teprve ve druhé nahlížecí tabulce určit hledané pořadí průchodu synů. Právě práce s pamětí a počet cache miss přístupů eliminují benefit získaný odstraněním několika výpočetních operací při použití jedné nahlížecí tabulky.

V případě větší arity hierarchie s mnohonásobným zvětšením nahlížecí tabulky, by počet cache miss přístupů ještě výrazně stoupl, varianta s dvojicí tabulek by se tak mohla stála časově efektivnější. Pro náš případ čtyř-ární BVH ale vychází lépe varianta s jednou tabulkou, pro zbytek měření využívajících metodu určování pořadí traverzace nahlížecí tabulku proto využíváme pouze jednu.



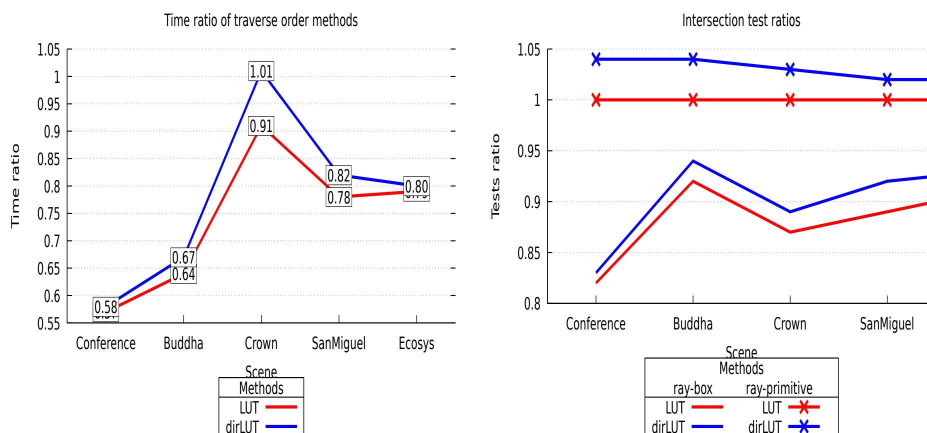
Obrázek 3.8: Poměr celkového času výpočtu metod *SAO* a *1Phase* s použitím jedné nahlížecí tabulky oproti použití dvojice nahlížecích tabulek k určení pořadí průchodu synů při dotazu o nejbližší průsečík s paprskem.

3.5.4 Určení pořadí ze směrů a z konstrukce

V další sadě měření jsme se zabývali rozdíly u metod využívající nahlížecí tabulky pro určení pořadí průchodu synů v průběhu hledání nejbližšího průsečíku paprsku s geometrickým primitivem (viz sekce 3.4).

Pro metodu využívající konstrukce nedojde nikdy ke zlepšení počtu testů průsečíku s geometrickými primitivy ve srovnání s původní binární hierarchii, ze které vycházíme. Příčinou je právě využití dělících os z konstrukce, podle kterých se u binární hierarchie určuje, zda se vydat nejprve do levého či pravého potomka. Právě uložením těchto dělících os a způsobu vzniku každého uzlu je tak simulován průchod částí původního stromu, který je teď reprezentovaný jedním uzlem. Důsledkem je dobře viditelný pokles testů paprsku s obalovým tělesem, způsobeným pouze přestavbou na n-ární hierarchii s menším počtem vnitřních uzlů.

Zdánlivá výhoda ale zůstává, že také nikdy nedojde ke zvýšení počtu testů s primitivy a poměr oproti původní binární hierarchii tak bude vždy roven právě jedné. Naproti tomu meto předpočítající pořadí vůči konečnému počtu směrů paprsků tuto vlastnost nemá a z důvodu malého počtu předpočítávaných směrů ke zlepšení nedochází. Z grafu 3.9 je patrné, že využívající konstrukce BVH je výhodnější časově i počtem testů průsečíku paprsku s obálkami i geometrickými primitivy. Ve výsledném porovnání proto dále používáme metodu s nahlížecí tabulkou na základě konstrukce.



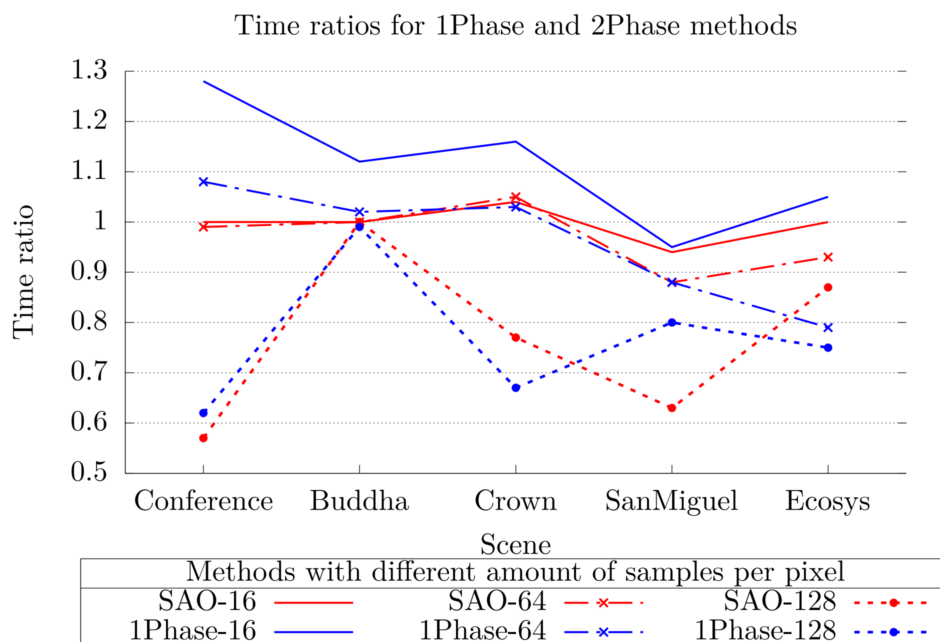
Obrázek 3.9: Porovnání metod určujících pořadí traverzace při hledání nejbližšího průsečíku za pomoci nahlížecí tabulky. Varianta *dirLUT* využívá předpočítaná pořadí pro 8 směrů paprsku, *LUT* představuje předpočítaná pořadí z konstrukce s jednou nahlížecí tabulkou. V levém grafu je poměr celkového času výpočtu, v pravém poměr počtu provedených testů průsečíku paprsku s obalovým tělesem a primitivem.

3.5.5 Srovnání metod podle plochy a pohledově závislých metod

Pro konečné měření používáme metodu *SAO* a *1Phase*, které se v průběhu práce a aplikací dříve popsaných změn ukázali jako nejlepší z kombinací optimalizace podle plochy a pohledově závislé optimalizace.

Výsledné hodnoty jsme získali z množiny alespoň čtyřech měření každé z metod, společně s různým počtem použitých vzorků na pixel. na rozdíl od zbytku práce jsme se zde omezili pouze na metodu syntézy *path* společně se stavbou výchozí BVH metodou *SAH*, která je běžně považována za tzv. zlatý standard ve své kategorii, zároveň se z části analýzy PBRT projevila i jako nejrychlejší s ohledem na celkovou dobu běhu programu. V grafu 3.10 jsou vyneseny celkové časy běhu programu za použití zmíněných metod v poměru s původní implementací z PBRT pro počet vzorků na pixel rovný 16, 64 a 128. Právě pro zvyšující se počet vzorků na pixel je patrné výrazné zlepšení u obou z metod způsobené menším poměrem času potřebného ke zpracování vstupních dat oproti celkové době výpočtu a naopak strávením větší části dotazováním do optimalizované BVH.

Poměr počtu testů u metody *SAO* zůstává pro všechny testované vzorky na pixel stejný, pro metodu *1Phase* mezi 16 a 128 vzorky na pixel klesne přibližně o jedno procento. Důvodem je menší vliv méně efektivní fáze sběru dat s větším počtem testů oproti zbylé části měření. Pro takto malou změnu jsme

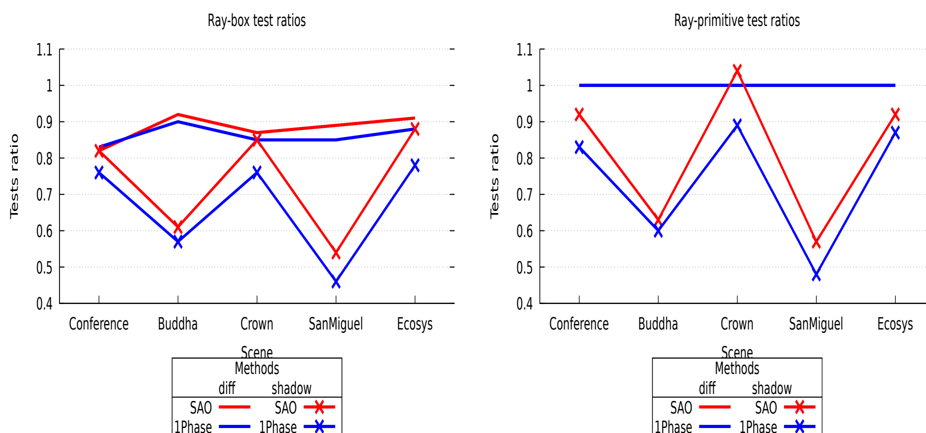


Obrázek 3.10: Porovnání celkového času výpočtu metod *SAO* a *1Phase* vůči neupravenému běhu programu s metodou syntézy obrazu *path* a stavbou BVH pomocí *SAH*. Naměřená data jsou porovnána pro různé počty vzorků na pixel s hodnotami z alespoň čtyř různých měření pro každou z použitých metod.

proto nepřidávali ke každému počtu vzorků samostatnou křivku do grafů 3.11, ale ponechali jsme pouze jednu reprezentativní s průměrnou hodnotou. Poměry časů strávených ve fázích výpočtu průsečíku s paprskem pro 128 vzorků na pixel jsou vypsány do tabulky 3.3 a vykresleny do grafů 3.12.

U obou výsledných metod se podařilo docílit redukce počtu testů průsečíku s obalovými tělesy v průměru o 13 % pro primární a odražené paprsky a o 30 % (26 % pro metodu *SAO* a 33 % pro *1Phase*) pro stínové paprsky. Zároveň se snížil i počet testů s primitivy v průběhu zpracování stínových paprsků v průměru o 19 % u metody *SAO* a 26 % u metody *1Phase*.

Celkové zrychlení obou metod je výrazně ovlivněno zvoleným počtem vzorků na pixel a pro malý počet může obzvláště pro metodu *1Phase* docházet i ke zpomalení v důsledku výpočetně náročnější fáze sběru dat. Až na scénu *Buddha* přesto dochází k výraznému celkovému zrychlení až o 53 % pro *SAO* a 57 % pro *1Phase* s průměrným zrychlením o 23 % u obou metod za použití 128 vzorků na pixel. Podrobnější výpis dat použitých v grafech je vypsán do tabulky 3.4.



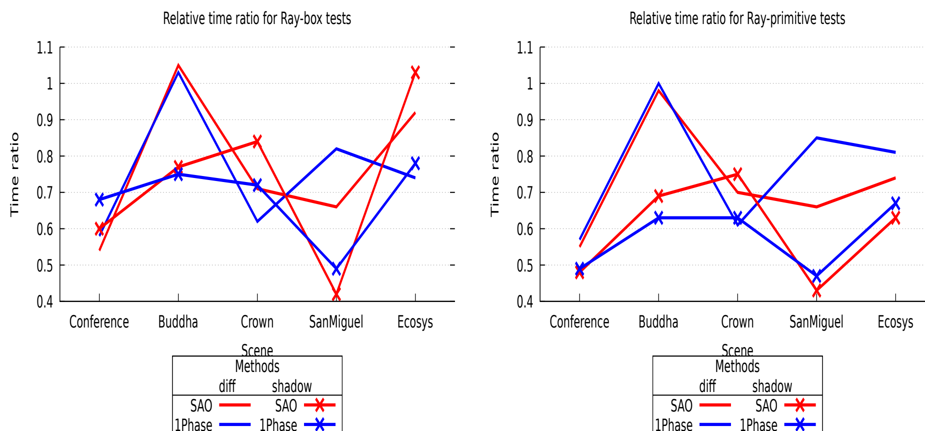
Obrázek 3.11: Porovnání počtu provedených testů průsečíku s paprskem pro metody *SAO* a *1Phase* za použití 128 vzorků na pixel, metody syntézy *path* a stavbou BVH metodou *SAH*. Poměry jsou uvedeny vůči původní implementaci PBRT.

Scene	Usedmethod	time total	#ray-box tests				#ray-primitive tests			
			diff		shadow		diff		shadow	
		[s]	[s]	[%]	[s]	[%]	[s]	[%]	[s]	[%]
Conference	Orig	263.5	78.6	29.83	33.3	12.64	33.9	12.86	6.3	2.39
	SAO	151.5	42.5	28.06	20.1	13.26	18.7	12.36	3.0	1.99
	1Phase	164.5	45.4	27.61	22.5	13.68	19.5	11.83	3.1	1.87
Buddha	Orig	80.5	21.8	27.07	5.8	7.24	6.2	7.64	1.0	1.27
	SAO	80.5	22.9	28.42	4.5	5.61	6.0	7.51	0.7	0.87
	1Phase	80	22.4	28.02	4.4	5.47	6.1	7.67	0.6	0.8
Crown	Orig	345.5	71.4	20.67	43.5	12.6	21.0	6.08	8.9	2.58
	SAO	264.5	50.9	19.24	36.7	13.86	14.7	5.55	6.7	2.53
	1Phase	230	44.3	19.26	31.2	13.57	12.8	5.56	5.6	2.44
San Miguel	Orig	712.6	350.1	49.13	56.6	7.94	88.1	12.36	10.1	1.42
	SAO	451.5	229.8	50.9	23.7	5.24	58.1	12.86	4.3	0.96
	1Phase	568.6	287.7	50.6	27.6	4.85	74.6	13.12	4.8	0.84
Ecosystem	Orig	312.5	167.9	53.72	35.2	11.25	27.4	8.78	4.5	1.45
	SAO	272.5	154.4	56.65	36.1	13.23	20.4	7.5	2.8	1.04
	1Phase	235	123.9	52.72	27.3	11.61	22.2	9.44	3.0	1.29

Tabulka 3.3: Časy strávené ve fázi hledání průsečíku během traverzace BVH a testování jednotlivých primitiv. Naměřená data jsou pro 128 vzorků na pixel.

3.6 Implementace a změny v PBRT

K rozšíření softwaru PBRT o výslednou implementaci jsme využili objektově orientovaného přístupu. Výsledkem je nová třída akcelerační struktury nazvaná *NBVHAccel* rozšiřující abstraktní třídu *Aggregate*, které dále vychází z abstraktní třídy všech primitiv (viz obr. 2.3). Volba použití této struktury



Obrázek 3.12: Porovnání relativního času stráveného ve fázích hledání průsečíku s paprskem pro metody *SAO* a *1Phase* za použití 128 vzorků na pixel, metody syntézy *path* a stavbou BVH metodou *SAH*. Poměry jsou uvedeny vůči původní implementaci PBRT.

je pak stejná, jako u původních variant *BVHAccel* a *KdTreeAccel*, tedy parametrem *Accelerator* ve vstupním souboru, zde s hodnotou *nbvh*.

Kromě přidání zdrojových kódů optimalizované čtyř-ární BVH, bylo třeba rozšířit stávající třídy primitiv o funkci optimalizace, aby bylo možné využít polymorfismu a provést pohledově závislou optimalizaci i pro instancované objekty, resp. nad nimi postavené BVH. Aby navíc příspěvky paprsků z fáze sběru dat byly použity i ve výsledném obrazu, došlo k drobné změně metod syntézy obrazu a příslušné abstraktní třídy *Integrator*. Změna však spočívá pouze v přeskočení počtu vzorků použitých pro sběr dat a možnost dosazení a předání kamery, která v sobě udržuje vykreslovaný obraz. Právě předáním kamery je tak možné využít příspěvky sledovaných paprsků z fáze sběru dat i ve výsledném výstupním obrazu.

Objektově orientovaná implementace PBRT a práce za pomoci abstraktních tříd se tak ve výsledku ukázala velice užitečnou, s možností snadného rozšíření a nutností pouze minimálních změn ve zbytku kódu.

Scene	Used method	time total [s]	SAH cost [-]	trace speed [Mrays/s]	#ray-box tests diff [M]	#ray-box tests shadow [M]	#ray-primitive tests diff [M]	#ray-primitive tests shadow [M]	Contracted nodes [k]	Contracted nodes [%]
16 samples per pixel										
Conference	Orig	23	128	12.13	6129	2917	513	169	0	0
	SAO	23	73	12.13	5010	2384	513	155	34	20.70
	1Phase	29.5	74	13.29	5095	2250	513	143	22	12.57
Buddha	Orig	10.5	16	12.64	2895	710	180	43	0	0
	SAO	10.5	9	13.24	2657	430	180	27	334	22.60
	1Phase	11.8	9	13.90	2609	414	180	26	234	14.80
Crown	Orig	28.5	34	15.02	5091	3447	443	336	0	0
	SAO	29.5	14	14.81	4425	2939	443	350	1102	23.77
	1Phase	33	14	15.89	4352	2656	443	301	507	9.70
San Miguel	Orig	62.5	54	9.39	22341	3556	1724	281	0	0
	SAO	58.5	32	9.56	19811	1920	1724	161	721	5.68
	1Phase	59.5	33	10.41	19145	1683	1723	137	308	2.22
Ecosystem	Orig	30	175	17.53	13469	3066	955	218	0	0
	SAO	30	110	17.53	12292	2711	955	200	248	1.14
	1Phase	31.5	113	18.57	11947	2409	954	191	155	0.68
64 samples per pixel										
Conference	Orig	79.5	128	14.08	24518	11669	2052	680	0	0
	SAO	78.5	73	14.25	20042	9537	2052	620	34	20.70
	1Phase	85.5	74	14.63	20174	8843	2052	565	22	12.57
Buddha	Orig	40.5	16	6.22	11584	2841	722	172	0	0
	SAO	40.5	9	13.75	10631	1723	722	108	334	22.60
	1Phase	41.3	9	14.10	10381	1626	722	104	234	14.79
Crown	Orig	109.5	34	15.66	20375	13791	1773	1347	0	0
	SAO	114.5	14	15.28	17708	11758	1773	1401	1102	23.77
	1Phase	113	14	15.79	17272	10493	1773	1202	507	9.69
San Miguel	Orig	257.5	54	9.08	89005	14164	6877	1121	0	0
	SAO	227.5	32	9.80	78927	7656	6877	643	721	5.68
	1Phase	226	33	10.07	75619	6516	6875	535	308	2.23
Ecosystem	Orig	148.5	175	14.18	53871	12263	3822	873	0	0
	SAO	137.5	110	15.32	49164	10842	3822	801	248	1.14
	1Phase	118	113	18.20	47487	9620	3816	768	155	0.68
128 samples per pixel										
Conference	Orig	263.5	128	8.50	49036	23339	4104	1360	0	0
	SAO	151.5	73	14.78	40084	19074	4104	1240	34	20.70
	1Phase	164.5	74	14.39	40280	17630	4104	1127	22	12.58
Buddha	Orig	80.5	16	13.84	23169	5682	1444	344	0	0
	SAO	80.5	9	13.84	21262	3448	1444	217	334	22.60
	1Phase	80	9	14.19	20737	3244	1444	208	233	14.77
Crown	Orig	345.5	34	9.93	40755	27585	3548	2695	0	0
	SAO	264.5	14	13.24	35421	23517	3548	2803	1102	23.77
	1Phase	230	14	15.05	34501	20904	3548	2408	508	9.72
San Miguel	Orig	712.6	54	6.55	177700	28273	13737	2240	0	0
	SAO	451.5	32	9.86	157582	15292	13737	1285	721	5.68
	1Phase	568.6	34	7.86	150742	12977	13731	1069	307	2.22
Ecosystem	Orig	312.5	175	13.47	107744	24523	7645	1745	0	0
	SAO	272.5	110	15.47	98330	21682	7645	1602	248	1.14
	1Phase	235	113	18.00	94889	19158	7633	1534	154	0.68

Tabulka 3.4: Detailní výsledky měření pro různé metody optimalizace společně s původní variantou programu. Počty testů průsečíku paprsku s obálkami i primitivy jsou uvedeny v milionech, množství uzlů podléhajících kontrakci v tisících s poměrem vůči celkovému počtu uzlů BVH.

Závěr

Tato práce pojednává o metodách optimalizace akceleračních struktur BVH a jejich uplatnění v počítačové praxi. Zkoumáním postupu softwaru PBRT se nám podařilo odhalit části syntézy obrazu s vysokou časovou náročností a navrhnout a implementovat dvojici metod kombinující různá optimalizační řešení včetně nového postupu průchodu synů uzlu BVH při dotazování na nejbližší průsečík s paprskem. Obě z těchto metod se navíc výrazně liší požadovanými vstupními daty. Zatímco první metoda využívá pouze plochu známou z konstruované BVH, druhá je pohledově závislá a vyžaduje odhad distribuce paprsků ve scéně.

Implementované metody, i s možnými variantami, jsme otestovali na množině pěti scén a výsledky porovnali s původní variantou rozšiřovaného softwaru. Za použití většího množství vzorků na pixel se podařilo docílit na první pohled výrazného zrychlení v časech celkového běhu programu jak u pohledově závislé optimalizace, tak u metody optimalizace podle plochy. V obou případech s průměrným zrychlením pro 128 vzorků na pixel o 23 % z celkového času syntézy obrazu.

Redukce času stráveného v jednotlivých fázích výpočtu průsečíku je patrná hlavně pro stínové paprsky s průměrným zrychlením testů s obalovými tělesy o 32% pro metodu pohledově závislé optimalizace *1Phase* a 27% pro metodu optimalizace podle plochy *SAO* a testy s primitivy u obou metod rychlejšími v průměru téměř o 42%. Obdobně pak hledání nejbližšího průsečíku s průměrným zrychlením o 24% pro metodu *1Phase* a 22% pro metodu *SAO* pro testy s obalovými tělesy a 23% a 27% pro testy s primitivy.

Obě metody se tak ukázaly jako výhodné řešení optimalizace BVH pro zpracování stínových paprsků i pro redukci celkového času běhu programu. Metoda *1Phase* je výrazně závislá na počtu použitých vzorků na pixel, avšak i za použití dostatečného množství vzorků jsou výsledky srovnatelné, nebo dokonce horší oproti metodě *SAO* (pro 512 vzorků na pixel v průměru o 2.3%). Optimalizace podle plochy navíc disponuje podstatnou implementační výhodou možného použití okamžitě po dokončení stavby výchozí binární hierarchie.

Literatura

[Áfr13] Attila T. Áfra. Faster incoherent ray traversal using 8-wide AVX instructions. Technical report, Babeş-Bolyai University, Cluj-Napoca, Romania, August 2013.

[BH09] Jiří Bittner and Vlastimil Havran. Rdh: Ray distribution heuristics for construction of spatial data structures. In *Proceedings of the 25th Spring Conference on Computer Graphics*, SCCG '09, pages 51–58, New York, NY, USA, 2009. ACM.

[FFD09] Bartosz Fabianowski, Colin Fowler, and John Dingliana. A Cost Metric for Scene-Interior Ray Origins. In P. Alliez and M. Magnor, editors, *Eurographics 2009 - Short Papers*. The Eurographics Association, 2009.

[FLF12] Nicolas Feltman, Minjae Lee, and Kayvon Fatahalian. SRDH: Specializing BVH Construction and Traversal Order Using Representative Shadow Ray Sets. In Carsten Dachsbacher, Jacob Munkberg, and Jacopo Pantaleoni, editors, *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2012.

[GHB15] Yan Gu, Yong He, and Guy E. Blelloch. Ray specialized contraction on bounding volume hierarchies. *Comput. Graph. Forum*, 34(7):309–318, October 2015.

[GPM11] Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. Simpler and faster hlbvh with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 59–64, New York, NY, USA, 2011. ACM.

[GS87] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl.*, 7(5):14–20, May 1987.

[HHS06] Vlastimil Havran, Robert Herzog, and Hans-Peter Seidel. On the fast construction of spatial hierarchies for ray tracing. pages 71–80, Sept 2006.

[HM08] W. Hunt and W. R. Mark. Ray-specialized acceleration structures for ray tracing. In *2008 IEEE Symposium on Interactive Ray Tracing*, pages 3–10, Aug 2008.

[JMV13] Bittner Jiří, Hapala Michal, and Havran Vlastimil. Fast insertion-based optimization of bounding volume hierarchies. *Computer Graphics Forum*, 32(1):85–100, 2013.

[KK86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. *SIGGRAPH Comput. Graph.*, 20(4):269–278, August 1986.

[NM14] Jae-Ho Nah and Dinesh Manocha. SATO: Surface Area Traversal Order for Shadow Ray Tracing. *Computer Graphics Forum*, 33(6):167–177, September 2014.

[ODJ16] Shinji Ogaki and Alexandre Derouet-Jourdan. An n-ary bvh child node sorting technique for occlusion tests. *Journal of Computer Graphics Techniques (JCGT)*, 5(2):22–37, June 2016.

[O'N14] Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering, Third Edition: From Theory To Implementation.* 3rd. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016.

[WBB08] Ingo Wald, Carsten Benthin, and Solomon Boulos. Getting rid of packets - efficient simd single-ray traversal using multi-branching bvhs -. In *2008 IEEE Symposium on Interactive Ray Tracing*, pages 49–57, Aug 2008.

[WBKP08] Bruce Walter, Kavita Bala, Milind Kulkarni, and Keshav Pingali. Fast agglomerative clustering for rendering. pages 81 – 86, 09 2008.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980.

Příloha A

Seznam použitých zkratek

PBRT Physically Based rendering Toolkit

BVH Hierarchie obalových těles

SAH Surface Area Heuristic

AABB Obalové těleso osově zarovnaného kvádru

BDPT Dvousměrové sledování cest

B Bajt paměti

b Bit paměti

BRDF Dvousměrová odrazová distribuční funkce

BSDF Dvousměrová rozptylová distribuční funkce

Příloha B

Obsah přiloženého CD

```
readme.txt.............................stručný popis obsahu CD
pbrt-v3....................zdrojové kódy PBRT s implementací
  src
    accelerators...............zdrojové kódy optimalizace BVH
scenes..............................připravené ukázkové scény
text.............................................text práce
  thesis.pdf......................text práce ve formátu PDF
  thesis.....................zdrojová forma práce ve formátu LaTeX
Build.sh..........................skript pro kompilaci programu
RunTests.sh.......skript pro spuštění programu na ukázkových scénách
```