VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HYBRIDNÍ RAYTRACING V ROZHRANÍ DXR

HYBRID RAYTRACING IN DXR

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ POLÁŠEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JOZEF KOBRTEK**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Polášek Tomáš, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Hybridní raytracing v rozhraní DXR**
Hybrid Raytracing in DXR
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s vykreslovací technikou ray-tracing a grafickým API DirectX12.
2. Otestujte a prozkoumejte možnosti real-time ray-tracing s pomocí DirectX12 a DirectX Ray Tracing (DXR).
3. Navrhněte hybridní renderer využívající DXR, který umožňuje interaktivní ray-tracing scény v kombinaci s rasterizací. Navrhněte množinu testovaných efektů.
4. Renderer naimplementujte.
5. Diskutujte dosažené výsledky a vhodnost implementace DXR do aktuálních herních enginů. Navrhněte, které efekty by bylo výhodné takto implementovat.

Literatura:

- dle doporučení vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kobrtek Jozef, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Cílem této práce je zhodnotit míru použitelnosti hardwarové akcelerace sledování paprsků na grafických procesorech, v současných herních a zobrazovacích enginech. K dosažení tohoto cíle je použit systém *DirectX Ray Tracing* a grafické akcelerátory *Nvidia Turing*. Práce obsahuje návrh a implementaci hybridního enginu s podporou akcelerace sledování paprsku, nad kterým jsou implementovány často používané grafické efekty – tvrdé a měkké stíny, odrazy a ambientní okluze. K hodnocení je přistoupeno z hlediska náročnosti integrace do existujícího enginu, výkonnosti výsledného systému a výhodnosti implementace zvolených grafických efektů. Součástí práce je integrace *DXR* do existujícího enginu používaného v reálných podmínkách a testování výkonnosti zahrnující parametry paprsků vyslaných za sekundu, času stavby akceleračních struktur a doby výpočtu sledování paprsku na GPU.

Abstract

The goal of this thesis is to evaluate the usability of hardware accelerated ray tracing in contemporary rendering engines. Specifically, *DirectX Ray Tracing API* and *Nvidia Turing* architecture are being examined. Design and implementation of a hybrid rendering engine with support for hardware accelerated ray tracing is included and used in implementation of frequently used graphical effects – hard and soft shadows, reflections, and Ambient Occlusion. The assessment is made in terms of difficulty of integration into a rendering engine, performance of the resulting system and suitability of implementation of chosen graphical effects. Performance parameters – including number of rays cast per second, time to build acceleration structures and computation time on the GPU – are tested and discussed.

Klíčová slova

Ray Tracing, Hybridní Ray Tracing, DXR, DirectX, DirectX Ray Tracing, Rasterizace, Nvidia Turing, Akcelerovaný Ray Tracing, Ambient Occlusion

Keywords

Ray Tracing, Hybrid Ray Tracing, DXR, DirectX, DirectX Ray Tracing, Rasterization, Nvidia Turing, Accelerated Ray Tracing, Ambient Occlusion

Citace

POLÁŠEK, Tomáš. *Hybridní raytracing v rozhraní DXR*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jozef Kobrtek

Hybridní raytracing v rozhraní DXR

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jozefa Kobrtka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Polášek
18. května 2019

Poděkování

Rád bych poděkoval vedoucímu této diplomové práce Ing. Jozefu Kobrtkovi za jeho odborné vedení a pomoc při řešení problémů. Děkuji také Martinu Sobkovi a Petru Smílkovi za jejich pomoc při integraci sledování paprsku do enginu ve firmě Hangar13.

Obsah

Kapitola 1

Úvod

Již od počátků počítačové grafiky existovaly snahy o co nejrealističtější zobrazování virtuálních scén aby vypadaly tak jako ve skutečnosti, kde platí fyzikální zákony šíření světla. Aktuálně používané techniky využívající rasterizaci se dopouštějí mnoha zjednodušení, které i přes jejich náročnou implementaci produkují nerealistické výstupní obrazy. Naproti tomu techniky využívající sledování paprsku jsou implementačně jednodušší a generované výsledky mohou působit věrohodněji. Jejich nevýhodou je ale vysoká výpočetní náročnost a nutnost komplexní znalosti celé scény. S příchodem architektury grafických akcelerátorů *Nvidia Turing* a obsažených *Ray Tracing* jader přichází možnost urychlování vykreslovacích algoritmů využívajících sledování paprsku, jejichž výstupy mohou být již nerozeznatelné od reality. Ovšem, potom vyvstává otázka: „Je tato technologie již použitelná v dnešních zobrazovacích enginech?“. Je možné její využití v real-time grafických systémech, jakými jsou například počítačové hry? Cílem této práce je odpovědět právě na výše položené otázky.

Kapitola 2 je zaměřena na některé skutečnosti a teoretické základy počítačové grafiky a realistického zobrazování. Kromě vymezení základních pojmů, které jsou v práci dále používány, jsou zde také popsány grafické efekty na kterých bude testována použitelnost celého systému. Následuje Kapitola 3, popisující použité aplikační programovací rozhraní *DirectX*, se zaměřením na *Direct3D* a *DirectX Ray Tracing*.

Práce pokračuje Kapitolou 4, která obsahuje návrh hybridního vykreslovacího enginu, dále použitého při testování. Prioritou tohoto enginu je umožnit přístup ke grafickému akcelerátoru na co nejnižší úrovni a zároveň automatizovat některé často používané akce. Za tímto účelem je použito výše zmíněné API *Direct3D* ve verzi 12, které zároveň umožňuje akceleraci sledování paprsku za použití rozhraní *DirectX Ray Tracing*. Celkový návrh enginu je zde předveden ve formě blokového diagramu a funkce jeho částí jsou dále rozvedeny.

Kapitola 5 obsahuje některé zajímavé části z implementace navrženého hybridního enginu pojmenovaného *Quark*. Hlavní část této kapitoly je zaměřena na práci s *DirectX Ray Tracing*, včetně začlenění této technologie do vykreslovacího řetězce. Dále je zde také popsána implementace jednotlivých grafických efektů – využívající sledování paprsku i rasterizaci – a samotného hybridního vykreslování.

Kapitola 6 se zabývá vyhodnocením výsledků získaných za použití automatického i manuálního testování implementovaného systému a vyvození závěrů z naměřených dat. Kromě popisu testovacích konfigurací tato kapitola obsahuje výsledky testů měření počtu paprsků vyslaných za sekundu, srovnání implementovaných grafických efektů s rasterizačními alternativami a měření parametrů stavby akceleračních struktur. Dále jsou zde také popsány zkušenosti s integrací technologie akcelerovaného sledování paprsku do reálného herního enginu.

Kapitola 2

Principy hybridního vykreslování

Počítačová grafika, a realistické zobrazování vůbec, je velmi rozsáhlý obor, který od svého vzniku provázelo mnoho různých teoretických modelů a metod. Cílem této kapitoly je shrnout některé zásadní skutečnosti a definovat termíny, které budou nadále v práci používány. Obsah není vyčerpávající popis všech možností vykreslování, ale jde pouze o informace zaměřené na hybridní vykreslování.

Kapitola začíná specifikací zobrazovací rovnice a popisem jejich jednotlivých částí. Dále jsou představeny dvě dnes často používané techniky vykreslování virtuálních scén – *rasterizace* a *sledování paprsku*. Po rozboru těchto technik a způsobu jakým jsou používány k řešení zobrazovací rovnice následuje popis jejich propojení v *hybridní* vykreslovací systém. Poslední částí této kapitoly je popis zvolených grafických efektů, pro pozdější testování.

2.1 Zobrazovací rovnice a její řešení

Zobrazovací rovnice [9, 10] je integrální rovnicí, která popisuje ustálený stav scény z pohledu energie fotonů vyzářených zdroji světla – jejím řešením je ustálené rozložení „světla“ ve scéně. Exaktním řešením pro danou scénu – která je popsána geometrií a vlastnostmi materiálů – a následným vykreslením vypočtených hodnot je potom možné získat fyzikálně přesný syntetický obraz.

2.1.1 Popis zobrazovací rovnice

Rovnice může být zapsána mnoha rovnocennými způsoby, mezi které patří integrace přes směry [10], nebo integrace přes plochy [9]. Tato práce bude nadále pracovat se zjednodušeným zápisem směrové integrace, který nebere v potaz změny s ohledem na vlnovou délku světla a čas, jejíž zápis lze formulovat následovně [8]:

$$L(\boldsymbol{x}, \omega_\mathrm{o}) = L_e(\boldsymbol{x}, \omega_\mathrm{o}) + \int_\Omega f_\mathrm{r}(\boldsymbol{x}, \omega_\mathrm{i} \rightarrow \omega_\mathrm{o})\, L(r(\boldsymbol{x}, \omega_\mathrm{i}), -\omega_\mathrm{i})\, cos\,\theta_\mathrm{i}\, d\omega_\mathrm{i} \tag{2.1}$$

Kde jednotlivé použité symboly mají následující význam:

- $\boldsymbol{x}$ značí pozici na některém z povrchů geometrie scény.
- ω_o a ω_i jsou směrové vektory, které postupně značí výstupní a vstupní směr zkoumaného světla.

- $L(\boldsymbol{x}, \omega_\mathrm{o})$ reprezentuje výslednou radianci bodu na povrchu $\boldsymbol{x}$, na který je nahlíženo z daného směru ω_o. Z pohledu radiometrie jde o zář (*radiance*) vyslanou ve směru vektoru ω_o a jde tedy o intenzitu záření $[\,\frac{W}{m^2}\,]$ na prostorový úhel sr – $[\,\frac{W}{m^2 sr}\,]$.

- $L_e(\boldsymbol{x}, \omega_\mathrm{o})$ je zář, emitovaná samotným povrchem v bodě $\boldsymbol{x}$ ve směru ω_o.

- $\int_\Omega d\omega_\mathrm{i}$ reprezentuje integraci přes všechny směry v hemisféře nad aktuálním bodem, ze kterých je sbírána vstupní zář. Každý směr je reprezentován dvěma úhly ve sférické souřadnicové soustavě a proto je tento integrál dvou-rozměrný.

- $f_\mathrm{r}(\boldsymbol{x}, \omega_\mathrm{i} \rightarrow \omega_\mathrm{o})$ je distribuční funkcí odrazu (*BRDF*, nebo „Bidirectional Reflectance Distribution Function“), která popisuje materiálové vlastnosti v aktuálním bodě povrchu $\boldsymbol{x}$. Výsledná hodnota specifikuje poměr světla odraženého – ve směru ω_o – ke světlu dopadajícímu ze směru ω_i.

- $L(r(\boldsymbol{x}, \omega_\mathrm{i}), -\omega_\mathrm{i})$ odpovídá záři dopadajícího světla ze směru ω_i. Pomocná funkce $r(\boldsymbol{x}, \omega_\mathrm{i})$ získá bod scény, který je první protnut paprskem vyslaného z bodu $\boldsymbol{x}$ ve směru ω_i.

- $cos\,\theta_\mathrm{i}$ je posledním prvkem integrovaného výrazu, jehož cílem je utlumit světlo podle úhlu dopadu od normály. Hodnotu je možné také získat pomocí skalárního součinu normály povrchu[1] a zkoumaného směru dopadu: $n \cdot -\omega_\mathrm{i}$.

2.1.2 Řešení zobrazovací rovnice

Jak již bylo výše zmíněno, řešením zobrazovací rovnice je ustálený stav scény, kdy jsou energetické hodnoty v jednotlivých bodech v rovnováze, který je dán zobrazovací rovnicí. Tento výsledek je z pohledu počítačové grafiky velmi užitečný, protože je s jeho pomocí možné syntetizovat fyzikálně věrný obraz scény. Finální obraz je získán vzorkováním vypočítané funkce $L(\boldsymbol{x}, \omega_\mathrm{o})$ podle modelu kamery, kterou je do scény nahlíženo.

Zobrazovací rovnici nelze řešit analyticky[2], k čemuž přispívá několik faktorů kterým je věnován zbytek této pod-sekce. Hlavním problémem je obsah integrálu v zobrazovací rovnici 2.1, který nelze pro složité scény analyticky řešit. Příčinou je přítomnost záře L na obou stranách rovnice, kdy pro výpočet záře v jednom bodě jsou nutné hodnoty záře všech viditelných bodu v hemisféře Ω [8].

Existuje mnoho algoritmů, které se snaží – alespoň v omezené míře – zobrazovací rovnici řešit. V zásadě je nutné vyřešit dva výše zmíněné problémy: integrace přes všechny směry hemisféry Ω a rekurzivní formulaci. Jednotlivé metody se liší v tom jak přesně počítají zobrazovací rovnici a které vlastnosti dokáží řešit. Čím je řešení komplexnější, tím vyšší má metoda výpočetní náročnost.

Nejpoužívanější třídou algoritmů pro aplikace zobrazující v reálném čase jsou algoritmy *Lokálního Osvětlení* [14]. Tyto algoritmy počítají osvětlení pro jednotlivé body, kdy jsou jednotlivé prvky zobrazovací rovnice degradovány na složku lokální – například „difuzní“ a „spekulární“ – a složku „ambientní“ – které nahrazuje integraci přes ostatní viditelné objekty a rekurzivní vyhodnocení jejich záře. Díky hrubé nepřesnosti nejsou tyto metody často považovány za řešení zobrazovací rovnice.

[1] Normála je v případě použitého zápisu součástí specifikace funkce *BRDF* pro daný materiál a proto je v rovnici pouze θ_i jako úhel mezi normálou a směrem dopadu.

[2] Kromě jednoduchých případů – například pro *BRDF* funkce, které vždy vrací 0.

Již komplexnějším řešení zobrazovací rovnice poskytují algoritmy založené na dělení scény na konečný počet prvků nebo plošek, nazývány také metody typu *Radiozita*. Pro tyto metody je použita výše zmíněná forma zobrazovací rovnice, která integruje přes všechny plochy [9]. Jednotlivé body scény jsou agregovány do větších plošek, kterých je výsledně konečně mnoho. Díky konečnému množství plošek lze integrál transformovat na sumu přes všechny plošky, čímž se již stává analyticky spočítatelným. Rovnice stále obsahuje rekurzivní použití hodnoty záře na obou stranách rovnice, která je ale již řešitelná iterativně. V zásadě je možné sestavit rozsáhlou soustavu rovnic a iterativně hledat její kořeny až do doby kdy dosáhne konvergence a tedy i ustáleného stavu. Omezením těchto metod je jejich zjednodušení *BRDF*, které musí být nezávislé na ω_o a ω_i, čímž je znemožněna věrohodná reprezentace lesklých povrchů[3].

Poslední třídou algoritmů jsou algoritmy *Sledování Paprsku* (*Ray Tracing*). Jejich základní myšlenkou je rekurzivní vrhání paprsků do scény, čímž je řešena viditelnost dvou libovolných bodů scény. Existují různé implementace, od základního zobrazování bez osvětlení, až po statisticky úplné řešení zobrazovací rovnice za použití metody *Monte Carlo*. Tato práce se hlouběji zabývá třídou algoritmů sledování paprsku v části 2.2 a proto zde nejsou dále rozváděny.

2.2 Použité vykreslovací techniky

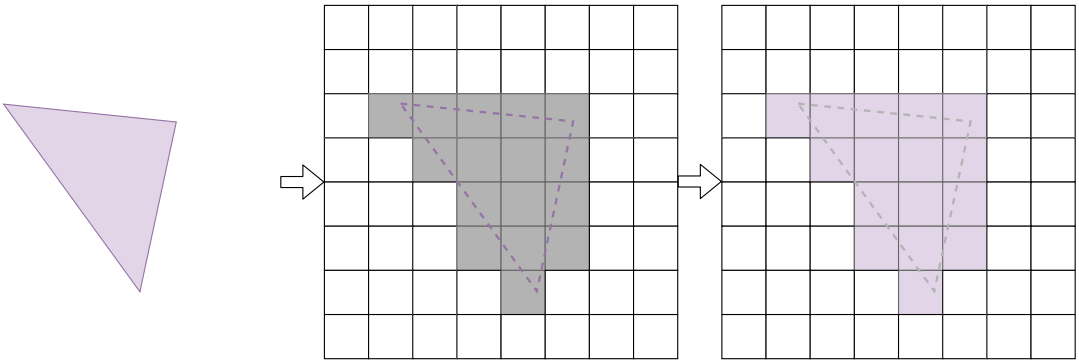
Vyřešením zobrazovací rovnice jsou získána veškerá data nutná pro syntetizaci výsledného obrazu, k čemuž je použita některá z vyreslovacích technik. Vykreslovací techniky lze také považovat za určité programovací rámce („framework“), které samy o sobě dokáží zobrazovací rovnici řešit. Pro hybridní zobrazovací engine navržený v této práci budou použity vykreslovací techniky *rasterizace* a *sledování paprsku*, jejichž vlastnosti a výhody použití jsou popsány v následujících sekcích. Informace v následujících pod-sekcích jsou převzaty z knihy *Computer Graphics* [6].

2.2.1 Rasterizace

Vykreslovací technika rasterizace [6, Kapitola 15] se skládá z několika hlavních částí – algoritmus rasterizace, z-buffer a obarvování pixelů. Cílem algoritmu rasterizace je transformace vektorového popisu grafických primitiv do obrazového rastru. Výstupem pro každé primitivum je tedy množina *fragmentů* a ke každému z nich přiřazených atributů – hloubka, interpolační parametry a další. Dalším použitým algoritmem je z-buffer, který řeší viditelnost jednotlivých fragmentů, založených na jejich hloubce. Poslední z hlavních částí je obarvování pixelů, které viditelným fragmentům přiřazuje výslednou barvu, založenou na jejich atributech. Vykreslovací technika rasterizace pracuje po primitivech, kterými jsou ve většině systémů trojúhelníky. Příklad vykreslení jednoho primitiva lze vidět na obr. 2.1.

Rasterizace je velmi široce používána v aplikacích, které vyžadují vykreslování v reálném čase, primárně díky její rychlosti a hardwarové podpoře ve formě grafických akcelerátorů. Této rychlosti lze dosáhnout díky vysokému paralelizmu, kdy každý trojúhelník lze zpracovávat odděleně, až do doby kdy je výsledek zapisován do výstupního rastru za použití z-bufferu.

[3] *BRDF* lesklých materiálů obsahuje tzv. „Specular Lobe“, který výrazně zvyšuje odrazivost v případech kdy úhel odrazu je podobný úhlu dopadu.



Obrázek 2.1: Ukázka jednotlivých fází vykreslovací techniky rasterizace. Vstupem je jedno primitivum (**vlevo**), které je nejdříve rasterizačním algoritmem převedeno do rastru obrazu (**uprostřed**) a následně jsou výsledné fragmenty obarveny (**vpravo**).

Z pohledu řešení zobrazovací rovnice je vykreslovací technika rasterizace, ve své čisté podobě, schopna řešit pouze velmi základní problémy. Díky z-bufferu je automaticky řešena viditelnost a tedy po vykreslení všech primitiv ve scéně je výsledný syntetizovaný obraz vzdáleně podobný reálné scéně. Pro vylepšení výsledků lze použít výše zmíněné části obarvování pixelů a provést výpočet *Lokálního Osvětlení*.

Díky vysoké rychlosti je tato technika vhodná pro první vykreslení scény, které lze použít pro zjištění viditelnosti a základní obarvení pixelů. Následně je možné nasadit další metody, které mohou s pomocí vygenerovaných dat pokračovat v realističtějším vykreslení scény („Deferred Shading“ blíže v části 2.4).

2.2.2 Sledování paprsku

Vykreslovací techniku *sledování paprsku* („Ray Tracing“) lze považovat za celou množinu postupů, které využívají *vysílání paprsků* („Ray Casting“) k řešení vykreslovacích úloh [23, 2]. Vstupem těchto technik je specifikace kompletní scény a případně výstupního rastru. Jedna operace vyslání paprsku obnáší následující akce:

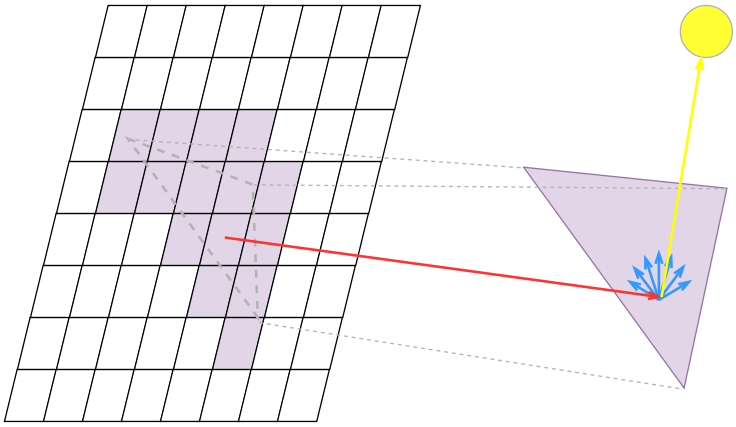
1. Vytvoření paprsku z parametrů: počáteční bod, směr a časový interval ve kterém bude scéna prohledávána.

2. Prohledávání scény ve směru paprsku a hledání nejbližšího průniku.

3. Vrácení času nejbližšího průniku, nebo nekonečna (maxima) pokud k průniku nedošlo.

Jednotlivé algoritmy sledování paprsku se liší podle způsobu použití paprsků a jejich vysílaného množství. Základní typy paprsků – *primární*, *sekundární* a *stínové* – lze vidět na obr. 2.2. Originální algoritmus sledování paprsku [23] vykresluje kompletní scénu vysláním jednoho – *primárního* – paprsku skrz každý pixel rastru. Následně záleží na vlastnostech zasaženého povrchu, zda je difuzní nebo lesklý. Pro lesklé povrchy je vyslán jeden sekundární paprsek podle zákona odrazu, který rekurzivně pokračuje. Pokud paprsek zasáhne povrch s difuzním materiálem, jsou vyslány stínové paprsky k jednotlivým zdrojům světla a paprsek je „obarven“ odpovídajícím způsobem. Výsledkem je mnohem kvalitnější řešení zobrazovací rovnice, než v případě rasterizace, které kromě tvrdých stínů zobrazuje i zrcadlové odrazy.

Velké množství dnes používaných metod sledování paprsků používá metody typu *Monte Carlo* pro vyhodnocení složitějších efektů [10]. Metoda integrace *Monte Carlo* umožňuje

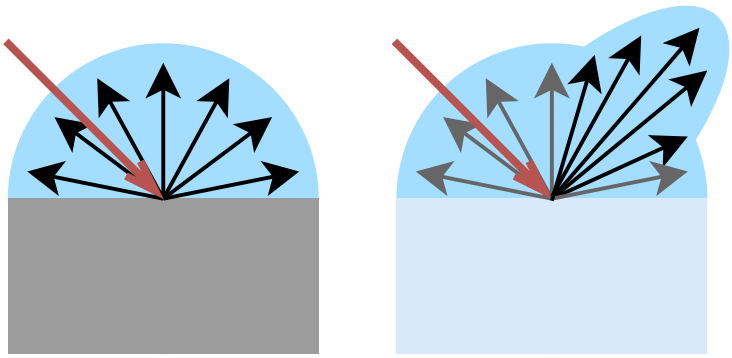


Obrázek 2.2: Příklad jednoduchého sledování paprsku, použitého pro vykreslení scény. Diagram obsahuje rastr, do kterého jsou zapisovány výsledné „barvy“ paprsků (**vlevo**) a vykreslované primitivum (**vpravo**). Kromě primárních paprsků (**červeně**), jsou zde také naznačeny paprsky sekundární (**modře**) a stínové (**žlutě**).

numerické řešení integrálů za použití náhodného vzorkování integrované funkce. V případě zobrazovací rovnice jde o integrál přes všechny směry v hemisféře nad bodem ($\int_\Omega d\omega_\mathrm{i}$), ze kterých je sbírána výsledná hodnota. Náhodné vzorkování lze tedy popsat jako výběr náhodných směrů, ve kterých jsou vrhány sekundární paprsky. Základní varianta Monte Carlo integrace, s adekvátně kvalitním zdrojem náhody, exaktně řeší daný integrál s limitou počtu iterací jdoucí k nekonečnu [10].

Mezi komplexní řešení zobrazovací rovnice patří např. algoritmy *Distributed Ray Tracing* [5] nebo *Path Tracing* [10]. Tyto algoritmy používají metodu *Monte Carlo* pro vyhodnocení integrálu přes hemisféru nad zkoumaným bodem. Oproti základnímu sledování paprsků se liší tím, že generují větší množství *sekundárních paprsků* po průniku se scénou, čímž dosahují pokročilých efektů, jakými jsou například měkké stíny nebo rozmazání pohybem. Tyto sekundární paprsky jsou vrhány pouze do předem dané úrovně zanoření, kdy dojde k postupnému návratu „barev“ paprsků (záře nebo jiných hodnot). Další změnou je vysílání vyššího množství *primárních* paprsků pro každý pixel, které jsou časově nebo prostorově posunuty a jejich následná agregace do výsledného pixelu. Díky vysílání několika posloupností paprsků – několika *cest* – je možné zmírnit šum a jiné artefakty získané skrz Monte Carlo integraci.

Nevýhodou algoritmů sledování paprsku, které využívají slepou Monte Carlo integraci, je jejich extrémní časová náročnost. Tato slepá, nebo „naivní“, Monte Carlo integrace velmi pomalu konverguje k výsledné hodnotě, a proto je nutné sbírat vysoké množství náhodných vzorků. Informované metody Monte Carlo integrace, jakou je například *Importance Sampling* [10], používají informace o předem známém rozložení dominantních prvků integrované funkce. V zobrazovací rovnici jsou to prvky: materiálová distribuční funkce odrazu (f_r), hodnota záře protnutého bodu (L) a tlumící faktor ($cos\,\theta_\mathrm{i}$). Tyto dominantní prvky jsou potom použity ke vzorkování oblastí, které mají velký přínos pro výslednou hodnotu, v kontrastu s uniformním vzorkováním u naivní Monte Carlo integrace. Příkladem může být použití lesklosti povrchu definované v *BRDF* a zvýšení vzorkování v oblasti lesklých odrazů – příklad lze vidět na obr. 2.3. *Importance Sampling* lze také použít pro výběr drobných změn již existujících paprskových cest, a tím například zlepšit konvergenci algoritmu pro scény s malým množstvím světla [21].



Obrázek 2.3: Rozdíl mezi vzorkováním difuzního materiálu (**vlevo**), jehož *BRDF* udává stejnou pravděpodobnost pro všechny směry odrazu a lesklý materiál (**vpravo**), jehož *BRDF* obsahuje tzv. „spekulární lalok“ („Specular Lobe“). Při použití *Importance Sampling* by mohla být většina vzorků lesklého materiálu obsažena právě v oblasti největšího odrazu.

2.3 Hardwarová akcelerace sledování paprsku

Vzhledem k časové náročnosti vykreslovacích technik sledování paprsku („Ray Tracing“), především vysokého opakování operace vyslání paprsku („Ray Casting“), je výhodné tyto techniky urychlovat pomocí specializovaných hardwarových akcelerátorů [7]. Ve srovnání s akcelerací rasterizačních vykreslovacích technik, nelze sledování paprsku urychlovat podobně přímočaře. Rasterizaci lze relativně jednoduše akcelerovat díky datové nezávislosti jednotlivých primitiv. Další výhodou rasterizace je dobrá lokalita dat, které je možné dosáhnout díky předvídatelnosti přístupů do paměti pro jedno vykreslované primitivum.

U vykreslovacích technik sledování paprsku nelze jednoduše předvídat se kterou částí scény bude paprsek kolidovat, čímž vzniká potřeba *náhodného* přístupu do kompletní reprezentace scény. Z relativně náhodného přístupu vyplývá také nemožnost optimalizovat lokalitu přístupu k datům scény – paprsek může za sebou zasáhnout prostorově velmi vzdálené části scény.

2.3.1 Akcelerace vysílání paprsku

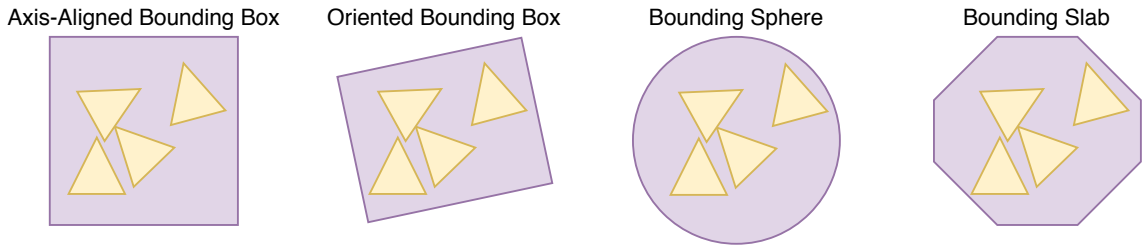
Prvním krokem urychlování *sledování paprsku* je akcelerace algoritmu *vysílání paprsku*, který je základem celého procesu. Abstrahovaný zápis algoritmu vysílání paprsku lze vidět na Algoritmu 2.1. Výpočetní složitost algoritmu vzniká díky smyčce přes všechny objekty ve scéně (řádek 3 algoritmu 2.1), jejichž počet může dosahovat velmi vysokých hodnot. Za objekty scény se v tomto případě považují jednotlivá primitiva – ve většině moderních systémů jde tedy o trojúhelníky.

```
1 Function rayCast(ray, scene):
      Result: Closest intersected primitive
2     closest := none;
3     forall objects in scene do
4         if intersects(ray, object) and distance(object) < distance(closest) then
5             closest := object;
6         end
7     end
8     return closest;
9 return
```

Algoritmus 2.1: Algoritmus vysílání paprsku do scény, jehož cílem je najít nejbližší primitivum ve scéně, vzhledem k cestě zadaného paprsku.

Druhým místem algoritmu, které je často urychlováno lze najít na řádku 4 algoritmu 2.1. Zde je prováděn výpočet a detekce průniku paprsku a primitiva, který je v základní verzi algoritmu provedena pro každou dvojici paprsek-primitivum. Existuje mnoho optimalizovaných algoritmů pro výpočet průniku paprsku s různými typy primitiv, včetně hardwarových implementací, a proto bych se dále zaměřil pouze na optimalizaci *počtu* kontrolovaných primitiv.

Tuto redukci množiny kontrolovaných objektů lze chápat ze dvou směrů. Prvním z nich je agregace množiny menších primitiv pod primitiva větší, což vede na tvorbu tzv. *obalových těles* [11]. Cílem Obalového tělesa je vyřadit velké množství operací průniku vůči základních primitivům (většinou trojúhelníky) a nahradit je jednou kontrolou průniku vůči danému tělesu. Existuje mnoho různých typů obalových těles, přičemž často používané varianty lze vidět na obr. 2.4. Mezi důležité parametry každého obalového tělesa patří kvalita obalení libovolné množiny prostorově lokalizovaných primitiv – tzn. jak těsně obalové těleso obaluje. Další důležitou vlastností, zejména z pohledu optimalizace, je rychlost – nebo náročnost – algoritmu, který počítá průniky daného obalového tělesa a paprsku.



Obrázek 2.4: Ukázky často používaných obalových těles pro pevně danou množinu primitiv – trojúhelníků.

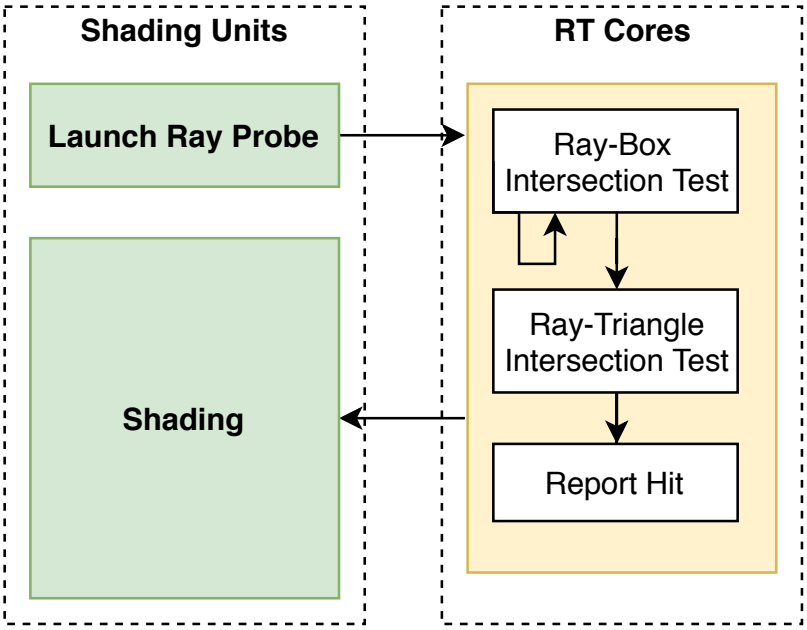
Druhým směrem optimalizace počtu kontrolovaných objektů je volba pouze těch, které mají šanci být v cestě daného paprsku. K úkolu výběru prostorově blízkých objektů se používají akcelerační datové struktury, které se v zásadě dělí do dvou kategorií – *mřížky* a *stromy* [7, 11]. Za mřížku je označena taková datová struktura, která pokrývá celý zadaný prostor uniformní mřížkou, kde jednotlivá pole jsou všechna stejně velká. Mřížky jsou výhodné v případě, kdy prostor obsahuje rovnoměrně rozprostřenou množinu primitiv a proto jsou vhodné pro pokrytí jednoho lokalizovaného modelu. Reprezentativní implementací může být například struktura *Grid* [7].

Častěji používané jsou stromové struktury, které dělí prostor nerovnoměrně a dokáží tedy pokrýt i rozptýlené scény. Jedním typem stromové akcelerační struktury je *Bounding Volume Hierarchy* [11], který využívá výše zmíněných obalových těles. Listy tohoto stromu obsahují množiny primitiv, které jsou obaleny první vrstvou obalových těles. Stavba stromu následně pokračuje agregací několika prostorově lokalizovaných těles nižší úrovně pod jedno obalové těleso vyšší úrovně. Výsledkem je strom, jehož kořen obsahuje obalové těleso celé scény. Mezi další stromové struktury patří například *kD stromy* [15], jejichž výhodou je jednodušší průchod[4] a nepřekrývání jednotlivých pod-stromů.

2.3.2 Hardwarová akcelerace

Existuje mnoho implementací hardwarové akcelerace sledování paprsku, pomocí výše zmíněných technik. V jednotlivých řešeních lze téměř vždy nalézt rozdělení hardware na urychlení průchodu akcelerační strukturou („Traversal Units“) a na výpočty průniků („Intersection Units“) [15]. Jednotky pro výpočet průniků se navíc často dělí na výpočet průniku s obalovým tělesem a výpočet průniku s grafickým primitivem.

Kromě specializovaných hardwarových prostředků lze také použít již existující grafické akcelerátory a implementovat průchod akcelerační strukturou a výpočty průniků pomocí *Compute Shaderů*. Této možnosti je často využíváno v systémech, které grafický akcelerátor již používají pro rasterizaci – například počítačové hry – a je tedy jednodušší použít stejné zařízení, než vyžadovat několik rozšiřujících karet. Nevýhodou tohoto přístupu je potenciálně nižší výkon a vysoké vytížení prostředků grafického akcelerátoru, jehož výkon je navíc používán pro stínování a další operace.



Obrázek 2.5: Diagram použití *Ray Tracing* jader u architektury *Nvidia Turing* [12]. Jádra jsou rozdělena do dvou typů, první z nich provádí výpočet průniků s obalovými tělesy a průchod akcelerační struktury, druhá potom průniky se samotnými primitivy.

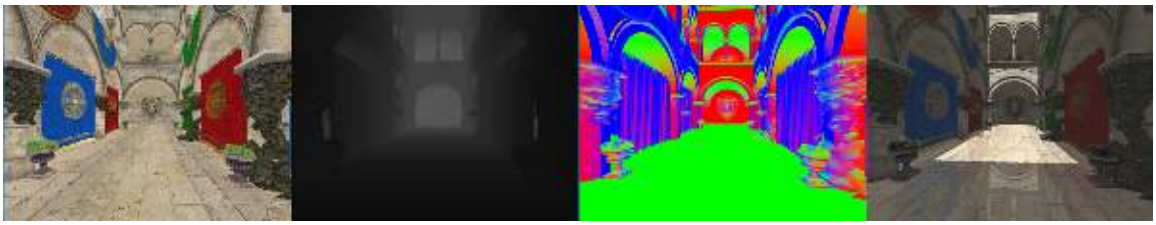
[4]Každá úroveň stromu je rozdělena pomocí jedné n-dimenzionální roviny, pro kterou existuje jednoduché řešení průniku s paprskem.

Tato práce pro akceleraci sledování paprsku využívá grafických akcelerátorů od firmy *Nvidia*, jejichž karty s architekturou *Turing* obsahují dedikovaný hardware pro akceleraci sledování paprsků [16, 12]. Tento systém používá dva typy jednotek: *Box Intersection Evaluators* a *Triangle Intersection Evaluators*. První z nich obsahuje také logiku procházení akcelerační datové struktury, což lze vidět na diagramu na obr. 2.5. Tento sytém používá blíže nespecifikovaný typ obalových těles založený na *Bounding Boxes*, jako akcelerační strukturu potom výše zmíněnou *Bounding Volume Hierarchy* [12]. Druhý typ jednotek provádí výpočet průniku s primitivy, kterými jsou v tomto případě trojúhelníky.

2.4 Hybridní vykreslování

Pod pojmem *Hybridního vykreslování* je v této práci myšleno propojení dvou výše popsaných vykreslovacích technik – *rasterizace* a *sledování paprsku*. Cílem této techniky je snížit počet vysílaných paprsků pro každý vykreslovaný snímek, pomocí před-vykreslení cílové scény za pomocí rasterizace, díky čemuž je možné přeskočit vysílání primárních paprsků. Výsledné propojení rasterizace a sledování paprsku se snaží o vybalancování výhod a nevýhod obou těchto přístupů a jejich odpovídajících hardwarových realizací [6, Kapitola 15].

Hlavní výhodou rasterizace je její vysoká rychlost, která umožňuje zpracovat velké množství trojúhelníku ve velmi krátkém čase. Této propustnosti je dosaženo masivní *paralelizací* a *zřetězením* na úrovni jednotlivých vertexů, primitiv a fragmentů. Každá složitější operace ve formě shaderů tento proces zpomaluje a proto je vhodné proces vykreslování rozdělit do několika fází, čehož využívají algoritmy založené na *odloženém vykreslování* (*Deferred Rendering*) [17]. Jejich základní myšlenkou je zpracovat všechnu potenciálně viditelnou geometrii jednoduchou sadou shaderů, jejichž výstup je shromážděn do tzv. *Geometry Bufferu* (*G-Buffer*). Příklad geometry bufferu s informací o albedo, hloubce a normále lze vidět na obr. 2.6.



Obrázek 2.6: Ukázka odloženého vykreslování (*Deferred Rendering*). V tomto případě je generován *G-buffer* se třemi informačními vrstvami – albedo barva (**vlevo**), hloubka (**uprostřed**) a normála (**vpravo**). Výsledný obraz (**vpravo**) je složen z jednotlivých vrstev.

Po vykreslení celé scény do *G-Bufferu* jsou jednotlivé vrstvy použity pro syntetizaci výsledného obrazu. Následující algoritmy již mohou pracovat pouze s viditelnou geometrií, o které jsou v jednotlivých vrstvách uloženy informace. Tímto je omezen výpočet náročných efektů pouze na fragmenty ve výstupním obraze, které již nebudou překryty jinými objekty ve scéně.

2.5 Grafické efekty

Při výběru efektů, na kterých bude prováděno testování systému hardwarové akcelerace sledování paprsku, jsem se přiklonil primárně k takovým efektům, které jsou řešeny v rasterizačních enginech za použití triků a algoritmů napodobujících vyžadovaný výsledek. Tato sekce obsahuje popis vybraných grafických efektů a jejich realizaci za použití rasterizace a sledování paprsku.

2.5.1 Tvrdé stíny

Rozpoznávací vlastností tvrdých stínů je jejich binární hodnota, kdy každá ploška může být pouze osvětlena nebo neosvětlena. Tvrdé stíny vznikají pro nekonečně malé světelné zdroje – bodové zdroje, spotlight atp. – nebo pro nekonečně vzdálené zdroje, což je případ směrových světel. Tento typ stínů je v počítačové grafice často používán díky relativně jednoduché implementaci [6, Kapitola 15]. Velkou nevýhodou je jeho nerealističnost, protože většina světel v reálném světě čistě tvrdý stín nevrhá. Příklad tvrdého stínu kulovitého objektu lze vidět na obr. 2.7.

Obrázek 2.7: Postupná změna tvrdého stínu na měkký, se zvětšujícím se zdrojem světla – od bodového (**vlevo**) po největší plošný (**vpravo**).

Tento efekt jsem vybral zejména kvůli jeho relativně nízké výpočetní náročnosti a možnostem jednoduchého srovnání s alternativami založenými na rasterizaci. Základní postup využití sledování paprsku pro výpočet tvrdých stínů (Algoritmus 2.2) začíná vykreslením G-bufferu za použití rasterizace. Následně je pro každý fragment tohoto bufferu získána pozice ve scéně, která je dále použita pro generování paprsků pro každé bodové světlo. Tyto paprsky jsou nastaveny tak, že jejich počátek leží v daném bodě scény a směřují do středu bodového světla. Následně je provedena jediná operace vyslání paprsku, která vrací nejbližší objekt, který byl po cestě paprsku nalezen. Pokud je tento objekt dále, než vybrané světlo, nebo neexistuje, potom lze říci, že vybrané světlo tento bod osvětluje.

```
1  Function rayTraceHardShadows(gbuffer, scene, lights):
       Result: Radiance value for each fragment
2      forall fragment in gbuffer do
3          position := positionForFragment(fragment);
4          radiance := 0;
5          forall light in lights do
6              ray := rayFromTo(position, light);
7              closest := rayCast(ray, scene);
8              if distance(light) ≤ distance(closest) then
9                  radiance := radiance + lightRadiance(light);
10             end
11         end
12         saveRadiance(gbuffer, radiance);
13     end
14 return
```

Algoritmus 2.2: Algoritmus vysílání paprsku do scény, jehož cílem je najít nejbližší primitivum ve scéně, vzhledem k cestě zadaného paprsku.

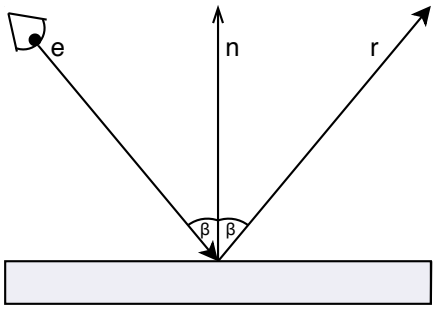
Velmi podobnou operací lze zobrazovat i spotlights, u kterých je navíc nutné kontrolovat zda paprsek je obsažen ve světelném kuželu. Operace lze také upravit pro směrová světla tak, že místo vyslání paprsku do středu světla, je paprsek vyslán proti směru vybraného směrového světla. Výsledek funkce vyslání paprsku je následně kontrolován tak, že pokud nebyl nalezen žádný objekt, potom směrové světlo daný bod osvětluje.

Alternativou pro generování tvrdých stínů za použití sledování paprsku je rasterizační metoda *Shadow Mapping* [6, Kapitola 15]. Hlavním principem je použití hloubkových map k určení viditelnosti daného bodu scény z pohledu světelného zdroje. Prvním krokem je pořízení hloubkové mapy za použití kamery umístěné do testovaného světla. Tato mapa je vytvořena pro každé testované světlo. Při vykreslování finálního syntetizovaného obrazu je potom provedena projekce testovaného bodu do prostoru stínové mapy světla a pokud je hloubka v této mapě nižší než ta vypočtená, lze říci, že je tento bod zastíněn.

2.5.2 Odrazy

Jedním z dlouhodobých problémů v počítačových hrách a dalších aplikacích používajících techniku rasterizace je vykreslování lesklých materiálů. Přesné odrazy a zrcadlení je sice možné vykreslovat pomocí triků, ale jde pouze o dobře specifikované povrchy, kterými jsou například plochá zrcadla. Příkladem takové techniky je umístění speciální kamery s vhodnou projekční maticí za vykreslovaný povrch [20]. Sledování paprsku umožňuje vykreslování přesných odrazů na geometrii s lesklými materiály a díky vrhání paprsků lze tento efekt rozšířit i na povrchy u kterých nelze jinak přesné odrazy vykreslit. Z těchto důvodů bylo zrcadlení přidáno do množiny testovaných efektů.

Odraz je z pohledu technik sledování paprsku principiálně velmi jednoduchý, díky možnostem vyslání nového paprsku po protnutí povrchu. Pokud je v průběhu sledování paprsku zasažen povrch, jehož materiál je dostatečně odrazivý (zrcadlový, „specular“), algoritmus vyšle paprsek nový, jehož počáteční bod je bodem průniku a směr lze vypočítat pomocí normály povrchu a úhlu dopadu prvního paprsku. Příklad tohoto odrazu lze vidět na obr. 2.8.



Obrázek 2.8: Při vykreslování odrazů je paprsek vysílán v opačném směru, než tomu je ve fyzickém světě. Úhel odrazu je roven úhlu dopadu, který je vypočten pomocí normály daného povrchu.

Výsledky této metody budou srovnány jak s výše zmíněnými triky, které lze použít pouze pro omezenou geometrii. Dalším srovnatelým efektem je *Screen-Space Reflections* [19, 20]. Odrazy v prostoru obrazu jsou zajímavým řešením, které kombinuje rasterizaci a v omezené míře sledování paprsku. Toto sledování paprsku je provedeno nad vykresleným *G-Bufferem*, kdy paprsky procházejí hloubkové mapy. Nevýhodou této metody je nemožnost zobrazit odrazy čehokoliv, co není aktuálně v G-Bufferu – a tedy ve finálním obraze.

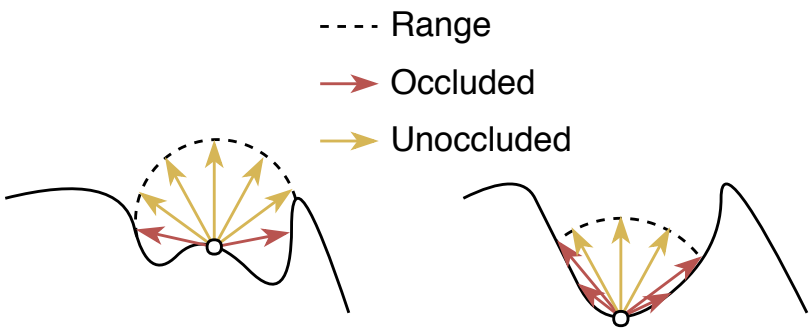
2.5.3 Ambient Occlusion

Ambient Occlusion je světelný efekt, který se snaží vizuálně napodobit snížené osvětlení bodů scény, které jsou obklopeny jinou geometrií [13]. Syntetizovaný obraz s touto přidanou hodnotou se ve výsledku jeví pozorovateli mnohem plastičtější, protože tyto drobné stíny zvýrazňují drobné deformace geometrie. V podstatě jde o vrhání stínů na velmi malém měřítku, kdy normální metody tvrdých stínů nelze použít. Příklad efektu lze vidět na obr. 2.9.

Obrázek 2.9: Ukázka vizuálního efektu *Ambient Occlusion*. Model je zobrazen bez efektu (**vlevo**) a s efektem (**vpravo**).

Tento efekt je automaticky přítomný pro komplexní řešení zobrazovací rovnice, díky integraci hemisféry záře nad zobrazovaným bodem. Kvůli náročnosti tohoto řešení se místo kompletní integrace provádí vyslání pouze několika paprsků, které jsou navíc omezeny v jejich délce [13]. Pokud tyto paprsky dosáhnou specifikované délky aniž by zasáhly jinou geometrii, je paprsek považován za „nezastíněný“. Pokud je zasažena geometrie, jde o „zastíněný“ paprsek. Vizualizaci těchto paprsků lze vidět na obr. 2.10. Výsledkem této metody

je tzv. „úroveň zastínění“, kterou lze vypočítat jako poměr *zastíněných* paprsků vůči celkovému počtu paprsků. Tento efekt byl zvolen právě z důvodu jeho implementace pomocí omezeného sledování paprsku.

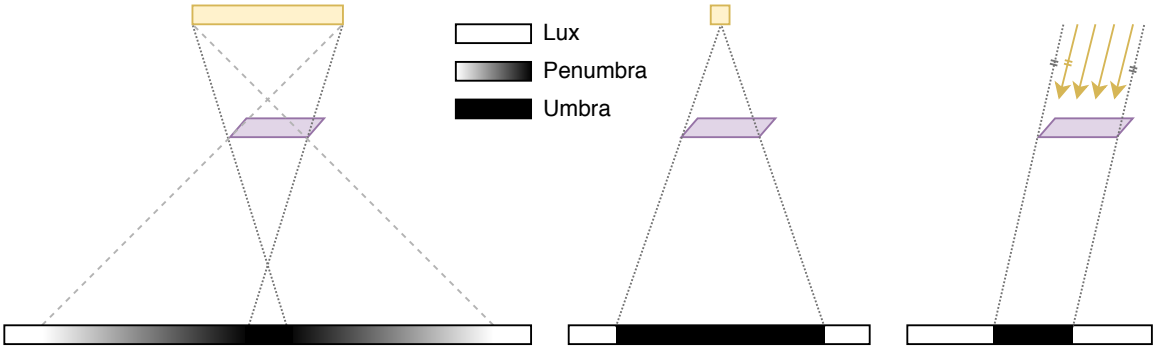


Obrázek 2.10: *Ambient Occlusion* napodobuje efekt sníženého osvětlení v mezerách a rozích. Málo zastíněný bod (**vlevo**) bude ve výsledku světlejší, než více zastíněný bod (**vpravo**).

Výsledný efekt bude porovnáván s rasterizačními metodami *Screen-Space Ambient Occlusion* [13] a její upravenou verzí *Horizon Based Ambiend Occlusion* [3]. Obě tyto metody opět pracují s hloubkovými mapami, ve kterých provádějí vysílání paprsků. Samotný výpočet probíhá podobně, jako již výše popsaný přístup využívající sledování paprsku.

2.5.4 Měkké stíny

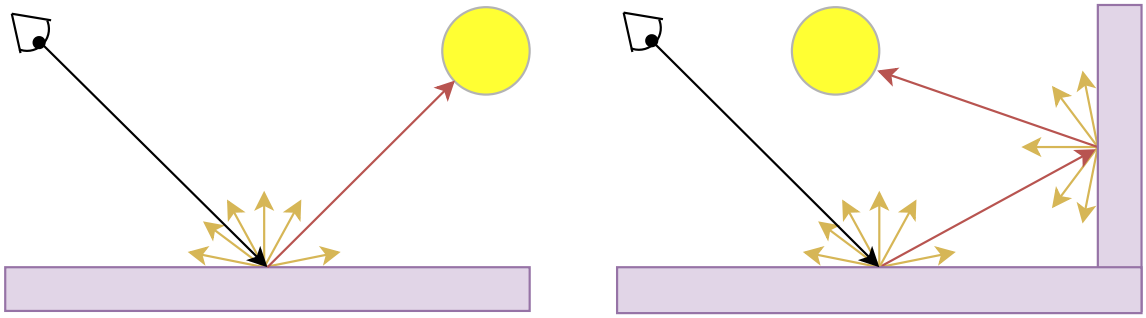
Posledním z testovaných efektů jsou realistické měkké stíny, které vznikají při použití plošných zdrojů světla. Oproti tvrdým stínům nemají pro jeden zdroj světla pouze binární hodnotu, ale na jejich okrajích vzniká postupný přechod, který se v optice nazývá „penumbra“. Příklad vzniku těchto stínů lze vidět na obr. 2.11. Rasterizační zobrazovače většinou měkké stíny pouze simulují, za pomocí rozmazání okrajů vygenerovaných tvrdých stínů [4].



Obrázek 2.11: Ukázka typů stínů – měkké stíny vznikají pro plošné zdroje (**vlevo**) a jejich vzhled působí mnohem přirozeněji díky přítomnosti postupného přechodu (tzv. „penumbra“). Tvrdé stíny vznikají například pro bodové světelné zdroje (**uprostřed**) a směrová světla (**vpravo**).

Měkké stíny lze za pomoci sledování paprsku řešit podobně jako tvrdé stíny s tím, že je pro každý plošný zdroj nutné vrhat paprsků několik, aby bylo možné určit míru zakrytí daného zdroje. Jelikož jde v princip o integraci přes plochu daného plošného zdroje

světla, je zde vhodné použít výše zmíněnou metodu *Monte Carlo integrace* [10]. Podporou plošných zdrojů světla se tento efekt blíží ke globálnímu osvětlení, které lze označit za komplexní řešení zobrazovací rovnice. Každá ploška scény se po osvětlení stává sekundárním zářičem a tedy novým plošným zdrojem světla. Rozšířením *Monte Carlo* vzorkování na plnou hemisféru – namísto pouze té části, ve které je plošný zdroj světla, protože všechny body jsou nyní potenciální světelné zdroje – se algoritmus dostává na úroveň metody *Path Tracing*. Rozdíl mezí přímým a nepřímým osvětlením lze vidět na obr. 2.12.



Obrázek 2.12: Ukázka přímého osvětlení (**vlevo**) a první úrovně nepřímého osvětlení (**vpravo**).

Tento efekt jsem zvolil jak kvůli jeho implementační a výpočetní náročnosti, tak i vizuálnímu výsledku, který je potenciálně jeho výstupem. Výsledky budou srovnávány primárně s algoritmy pro falešné měkké stíny [4]. Tyto metody pracují s již popsanými tvrdými stíny, které rozmazávají takovým způsobem, aby výsledek působil realisticky.

Kapitola 3

Rozhraní DirectX 12

Tato kapitola se zabývá základními rysy a principy rozhraní *DirectX 12*, na kterém jsem se rozhodl postavit výsledný zobrazovací engine. Toto rozhodnutí bylo založeno primárně na možnostech hardwarově akcelerovaného sledování paprsků, které jsou přístupné v tomto rozhraní – v počátcích této diplomové práce neexistovaly alternativní implementace kromě *DirectX Ray Tracing*.

Po počátečním úvodu do základních principů knihovny *DirectX* následuje rozbor rozhraní *Direct3D 12*, jako základní vykreslovací části. Zbytek kapitoly je potom věnován modulu pro akcelerované sledování paprsku, který je součástí *DirectX Ray Tracing* (*DXR*) rozhraní. Po popisu základů tohoto rozhraní následuje podrobný popis architektury *DXR*, spolu s mapováním na hardwarové možnosti grafických akcelerátorů *Nvidia Turing*. Závěr kapitoly je potom věnován emulační vrstvě, která umožňuje používat *DXR* i na grafických akcelerátorech bez hardwarové podpory sledování paprsku.

3.1 Knihovna DirectX

Knihovna *DirectX* je množinou aplikačních programovacích rozhraní, které umožňují tvorbu multimediálních aplikací pro operační systém *Microsoft Windows* a herní konzole *XBOX*. Mezi její nejdůležitější části, z pohledu této práce, patří *Direct3D* a *DXGI* a *DirectX Ray Tracing*, které budou hlouběji popsány v následujících sekcích. Zajímavou komponentou je také *DirectCompute*, umožňující výpočet obecných úloh na grafickém adaptéru (*General Purpose GPU*, *GPGPU*), které je ale v případě *Direct3D 12* zpřístupněno skrz výpočetní pipeline. Knihovna je postavena na tzv. *Component Object Model* (*COM*).

3.2 Grafické rozhraní Direct3D 12

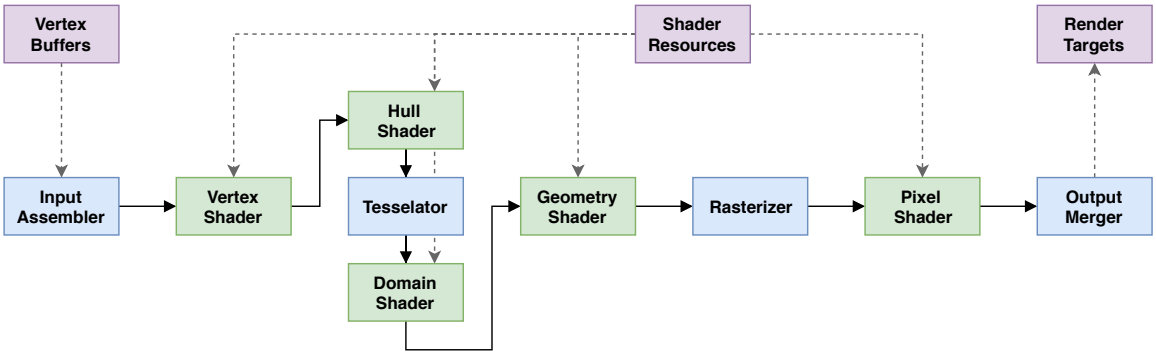
Direct3D[1] je aplikační programovací rozhraní, které umožňuje vykreslování 3D grafiky za použití předem zvolených zařízení. Samo o sobě *Direct3D* (*D3D*) neumožňuje výběr cílového *adaptéru*, na kterém se budou vykreslovací výpočty provádět. Z tohoto důvodu je součástí *DirectX* také rozhraní *DirectX Graphics Infrastructure* (*DXGI*), jehož cílem je také enumerace dostupných zařízení. Společně jsou tyto dvě rozhraní – *D3D* a *DXGI* – často používány jako základ vykreslovacích a herních enginů.

Tato práce využívá *Direct3D* ve verzi 12, která se liší od předchozích verzí ve složitosti vrstvy abstrakce mezi cílovým zařízením a aplikací, která ho používá. Cílem *Direct3D 12*

[1] Informace přejaty z docs.microsoft.com .

je umožnit vývojáři velmi nízkoúrovňový přístup k výpočetním prostředkům grafických akcelerátorů a tím zlepšit možnosti optimalizace. Další výhodou nízkoúrovňového přístupu je snížení režie (*overhead*) vrstvy ovladačů a *Direct3D* za běhu aplikace[2].

Hlavní částí *Direct3D 12* je rasterizační pipeline, která reprezentuje tok dat nutných pro syntetizaci výstupního obrazu. Zjednodušený diagram obsahující všechny fáze lze vidět na obr. 3.1. V *Direct3D 12* je nutné pipeline objekt plně konfigurovat před jeho použitím a po sestavení ho již není možné měnit. Pipeline obsahuje několik konfigurovatelných částí – *Input Assembler*, *Tesselator*, *Rasterizer* a *Output Merger* – které lze pouze nastavit pomocí množiny předem daných parametrů. Důvodem omezených možností je jejich přímá implementace ve výpočetním hardware, která umožňuje vyšší výkonnost. Důležitou částí rasterizační pipeline jsou její programovatelné části, jejichž funkcionalitu lze specifikovat pomocí programů (*shaderů*) napsaných v programovacím jazyce *High-Level Shading Language* (*HLSL*).



Obrázek 3.1: Diagram rasterizačního pipeline v *Direct3D 12* Některé vykreslovací fáze jsou pouze konfigurovatelné (**modré**), jiné jsou plně programovatelné skrze shadery (**zelené**). Externí paměťové zdroje jsou vyznačeny **fialově**.

3.3 DirectX Ray Tracing

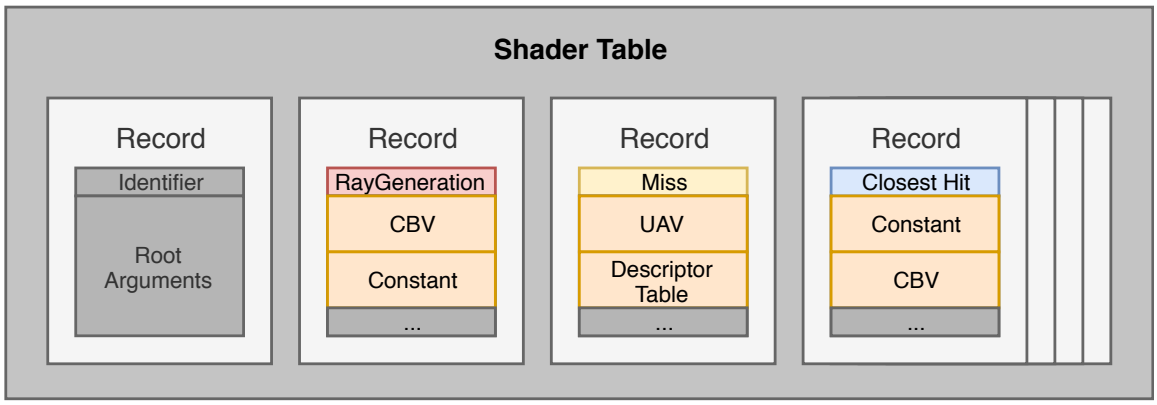
DirectX Ray Tracing (*DXR*) je aplikační programovací rozhraní, které zpřístupňuje funkcionalitu hardwarově akcelerovaného sledování paprsku. *DXR* bylo první použitelnou implementací, která tuto funkci zpřístupňovala pro grafické akcelerátory *Nvidia Turing*. V době psaní této práce je již k dispozici také alternativní implementace využívající grafické API *Vulkan*, zpřístupněna v rozšíření pod názvem „VK_NVX_raytracing“ [3]. Architektura obou těchto API pro sledování paprsku je velmi podobná, díky stejným přístupům obou vykreslovacích rozhraní a využití stejného hardware.

Následující pod-sekce se zabývají jednotlivými novými strukturami, které *DXR* přidává nad rámec *Direct3D 12*. Informace v těchto sekcích byly získány z oficiální specifikace rozhraní [1]. Na závěr je celý systém také shrnut a je ukázáno jak jsou jednotlivé části propojeny do většího celku, který umožňuje sledování paprsku.

[2] Viz https://developer.nvidia.com/dx12-dos-and-donts .
[3] Viz https://devblogs.nvidia.com/vulkan-raytracing/ .

3.3.1 Tabulka shaderů

Prvním novým konceptem v *DXR* jsou tabulky shaderů. Každou tabulku lze abstraktně chápat jako množinu shader programů a jejich parametrů. Diagram zobrazující možnosti konfigurace tabulky lze vidět na obr. 3.2. Tabulka shaderů obsahuje záznamy, přičemž každý záznam obsahuje informaci o přiřazeném shader programu a seznam parametrů předaným tomuto shaderu při jeho invokaci. Tento přístup je rozdílný oproti rasterizační pipeline, kde pro každý možný shader je právě jeden „slot“, který má pouze dvě možnosti – shader program je nastaven, nebo není. V případě pipeline pro sledování paprsku toto není možné, protože některé typy shaderů jsou specifické pro objekty scény. Obecně při sledování paprsku není jisté, který objekt daný paprsek zasáhne a proto je nutné aby všechny shadery byly přístupné a specifický shader je vybrán až za běhu – podle zasaženého objektu. Proto může jeden pipeline objekt pro sledování paprsku obsahovat obecně mnoho tabulek shaderů.

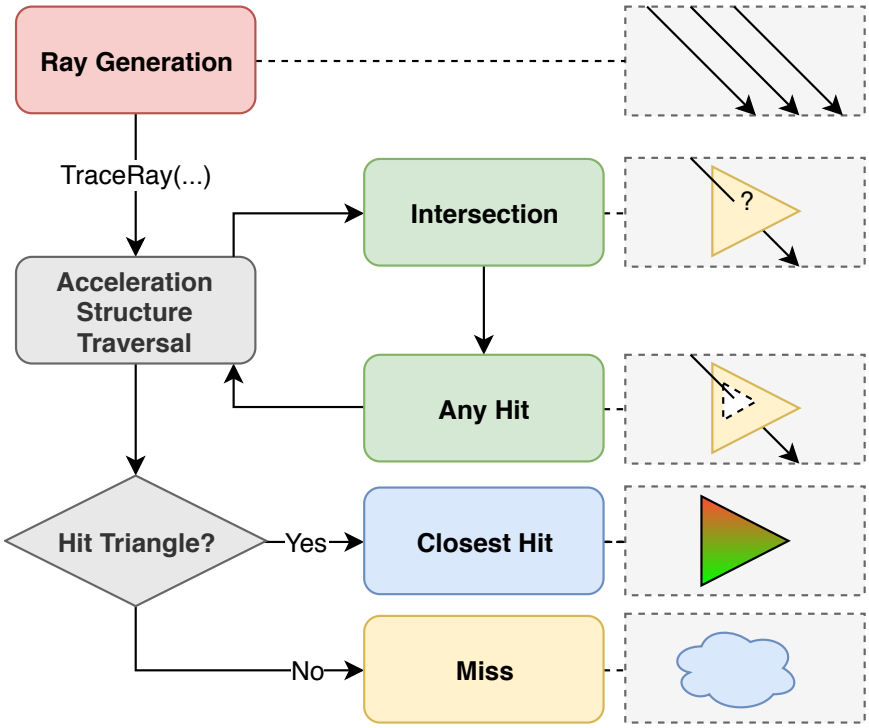


Obrázek 3.2: Tabulka shaderů umožňuje konfiguraci množiny shaderů a jejich vstupních parametrů. Každá tabulka obsahuje malé množství shaderů – např. *Ray Generation* tabulka obsahuje pouze jeden záznam, používající *Ray Generation* shader.

Spolu s konceptem tabulek shaderů představuje *DXR* také nové typy shader programů. Diagram jejich propojení lze vidět na obr. 3.3, přičemž jejich funkce jsou následující [1]:

- **Ray Generation** shader: Je vždy spuštěn jako první – počet instancí je dán nastavením příkazu pro spuštění sledování paprsku (sekce 3.3.4). Jeho cílem je generování primárních („kamerových“ nebo jiných) paprsků, přičemž každému z nich nastavuje *počátek*, *směr*, *časový interval* a počáteční hodnotu *payload*. Každá instance může obecně vyslat neomezené množství paprsků do vybrané scény. Mezi další zajímavé parametry vyslání paprsku patří například hodnota masky, která umožňuje některé objekty ignorovat. Tento typ shaderů musí v pipeline existovat **právě jednou**.
- **Intersection** shader: Instance tohoto shaderu vznikají vždy, když je nutné provést kontrolu průniku paprsku s některým typem primitiv. Obecně může tento shader řešit otázku průniku s libovolným primitivem a je tedy možné použít například procedurální geometrii. Tento typ shaderu je **specifický** pro daný objekt a **není povinný**. Pokud není specifikován, potom bude pro primitiva typu trojúhelník a *Bounding Box* použit výrobcem specifikovaný způsob výpočtu – je tedy možné použít například hardwarové jednotky a výpočet může být rychlejší.

- **Any Hit** shader: Funkce tohoto typu shaderu je velmi podobná *Intersection* shadrům a jsou spouštěny vždy, když *Intersection* shader detekuje průnik. Jejich funkcí je jemnější detekce, zda opravdu došlo k průniku – například kontrola průhlednosti. Velkou výhodou jejich použití – stejnou funkci je možné implementovat i v *Intersection* shaderu – je zachování použití výrobcem specifikovaného řešení průniku s primitivy. Tento typ shaderu je opět **specifický** pro daný objekt a **není povinný**. Pokud není specifikovaný, je na otázku zda došlo ke kolizi automaticky předpokládána kladná odpověď.



Obrázek 3.3: Schéma funkce pipeline pro sledování paprsku v *DirectX Ray Tracing*. Jednotlivé shadery jsou spouštěny podle schématu, přičemž některé nemusí být specifikovány (**zeleně**).

- **Closest Hit** shader: V případě, že průchod akcelerační strukturou dojde až do listů stromu a je nalezen průnik s některým z těchto primitiv (trojúhelníků), potom následuje spuštění shaderu *Closest Hit*. Jeho primární funkci lze přirovnat k *pixel* (nebo *fragment*) shaderům, protože generuje výslednou „barvu“ daného paprsku. Ve skutečnosti je možné aby paprsek nesl i jiné hodnoty než barvu, protože typ **payload** lze programově definovat. Tento shader je hlavním důvodem pro existenci tabulek shaderů, jelikož umožňuje obarvovat jednotlivé objekty (materiály) pomocí různých shaderů a je tedy **specifický** pro daný objekt. Kromě *payload* je také tomuto shaderu předána další struktura s atributy, které například pro trojúhelník definují pozici pomocí barycentrických souřadnic. Na rozdíl od dvou výše uvedených shaderů je z něj také možné **vrhat nové paprsky** – až do předem definované hloubky zanoření. Tento shader **musí být definován** pro každý objekt.

- **Miss** shader: Pokud nedojde k nalezení průniku s listovým primitivem (trojúhelníkem), potom následuje spuštění tohoto shaderu. Tento shader musí být opět definován **právě jednou** a umožňuje **vrhání nových paprsků**. Příkladem jeho použití může být vykreslení oblohy (*Skybox*).

- **Callable** shader: Tento typ shaderu, který není nutně začleněn do výsledné pipeline, lze použít pro organizaci společného kódu. Po jeho registraci je možné ho volat z shaderů typu *Ray Generation*, *Miss* nebo *Closest Hit*.

Posledním důležitým konceptem jsou skupiny shaderů přiřazované vykreslovaným objektům (materiálům) – tzv. *Hit Groups*. Každá skupina se skládá ze slotů pro tři shadery: *Intersection*, *Any Hit* a *Closest Hit*, přičemž pouze poslední z nich je povinný. *Hit Group* je nejmenší jednotkou shaderů, kterou lze přiřadit k jednotlivým objektům scény. Každé skupině odpovídá jedna výše zmíněná tabulka shaderů, která spolu s odkazem na *Hit Group* obsahuje také parametry jednotlivých shaderů. Celkově tedy existují tři typy tabulek shaderů: *Ray Generation*, *Miss* a *Hit Group*. *Ray Generation* tabulka je omezena na jeden použitý shader, který je vybrán při spuštění operace sledování paprsků. Zbylé dvě tabulky mohou obsahovat neomezený počet záznamů, ze kterých je vybíráno při volání **TraceRay** funkce v shaderech. Funkce **TraceRay** umožňuje vyslání paprsku do vybrané scény, kterých může být shaderu předáno obecně větší množství. Součástí volání je navíc informace o použitých *Hit Groups* a index vybraného *Miss shaderu* z *Miss shader* tabulky.

3.3.2 Vykreslovací řetězec a jeho konfigurace

Podobně jako rasterizační protějšek je nutné pipeline pro sledování paprsku konfigurovat a následně sestavit, čímž jsou znemožněny další změny. Prvním konfigurovatelným parametrem jsou knihovny shaderů, které mapují názvy shaderů na jejich binární reprezentaci. Tyto shadery je nutné přeložit a definovat jejich exportovaná rozhraní, která mají podobu textových řetězců. Následně jsou všechny odkazy na shadery v dané pipeline realizovány pomoci odkazů na tyto řetězce.

Mezi další důležité možnosti konfigurace patří nastavení maximální délky výše zmíněného *payload* paprsků, které musí být nastaveno tak, aby pojalo největší strukturu definující tato data. Podobně je také nutné zadat velikost struktury atributů předaných *Closest Hit* shaderu. Posledním kritickým nastavením je maximální hloubka rekurze, která definuje kolikrát je možné se zanořit a vysílat další paprsky v *Closest Hit* a *Miss* shaderech.

3.3.3 Akcelerační struktury

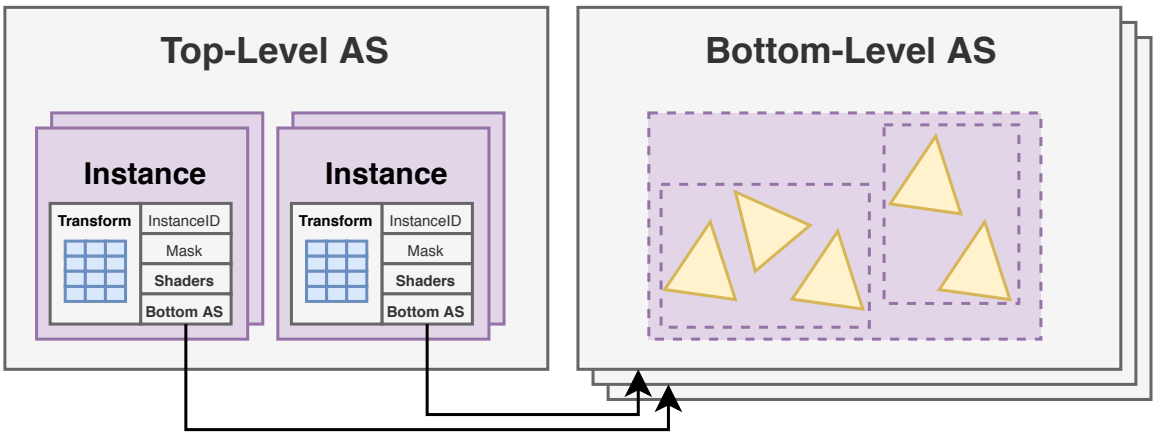
Pro sledování paprsku je nutné dopředu znát celou scénu, protože k ní paprsky přistupují náhodně. Z tohoto důvodu je před samotným spuštěním sledování paprsku nutné tuto scénu definovat, k čemuž jsou použity akcelerační struktury. Tyto struktury jsou v případě *DXR* rozděleny do dvou vrstev – *vrchní vrstva* a *spodní vrstva*.

Spodní vrstva, někdy také *modelová vrstva*, je použita k držení informací o vykreslovaných modelech. V jedné scéně může být těchto vrstev několik, kde každá může například obsahovat jeden model[4]. Specifické pro spodní vrstvy je, že jejich akcelerační struktury obsahují množiny vykreslovaných primitiv (trojúhelníků) v jejich listových uzlech.

[4]Pro dosažení nejvyšší výkonnosti je doporučeno aby počet vrstev byl co nejnižší, viz https://devblogs.nvidia.com/rtx-best-practices/.

Vrchní vrstva (*vrstva instancí*) obsahuje informace o instancích spodní vrstvy ve virtuální scéně – jedna spodní vrstva je tedy nejmenší jednotka umístitelná do scény. Listy vrchní vrstvy neobsahují trojúhelníky, ale speciální instanční uzly. Diagram propojení vrchní a spodní vrstvy lze vidět na obr. 3.4, kde instanční uzly obsahují následující informace:

- **Transformace**: Tato transformace je použita pro umístění instance do virtuální scény. Při průchodu kompletní akcelerační struktury je pomocí ní transformován paprsek ze světového souřadnicového prostoru do souřadnicového prostoru modelu. Transformace je reprezentována **maticí 4x3**, čímž jsou umožněny různé transformace, včetně translace.
- **ID instance**: Identifikátor, ke kterému je možno přistoupit z *Intersection*, *Any Hit* a *Closest Hit* shaderů. Lze ho tak použít například pro definici materiálu.
- **Maska**: Maska použitá pro vynechání některých instancí při vysílání paprsku do scény.
- **Shadery**: Specifikace *Hit Group*, která má být použita v případě zásahu této instance paprskem. Hodnota je upravena pomocí offsetu od začátku indexované tabulky shaderů, která je specifikována pro každé vyslání paprsku.
- **Model**: Model který má být umístěn do scény. Definovaný pomocí odkazu na akcelerační struktury spodní vrstvy. S pomocí informací o transformaci a modelu je potom možné stavět akcelerační strukturu vrchní vrstvy.

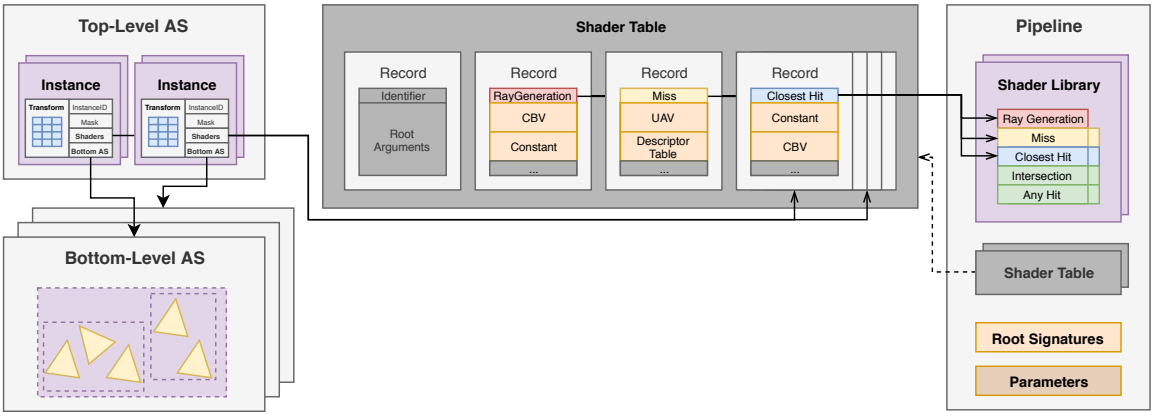


Obrázek 3.4: Akcelerační struktura je dvou-úrovňová, přičemž spodní vrstva obsahuje modely a vrchní vrstva instance modelů ze spodní vrstvy.

Stavba kompletní akcelerační struktury začíná postupným sestavením spodních vrstev ze zobrazované geometrie. Tato geometrie je definovaná stejně jako v případě rasterizační pipeline a to pomocí bufferu vertexů a množiny indexů do něj. Po dokončení stavby spodních vrstev jsou použity aktuální pozice jednotlivých objektů virtuální scény pro tvorbu vrstvy vrchní. Stavba vrchní vrstvy je mnohem méně výpočetně náročná, protože každý list obsahuje pouze jeden *Bounding Box*, namísto potenciálně vysokého množství trojúhelníků. Vrchní vrstvu je proto možné přegenerovat i jednou za vykreslený snímek, čímž je umožněno vykreslení pohyblivých scén.

3.3.4 Spuštění sledování paprsku

Po dokončení výše popsaných konfiguračních kroků, jejichž výsledkem je propojení celé architektury viditelné na obr. 3.5, lze přejít k samotnému výpočtu finálního syntetizovaného obrazu. Spuštění operace sledování paprsku je velmi podobné výpočtům pomocí *Compute* shaderů (*GPGPU*), kdy je na začátku nutné nastavit parametry tzv. vyvolání („Dispatch“). Podobně jako u *GPGPU* se zde nastavují parametry prostoru spuštěných instancí, které jsou umístěny do 3-dimenzionální mřížky definované třemi parametry – *šířka*, *výška* a *hloubka* [1].



Obrázek 3.5: Přehled architektury systému *DirectX Ray Tracing*. Diagram obsahuje jednotlivé části *DXR* – pipeline (**vpravo**), tabulku shaderů (**uprostřed**) a akcelerační struktury (**vlevo**) – a jejich propojení.

Dalším povinným parametrem jsou odkazy na použité tabulky shaderů pro *Ray Generation*, *Miss*, *Hit Group* a *Callable* shadery. Shadery mimo tyto definované tabulky nelze v aktuálním vyvolání používat. Posledním krokem před samotným spuštěním sledování paprsku je nastavení bufferů s akceleračními strukturami, které obsahují použité scény.

3.4 Vrstva zpětné kompatibility

Vrstva zpětné kompatibility *DXR* (*DXR Fallback Layer*) je knihovna[5] umožňující testování nových vlastností přidaných rozhraním *DXR* do *Direct3D 12* na grafických adaptérech které nepodporují hardwarově akcelerované sledování paprsku. Vrstva funguje na principu softwarové emulace potřebných funkcí, které jinak nejsou přístupné. Jejím cílem je umožnit vývoj a ladění aplikací využívajících hardwarovou akceleraci sledování paprsků na grafických akcelerátorech bez podpory této funkce.

Oproti některým nejasnostem ve specifikaci grafických karet *Nvidia Turing* [16], lze v této knihovně najít přesný popis veškeré funkcionality. Knihovna používá akcelerační strukturu typu *Bounding Volume Hierarchy*, kde obalové těleso je *Axis-Aligned Bounding Box*.

[5] Více informací a zdrojové kódy je možné najít v repozitáři projektu D3D12RayTracingFallback .

Kapitola 4

Návrh hybridního vykreslovacího enginu

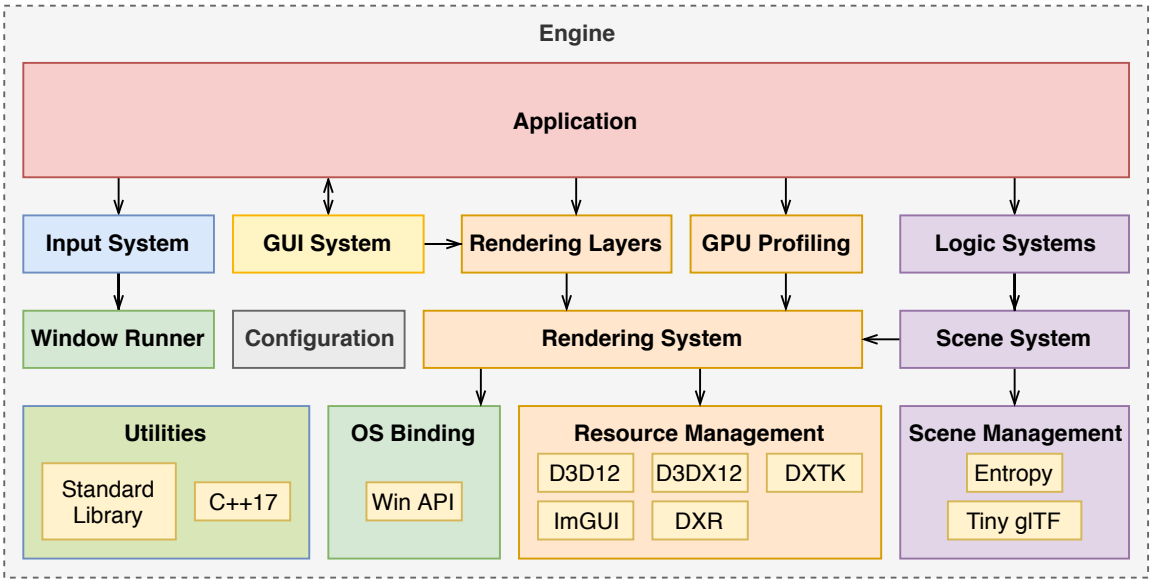
Tato kapitola se zabývá návrhem hybridního vykreslovacího enginu, který je dále v práci používán pro testování vlastností hardwarově akcelerovaného sledování paprsku. Engine je vytvořen za účelem maximální kontroly nad testovaným systémem, kterou by nebylo možné získat za použití již existujících enginů. Návrh je primárně zaměřen na vykreslovací systém podporující hardwarově akcelerované sledování paprsku a podporu komplexních scén s mnoha objekty.

Kapitola začíná přehledem architektury celého enginu, na kterém je znázorněno jak jsou jednotlivé části propojeny. Následují sekce podrobněji se zabývající jednotlivými částmi enginu a jejich návrhem. Postupně jsou zde popsány systémy pro profilování, propojení s operačním systémem, vykreslování, správu scén, uživatelské rozhraní a aplikační nadstavby.

4.1 Přehled architektury

Vykreslovací engine je navržen modulárně a skládá se z několika vzájemně komunikujících systému. Diagram zjednodušeného schématu celého enginu lze vidět na obr. 4.1. Každá část enginu je rozdělena do několika vrstev, na kterých jsou stavěny vrstvy vyšší. Ve spodní části diagramu lze vidět základní bloky, jejichž součástí jsou také knihovny, které budou použity při jejich implementaci. Jednotlivé tematické celky, kterým jsou věnovány následující sekce, jsou navíc barevně odlišeny.

Jako implementační jazyk vykreslovacího enginu byl zvolen programovací jazyk *C++* ve standardu *C++17*. Výhodou tohoto jazyka je jeho efektivita, statické typování a jeho časté využití pro vývoj aplikací používajících *DirectX* – obzvláště herních a vykreslovacích enginů. Kromě standardních knihoven jazyka *C++* bylo použito mnoho dalších podpůrných knihoven, důvody jejichž zvolení lze najít v příslušních sekcích této kapitoly.



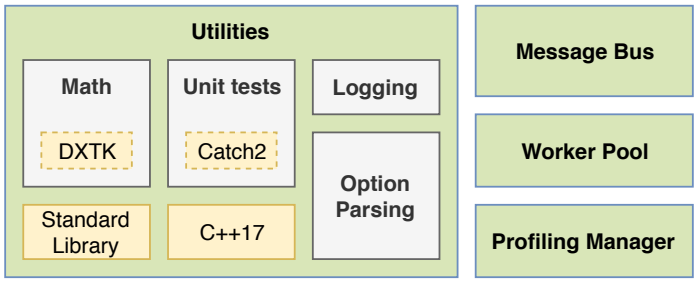
Obrázek 4.1: Diagram systémů tvořících výsledný vykreslovací engine a jejich propojení. Jednotlivé tematické celky jsou barevně odlišeny, kde například **fialově** jsou označeny systémy správy scén. Ve spodní vrstvě lze také vidět použité knihovny, jejichž pozadí je **žluté**.

Součástí enginu je několik celků, které mají podpůrnou funkci – nástroje (**zelená**), konfigurace enginu (**šedá**) a zpracování vstupu (**modrá**). Dále je zde také systém vazby na operační systém (**tyrkysová**), jehož primární funkcí je správa aplikačního okna umožňující zobrazení vykreslovaných scén. Kritickým celkem je systém vykreslování (**oranžová**), který využívá *Direct3D 12* pro syntetizaci obrazu. Cílem systému správy scén (**fialová**) je načtení scény a její uchování ve formě interní reprezentace enginu. Engine obsahuje také možnosti uživatelského vstupu (**modrá**) a grafického uživatelského rozhraní (**žlutá**). Poslední částí, která celý engine završuje a zpřístupňuje ho uživateli, je rozhraní aplikace (**červená**).

4.2 Profilování a obecné nástroje

Součástí návrhu jsou některé obecně používané třídy a struktury, které jsou shrnuty pod názvem „nástroje“. Diagram zobrazující rozsah těchto obecných nástrojů lze vidět na obr. 4.2. Zvolením implementačního jazyka *C++* odpadlo programování základních datových struktur a mnoha různých algoritmů, které jsou v enginu používány. Tyto algoritmy jsou součástí **standardní knihovny** jazyka **C++**.

Mezi obecné nástroje patří testování (*Unit Testing*), pro které byla zvolena knihovna **Catch2**. Tato knihovna je výhodná díky jednoduchosti nasazení a integraci s mnoha vývojovými prostředími. Testování grafických aplikací je obecně problematické a proto budou testovány pouze některé části kódu – například datové struktury, algoritmy a matematické funkce. Další obecnou částí enginu jsou matematické funkce, jejichž implementace je převážně ponechána na knihovně **DirectX ToolKit** (*DXTK*). Knihovna *DXTK* je nastavena pro maximální kompatibilitu s rozhraním *DirectX* a proto bude použita pro většinu matematických výpočtů.

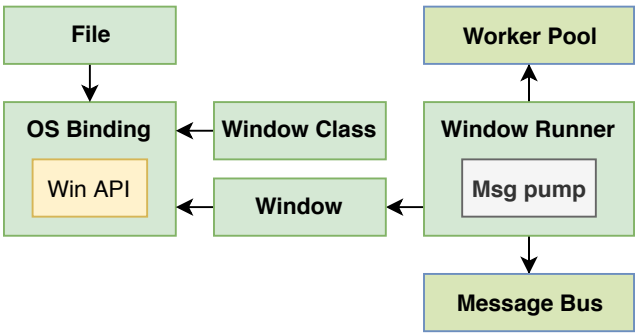


Obrázek 4.2: Diagram zobrazující obecné nástroje a utility, které nelze zařadit jinam v ekosystému výsledného enginu. Použité knihovny jsou zvýrazněny **žlutou** barvou.

Neméně důležitou částí je systém logování, který umožňuje uchovávání zpráv za běhu aplikace – pro okamžité vypsání do terminálu, nebo pro pozdější použití. Na tento systém navazuje manažer profilování, jehož cílem je uchovávat libovolná data měření aplikace. Mezi tato data patří například měření času zabraného některou částí enginu, nebo výsledky profilování na GPU. Cílem systému profilování je shromažďovat a následně zobrazovat data v reálném čase, aby bylo možné srovnávat výkonnost jednotlivých řešení. Poslední kritickou částí pro běh enginu je softwarová sběrnice pro předávání správ. Tato sběrnice pracuje na principu poštovních schránek (*Publish-Subscribe*), kdy se jednotlivé koncové body mohou registrovat pro zápis zpráv na sběrnici, nebo si vytvořit schránku. Tento systém předávání zpráv byl navržen pro více-vláknové prostředí a je silně typovaný – pro každý typ zpráv existuje oddělená sběrnice.

4.3 Vazba na operační systém

Tato část enginu je zodpovědná za komunikaci s operačním systémem vytvořením vrstvy abstrakce, která agreguje různá chování pod jednotné rozhraní. Kromě funkcí zobrazených na obr. 4.3, spadá pod tento systém například i textový výstup. Za hlavní funkci lze považovat správu oken, která je dále použita pro zobrazování výstupu a získávání uživatelského vstupu. K tomuto účelu je použita knihovna *Windows API* (*WinAPI*), která umožňuje přístup k funkcím operačního systému *Microsoft Windows*.



Obrázek 4.3: Diagram systému zabezpečujícího komunikaci s operačním systémem.

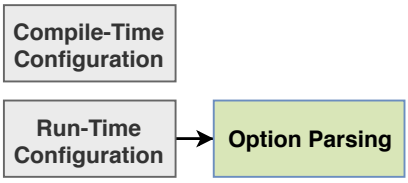
Zajímavou částí systému práce s okny v *Microsoft Windows* je vytvoření tzv. „čerpadla zpráv“ (*Message Pump*), které musí periodicky shromažďovat zprávy zasílané operačním systémem. Tato funkce je na diagramu schována pod názvem *Window Runner*. Z důvodu

lepší organizace vláken a rozdělení práce je jedno celé vlákno vyhrazeno pro periodické spouštění čerpadel zpráv. Problémem použití odděleného vlákna je, že toto čerpadlo musí vždy běžet na stejném softwarovém vlákně, na kterém bylo vytvořeno původní okno. Stejné omezení platí pro všechna nastavení oken (maximalizace, změna velikosti, atp.) a proto bylo nutné vytvořit asynchronní rozhraní okna, které komunikuje s jeho vlastnickým vláknem.

Spolu s mnoha různými typy zpráv jsou čerpadlu předávány také zprávy s uživatelským vstupem – myš, klávesnice a další. Tyto typy zpráv jsou předávány přes výše popsanou softwarovou sběrnici správ, aby mohly být zpracovány v uživatelské aplikaci.

4.4 Konfigurace enginu

I přes poměrně úzké zaměření navrhovaného enginu je výsledný systém relativně komplexní a proto je do návrhu začleněn také systém konfigurace. Tato konfigurace je prováděna na dvou úrovních, podle času kdy ji lze měnit – za překladu (*Compile-Time*) a za běhu (*Run-Time*). Výhodou oddělení těchto dvou částí jsou možnosti optimalizace ze strany překladače a možnosti větších změn v rámci enginu. Nastavení měnitelné za běhu lze potom načítat například z příkazové řádky při spuštění, což lze také vidět na diagramu systému na obr. 4.4.

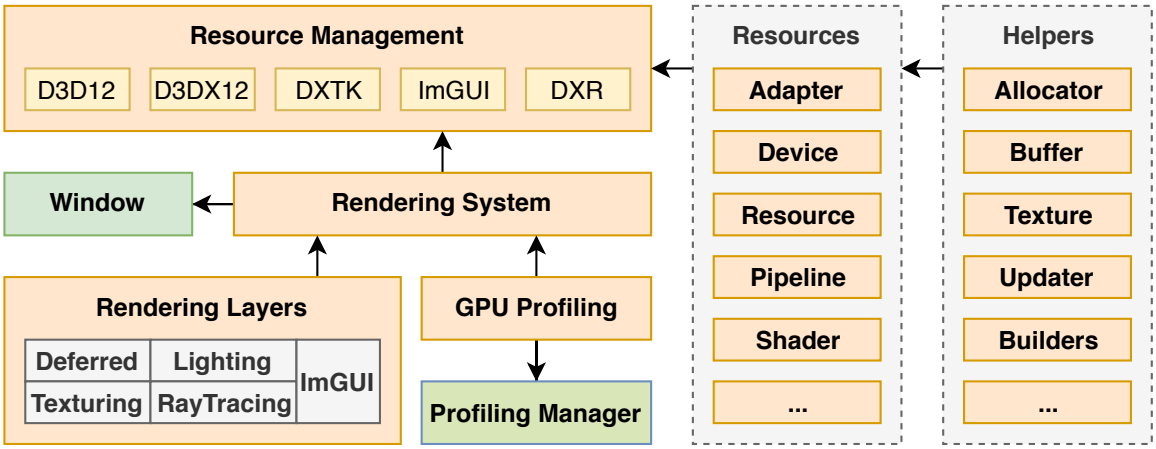


Obrázek 4.4: Rozdělení konfigurace enginu na dobu překladu (*Compile-Time*) a dobu běhu (*Run-Time*).

4.5 Vykreslovací systém

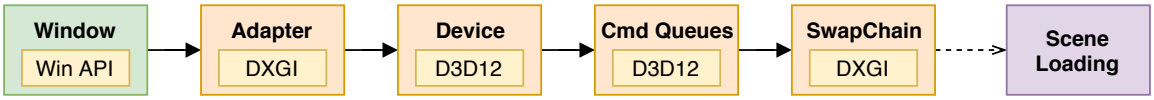
Cílem vykreslovacího systému je umožnit použití grafických adaptérů pro vykreslování a zobrazování syntetizovaných obrazů a jako takový zodpovídá za veškeré zdroje spojené s grafickým akcelerátorem. Tento systém je nejrozsáhlejší částí navrženého vykreslovacího enginu a jeho diagram lze vidět na obr. 4.5. Návrh je založen na použití výše zmíněných knihoven *DirectX 12*, mezi které patří: *Direct3D 12*, *D3DX12*, *DirectX ToolKit* a *DirectX Ray Tracing*. Přímo také využívá již popsaného systému oken a knihovny *ImGUI*, použité pro vykreslování uživatelského rozhraní.

Ústřední částí, na které celý systém staví, je správa zdrojů grafického adaptéru. Každý zdroj je nutné alokovat a po dokončení jeho používání ho opět uvolnit. Mezi nejdůležitější zdroje patří *adaptér* a *zařízení*, které společně reprezentují používaný akcelerační hardware. Příklady dalších zdrojů lze vidět v pravé části (**Resources**) diagramu 4.5. Kromě zdrojů je zde také přidán pojem pomocných tříd (**Helpers**), které pracují jako adaptéry nad zdroji a agregují některé často používané operace pod přehledné rozhraní.



Obrázek 4.5: Diagram zobrazující návrh systému vykreslování.

Vykreslovací vrstvu je před použitím nutné inicializovat, čímž jsou vytvořeny všechny nutné zdroje a dochází také například k vybrání cílového adaptéru (fyzického zařízení). Inicializace postupuje v pořadí zleva, podle diagramu na obr. 4.6. Při inicializaci jsou navíc spouštěny různé možnosti ladění, podle výše popsaného systému nastavení.



Obrázek 4.6: Postup inicializace vykreslovacího systému – začíná vytvořením okna a navazuje na něj načtení scény. Jednotlivé prvky obsahují knihovny, ze kterých byly vytvořeny (**žlutě**).

Samotné vykreslování je rozděleno do tzv. „vykreslovacích vrstev“ (*Rendering Layers*). Úkolem jednotlivých vrstev je realizace specifických efektů a zároveň jsou použity pro organizaci kódu. Součástí kódu jednotlivých vrstev jsou také shadery. Každá vrstva má předem definované vstupní a výstupní zdroje, které jsou nastaveny při její konstrukci. Příkladem může být vrstva vykreslení uživatelského rozhraní (pomocí *ImGUI*), jejíž vstupem je výstupní syntetizovaný obraz. Tato vrstva následně provede vykreslení GUI přes vstupní obraz a uloží ho do bufferu, který již lze zobrazit ve výstupním okně.

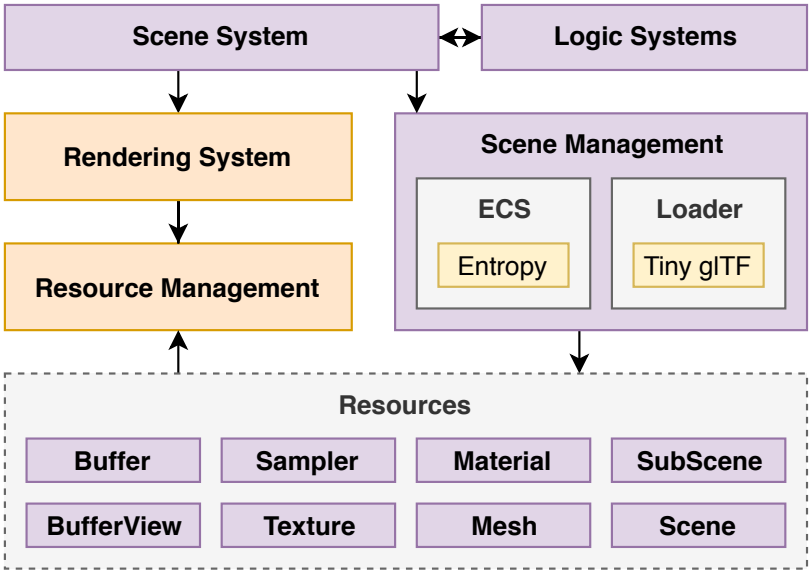
Posledním zajímavým tématem vykreslovacího systému je profilování algoritmů běžících na GPU. Výstupy časovačů získaných na grafickém adaptéru jsou opět ukládány do výše popsaného manažera profilování. Díky tomu je možné zobrazovat profilovací informace jednotně a tedy i porovnat celkové zatížení výpočetních prostředků při výpočtu.

4.6 Správa scén

Druhým nejrozsáhlejším systémem navrženého enginu je systém správy scén, který je použit pro načítání, správu a aktualizaci virtuálních scén. Diagram zobrazující jednotlivé komponenty navrženého systému lze vidět na obr. 4.7. Celý systém správy scén je postaven na použití *Entity-Component-System* (*ECS*) paradigmatu, čehož je dosaženo použitím knihovny *Entropy*[1].

[1]Knihovna implementována součástí předchozí práce autora [18].

Základní myšlenkou *Entity-Component-System* paradigmatu je rozdělení virtuálních objektů scény na *Entity*, které jsou pouze abstraktními kontejnery pro *Komponenty* [18]. Úkolem *Komponent* je držení dat nutných pro reprezentaci entity – například model, barva, velikost nebo pozice ve scéně. Poslední částí tohoto paradigmatu jsou *Systémy*, které přistupují k *Entitám* a jejich *Komponentám* a provádějí nad nimi transformace.

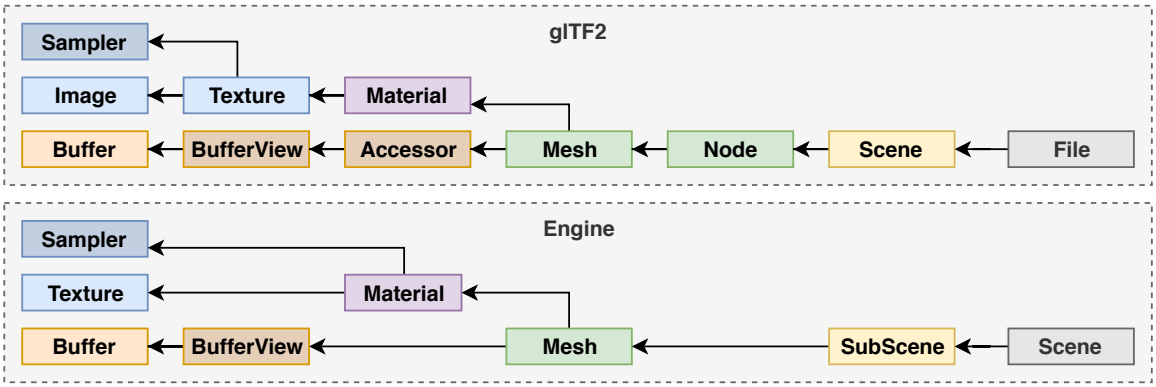


Obrázek 4.7: Diagram návrhu systému pro správu scén. Použité knihovny jsou obsaženy ve **žlutých** uzlech.

Interní reprezentace scény je uložena v *Entity-Component-System* struktuře. Nad touto strukturou je množina zdrojů (obr. 4.7, **Resources**), které jsou používány pro definici vzhledu vykreslených entit. Za účelem ukládání těchto zdrojů pro jejich pozdější využití při vykreslování je použito správy zdrojů vykreslovacího systému.

Důležitým tématem při návrhu enginu je také načítání a reprezentace scén. Za tímto účelem byl zvolen formát *glTF 2.0*[2]. Výhodou tohoto formátu je otevřený standard a jeho rozšířené používání v mnoha dalších projektech. Formát je založen na textové reprezentaci (*JSON*) kombinované s binárními daty pro textury, vertexy a další. Pro ulehčení práce se soubory *glTF* je použita knihovna *TinyGlTF*, která zprostředkovává analýzu souborů scény a jejich načítání. Samotná interní reprezentace scény v enginu je inspirována hierarchií použitou v *glTF* – srovnání obou hierarchií lze vidět na diagramu na obr. 4.8.

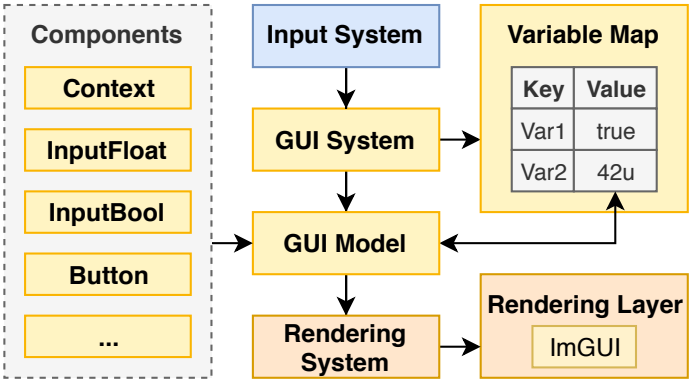
[2]Viz https://github.com/KhronosGroup/glTF/blob/master/specification/2.0 .



Obrázek 4.8: Srovnání hierarchií zdrojů použitých v *glTF* a interní reprezentace v navrženém vykreslovacím enginu.

4.7 Uživatelský vstup

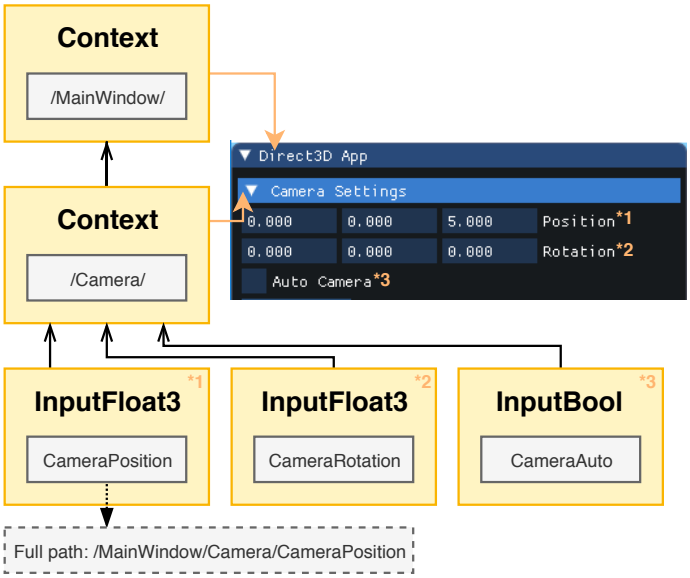
Mezi dva hlavní způsoby uživatelské interakce s aplikací patří vstup z periferií – klávesnice, myš atp. – a grafické uživatelské rozhraní (*GUI*). Propojení těchto dvou systémů a jejich návrh lze vidět na obr. 4.9. Vstupní systém zpracovává výše zmíněné zprávy z čerpadla zpráv a předává je přes sběrnici ostatním systémům v enginu. Na tyto zprávy je následně možno reagovat přímo, pomocí vstupního mapování, které umožňuje pro každou akci – např. stisk tlačítka – asociovat požadovanou reakci. Tyto reakce jsou definovány pomocí *lambda* funkcí, čímž je programátorovi umožněna volnost v jejich implementaci.



Obrázek 4.9: Diagram návrhu systému grafického uživatelského rozhraní. Systém umožňuje oboustranné propojení prvků GUI s proměnnými. Pro výsledné vykreslení je použita knihovna *ImGui*.

Druhým ze vstupních systémů je systém grafického uživatelského rozhraní. Tento systém, založený na modifikovaném návrhovém vzoru *Model-View-Viewmodel*, umožňuje tvorbu jednoduchých uživatelských rozhraní a synchronizaci hodnot. Mezi základní prvky patří *komponenty*, které reprezentují jednotlivé prvky uživatelského rozhraní – tlačítko, číselný vstup atp. Na druhé straně jsou proměnné, uložené v mapě proměnných, kterou lze považovat za jednoduchou databázi. Proměnné jsou uloženy jako páry klíč-hodnota, kde klíčem je název proměnné.

Tyto dva koncepty propojuje model uživatelského rozhraní, který obsahuje aktuální *komponenty* a jejich mapování na proměnné. Komponenty jsou uspořádány do stromů, kde kořenem je vždy nějaký typ okna. Speciálním typem komponent jsou kontexty, které umožňují uspořádání komponent do skupin, jejich zanořování a tvorbu oken. Hierarchie kontextů a jejich vnořených komponent je možné vyjádřit jako cesty v souborovém systému, příklad lze vidět na obr. 4.10.



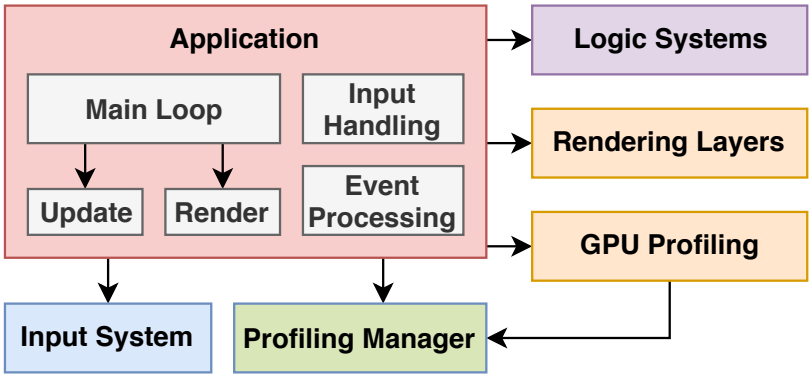
Obrázek 4.10: Příklad *GUI* vytvořeného ve výsledném enginu. Na obrázku lze vidět stromovou strukturu komponent a kontextů (**vlevo**) a odpovídající uživatelské rozhraní (**vpravo**). Jednotlivé prvky lze adresovat jejich plnými cestami, podobně jako v souborových systémech.

Model uživatelského rozhraní je předán odpovídající zobrazovací vrstvě, která ho vykreslí. Pro vykreslení a implementaci jednotlivých komponent je použita knihovna *ImGui*.

4.8 Aplikace

Poslední vše-završující částí navrženého enginu je rozhraní aplikace, které umožňuje tvorbu programů používající výsledný engine. Rozhraní je navrženo tak, aby bylo použitelné jednoduchým děděním hlavní třídy **Application**. Následně jsou implementovány některé z před-připravených funkcí, čímž je specifikováno chování vytvořené aplikace. Tato nová aplikační třída je následně předána hlavnímu enginu, který ji začlení do okolního systému a spustí její hlavní funkci. Diagram rozhraní aplikace a její propojení se zbytkem enginu lze vidět na obr. 4.11.

Ústřední částí aplikace je hlavní programová smyčka, která zodpovídá za periodické provádění operací vykreslování a případných aktualizací virtuální scény. Dále aplikace obsahuje zpracování vstupu od uživatele, které lze vidět na diagramu propojení aplikace na obr. 4.11. Tento systém zpracovává zprávy o vstupních událostech zasílaným výše popsaným čerpadlem zpráv. Pro každou zprávu je možné nastavit odpovídající akci, čímž je umožněno mapování vstupů na akce ve virtuálním světě. Další nezbytnou částí aplikačního rozhraní je zpracování událostí, které jsou opět zasílány přes softwarovou sběrnici zpráv. Mezi tyto zprávy patří například informace o požadavku na ukončení aplikace, změna velikosti vykreslované plochy nebo události myši a klávesnice.



Obrázek 4.11: Diagram rozhraní aplikace a její zapojení do vykreslovacího enginu. Implementace některých částí (**šedě**) je ponecháno na uživateli.

Kapitola 5

Implementace a experimenty

Cílem této kapitoly je popis některých částí implementace výše navrženého vykreslovacího enginu nazvaného *Quark*, který je dále použit pro testování vlastností akcelerovaného sledování paprsku. Kapitola se také zabývá návrhem a implementací testovacích scénářů a experimentů, jejichž hlavní prioritou je umožnit automatické testování s různými vstupními podmínkami. Zdrojové kódy výsledného enginu jsou otevřené a je možné k nim získat přístup skrz repozitář projektu[1].

Výše popsaný návrh zobrazovacího enginu lze shrnout do pěti částí. První z nich je systém vykreslování, jehož implementaci je věnována většina této kapitoly. Druhým důležitým systémem je správa scén, jejímž cílem je umožnit načítání scén – z interní reprezentace, nebo ze souborů formátu *glTF* – a generování seznamů entit pro vykreslovací systém. Do třetí kategorie se řadí systémy pro uživatelský vstup z periferií nebo skrz *GUI*. Dále jsou zde také systémy pro profilování programu na *CPU* a *GPU*, logování výsledků a další podpůrné utility. Poslední, vše završující částí je systém aplikace, jehož cílem je umožnit uživateli enginu tvorbu aplikací.

5.1 Sledování paprsku

Pro akceleraci sledování paprsků je v této práci použito rozhraní *DirectX Ray Tracing*, jehož hlavní prerekvizitou je funkční vykreslovacího řetězce využívající rozhraní *Direct3D 12*. Implementace *Direct3D 12* je i přes její časovou náročnost již popsána v mnoha jiných projektech a proto se zbytek této sekce zabývá specifiky akcelerace sledování paprsků a rozhraním *DirectX Ray Tracing* (*DXR*).

Implementace je rozdělena do dvou částí nazvaných *Fallback cesta* a *Hardware cesta*. Rozdělení umožňuje testovat jak čistý systém akcelerovaného sledování paprsku na podporovaných grafických akcelerátorech tak i *Fallback* vrstvu, která je zpětně kompatibilní se staršími zařízeními. Při implementaci bylo nutné pro tyto dva systémy vytvořit společnou vrstvu abstrakce, díky které nezáleží na použité technologii. Engine také obsahuje čistě rasterizační implementaci, která umožňuje spuštění aplikace i bez podpory *Fallback vrstvy* – v tomto případě ale není možné vykreslovat efekty generované pomocí sledování paprsku.

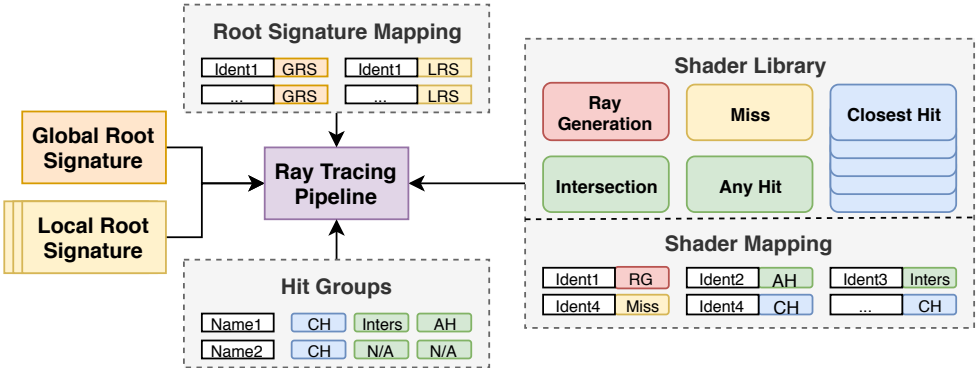
Pro použití nových struktur a tříd z knihovny *DirectX Ray Tracing* je nutná relativně aktuální verze operačního systému Windows 10 – počáteční podpora je k dispozici ve verzi 1809. Následně je možné použít vyšší verze tříd **Device** a **GraphicsCommandList**, které zpřístupňují metody pro nastavení a spuštění sledování paprsku.

[1] Viz https://gitlab.com/tomtomp/Masters-DXR-Main/tree/Integration .

Při implementaci a následném testování se objevilo několik nekonzistencí mezi *Fallback* a *Hardware* implementacemi, které se projevovaly chybami při vykreslování. Zajímavou vlastností *Fallback* implementace je pokles výkonnosti vznikající při vrhání osově zarovnaných paprsků, se kterým se lze například setkat při ortografické projekci. Profilování odhalilo příčinu tohoto problému v implementaci *Fallback vrstvy*, kde jeden z výpočetních shaderů generuje singularity při detekci průniku paprsků s osově zarovnanými obalovými kvádry (*AABB*). Mezi další problémy patřily také chyby překladu shaderů pro sledování paprsků, které vznikaly skrz proces kompilace tzv. *Uber Shaderů*. V některých případech bylo nutné kód přepsat do jiné podoby, aby například nebylo vyslání paprsku podmíněné hodnotou proměnné. I přes tyto chyby je ale *Fallback vrstva* stále velmi užitečná pro vývoj a testování algoritmů, i přes její omezenou výkonnost.

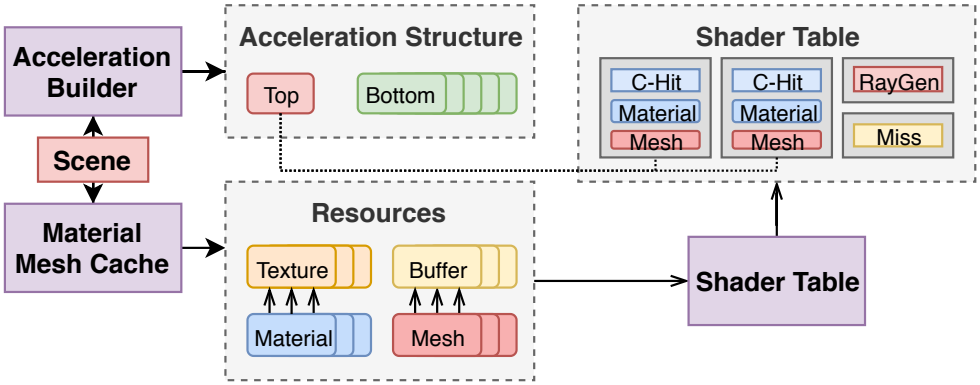
Pro jednodušší používání obsahuje engine několik pomocných tříd, které umožňují konfiguraci a přípravu výpočtů pomocí sledování paprsku. Prvním krokem je vytvoření tzv. *Kořenových signatur* (*Root Signatures*), které definují uniformní rozhraní shader programů. V případě pipeline pro sledování paprsků je možné těchto signatur vytvořit několik a rozdělit je do dvou kategorií – globální a lokální. Globalní signatura je pouze jedna a musí být stejná pro celou sestavu shaderů [1] (sekce 3.3). Data globálních signatur jsou předávána stejným způsobem, jako v případě rasterizační pipeline, skrze nastavení v příslušném seznamu příkazů, před spuštěním operace výpočtu. Lokálních signatur je možno vytvořit několik, avšak maximálně jednu pro každý typ shader programu v pipeline sledování paprsku a jsou následně navázány na daný typ shaderu. Hlavním rozdílem oproti globálním signaturám je způsob, jakým se jim předávají data a omezení na ně vztažená. Vstupní data jsou lokálním signaturám předávána skrz tabulky shaderů (sekce 3.3). Tento způsob umožňuje například specifikovat specifické textury a buffery pro každý model ve scéně. Zajímavou vlastností je také neomezená velikost těchto dat, oproti globálním signaturám [1]. Jejich hlavní nevýhodou je nutnost dynamického načítání těchto dat, které je v režii grafického akcelerátoru, což ve výsledku znamená nižší výkon při použití většího množství dat. Pro stavbu obou typů signatur jsou vytvořeny pomocné třídy (**RootSignature** a **RootSignatureBuilder**), které umožňují jejich jednoduchou tvorbu podle zadaných požadavků.

Druhým krokem je konfigurace a stavba pipeline pro sledování paprsku. Pipeline lze považovat za množinu nastavení, které po její finalizaci již nelze měnit. Tento přístup je u nízkoúrovňových API běžný, z důvodu možnosti optimalizací na straně ovladače grafického adaptéru. Konfigurace pipeline pro sledování paprsku je oproti rasterizační pipeline velmi rozdílná – diagram celého procesu lze vidět na obr. 5.1. Z důvodu potenciálního použití mnoha různých implementací *Closest Hit* shaderů – například různé typy materiálů – jsou shadery přidávány skrz tzv. *knihovny shaderů*. Tyto knihovny, kromě přeloženého kódu shaderů, obsahují také mapování jejich vstupních bodů (hlavních funkcí) na unikátní identifikátory, které jsou reprezentovány znakovými řetězci. Toto mapování se dále používá k odkazu na již přidané shader programy. Výše zmíněné signatury jsou do pipeline přidány a asociovány s shadery které je používají. Shadery jsou také rozděleny do skupin – tzv. *Hit Groups* – kde každá skupina reprezentuje jeden typ geometrie ve scéně (například materiál). Poslední částí konfigurace pipeline sledování paprsku je nastavení parametrů, mezi které patří například maximální hloubka rekurze nebo velikost datové struktury jednotlivých paprsků. Celý tento proces je opět zabalen v pomocné třídě **RTPipeline**, která mimo jiné umožňuje jednodušší mapování shaderů na cílové signatury.



Obrázek 5.1: Diagram částí nutných pro konfiguraci pipeline pro sledování paprsku. Kromě samotných shaderů, uložených v knihovných shaderů, je také nutná definice jejich lokálních a globálních signatur. Shadery jsou po přidání odkazovány pomocí jejich identifikátorů.

Po dokončení prvních dvou kroků končí fáze počáteční inicializace. V průběhu samotného vykreslování již není vhodné pipeline ani signatury často měnit z výkonnostních důvodů[2]. Následující kroky je v jisté míře nutné opakovat pro každou změnu ve vykreslované scéně, množství změn záleží na rozdílu mezi konfiguracemi scén – například pro pohyb objektů ve scéně je nutné změnit pouze vrchní vrstvu akcelerační struktury.



Obrázek 5.2: Diagram přípravy pro vykreslení vstupní scény pomocí pipeline sledování paprsku. Vstupní geometrie je použita pro sestavení akcelerační struktury. Dále je vytvořen seznam použitých textur a bufferů, ze kterých jsou vytvořeny specifikace jednotlivých geometrických sítí (*meshů*) a materiálů. Posledním krokem je naplnění tabulky shaderů, která obsahuje data pro lokální shader signatury.

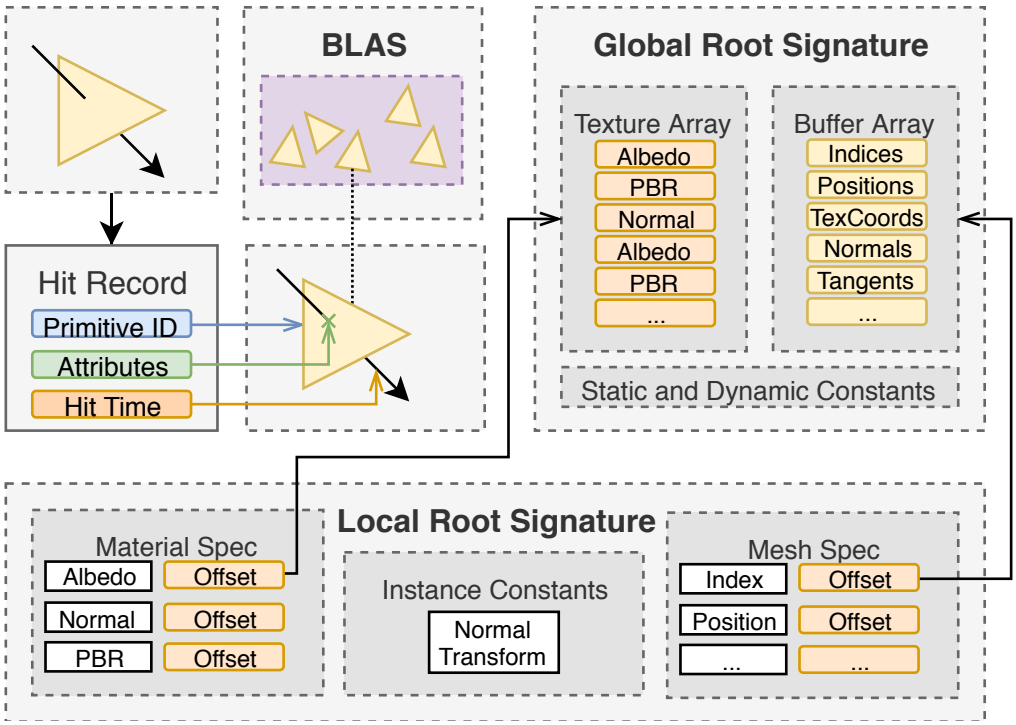
Třetím krokem je příprava akceleračních struktur pro zobrazované modely – diagram tohoto procesu lze vidět na obr. 5.2. Nejdříve je nutné vytvořit seznam použitých trojúhelníkových sítí a jejich vertex bufferů. Pro každý model je následně vytvořena jedna akcelerační struktura spodní úrovně, obsahující jeden nebo více meshů. Po zpracování všech modelů je vytvořena akcelerační struktura vrchní úrovně, která vytváří asociaci mezi modelem ve spodní vrstvě a jeho pozicí ve scéně pomocí transformační matice. Pro úlohu stavby akceleračních struktur bylo opět vytvořeno několik pomocných tříd – **RTBottomLevelAS**,

[2]Viz https://devblogs.nvidia.com/practical-real-time-ray-tracing-rtx/ .

RTTopLevelAS a **RTAccelerationBuilder** – jejichž cílem je kromě samotné stavby také caching již vytvořených struktur a nastavení jejich parametrů.

Krok čtvrtý se podobně jako stavba akceleračních struktur zabývá přípravou vstupní scény k zobrazení. V tomto případě jde ale o samotné buffery a textury, ke kterým je přistupováno z shader programů. Pro každý aktuálně viditelný model ve scéně jsou shromážděny jeho textury a buffery – pomocí třídy **MaterialMeshCache** – ze kterých jsou vytvořeny dva seznamy. Následně je pro každý materiál vytvořena specifikace, která udává pozici jeho textur v seznamu textur. Podobně je také zpracován seznam bufferů, ze kterého jsou vytvořeny specifikace meshů. Příklad těchto seznamů lze vidět na obr. 5.3 vpravo nahoře.

Poslední přípravnou částí je vytvoření tabulky shaderů, která specifikuje použité shadery a data předaná jejich lokálním signaturám. V případě tohoto enginu jsou zde uloženy výše zmíněné specifikace materiálů a meshů. Pro správu tabulky shaderů je opět použita pomocná třída – **RTShaderTable** – která automaticky tabulku také nahraje na grafický akcelerátor. Po dokončení všech přípravných kroků je nyní možné spustit sledování paprsku.



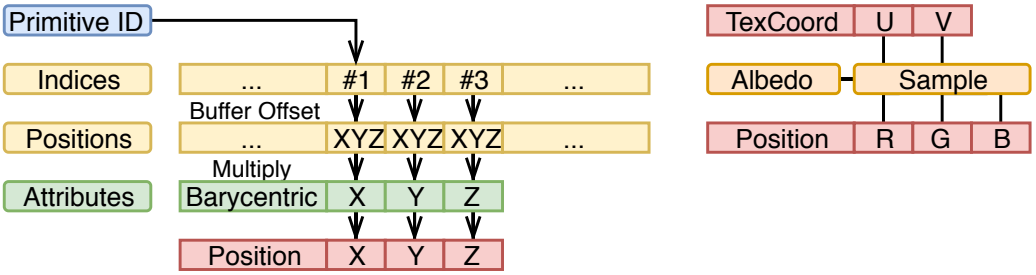
Obrázek 5.3: Ukázka parametrů předaných shaderům pro sledování paprsku, ze kterých je možné získat interpolovaná data, jejichž hodnoty jsou automaticky generovány v případě rasterizačních shaderů.

Po nastavení parametrů globální signatury a nastavení použité pipeline konfigurace je proveden tzv. *dispatch* který, podobně jako v případě *Compute* shaderů, vyžaduje specifikaci počtu vypuštěných vláken – například rozměry výstupního bufferu.

Samotná implementace shaderů pro sledování paprsků je také zajímavá. Oproti rasterizaci, kdy například *Vertex* shader automaticky získá vertex atributy jako pozice a normála, je nutné u shaderů sledování paprsku tyto informace vypočítat manuálně. Na obr. 5.3 lze vidět všechny parametry, jejichž použití je dále popsáno.

Celý proces začíná při zásahu geometrie některým z vyslaných paprsků, čímž jsou vygenerovány informace o zásahu. Mezi nejdůležitější z nich patří: *Index primitiva*, *Atributy*

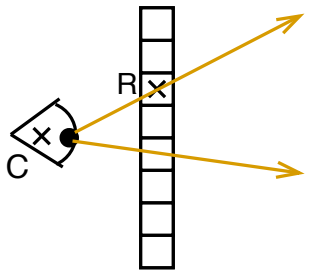
zásahu a *Čas zásahu*. Index primitiva je použit jako index zasaženého trojúhelníku. Pořadí trojúhelníků je dáno pořadím primitiv v index bufferu a proto je z této informace možné získat právě tři indexy reprezentující vertexy zasaženého trojúhelníku. Atributy zásahu obsahují barycentrické koordináty, s jejichž pomocí lze zpřesnit místo zásahu a provést interpolaci vertex atributů (obr. 5.4). Čas zásahu udává jak dlouhá je úsečka od počátečního bodu k bodu zásahu. Pomocí získaných indexů z index bufferu lze získat další vertex atributy jakými jsou například pozice, texturovací koordináty, normála a tangenta. K texturám je přistupováno podobným způsobem, ale k jejich indexaci jsou použity již vypočítané texturovací koordináty.



Obrázek 5.4: Pomocí indexu primitiva a předaných bufferů a textur je možné dopočítat data, která jsou automaticky přístupná v rasterizačních shaderech.

5.2 Efekty využívající sledování paprsku

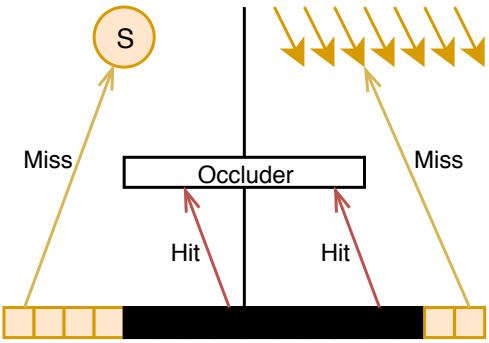
Před samotným popisem implementovaných efektů je nutné také zmínit implementaci primárních paprsků, které jsou v hybridním sledování paprsku nahrazeny vykreslením skrz rasterizační průchod. Pro vyslání paprsků je nutné získat informaci o jejich počáteční pozici a směru. Samotná operace začíná vytvořením instancí *Ray Generation* shaderů, které jsou rozloženy do mřížky o velikost výstupního rastru. Každá z těchto instancí, v případě „čistého“ sledování paprsku s vysíláním primárních paprsků, vypočítá svoji pozici ve výstupním rastru. Následuje transformace do prostoru tzv. *Normalized Device Coordinates*, čímž je získána pozice na plátně primární kamery. Posledním krokem je transformace získané pozice do světového souřadnicového systému pomocí inverzní matice kamery. Finální stav lze vidět na obr. 5.5. Z bodů C a R lze nyní již jednoduše vypočítat počáteční pozici paprsku a jeho směr.



Obrázek 5.5: Diagram zobrazující nastavení kamery a rastru, na kterém lze vidět body nutné pro výpočet parametrů kamerových paprsků

5.2.1 Tvrdé stíny

Prvním z implementovaných efektů využívajících akcelerované sledování paprsku jsou tvrdé stíny. Problém lze definovat jako určení viditelnosti světelného zdroje z dotazovaného bodu scény. Řešení pomocí sledování paprsku je velmi jednoduché a vyžaduje pouze jeden paprsek pro každé dotazované světlo. Nejdříve je nutné vypočítat směr a vzdálenost světelného zdroje od dotazovaného bodu. Následně je vytvořen paprsek ve směru světla, jehož délka je nastavena tak, aby se zastavil těsně před zdrojem světla. Podobně jsou implementována i směrová světla, u kterých je paprsek vysílán proti směru světelných paprsků. Diagram zobrazující tento proces lze vidět na obr. 5.6.



Obrázek 5.6: Diagram zobrazující implementaci tvrdých stínů za pomocí sledování paprsku. *Stínové paprsky* jsou vysílány ve směru světla, nebo proti směru směrového světla. Pokud dojde k zásahu geometrie, lze říci, že zdroj světla nelze vidět.

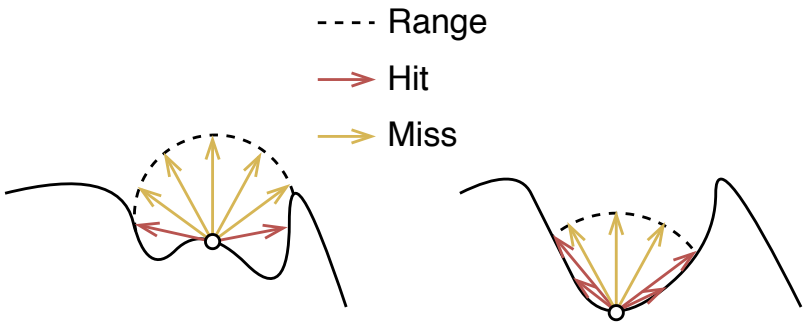
Pro implementaci jsou použity pouze *Ray Generation* a *Miss* shadery. Vysílání paprsků je nastaveno tak, aby nespouštělo *Closest Hit* shadery a přijalo pouze první zásah, čímž je docíleno vyšší výkonnosti[3]. Datová struktura každého *stínového paprsku* obsahuje pouze jednu pravdivostní hodnotu (typu *bool*), která určuje zda byla zasažena scéna. Tato hodnota je před vysláním paprsku nastavena na true – tedy předpokládáme zásah scény. Implementace *Miss* shaderu tuto proměnnou resetuje na hodnotu false. Po návratu lze tedy z hodnoty přímo detekovat, zda je světlo viditelné.

5.2.2 Ambient Occlusion

Podobným způsobem je implementován efekt ambient occlusion, který opět využívá *stínové paprsky*. Diagram zobrazující implementaci tohoto efektu lze vidět na obr. 5.7. Oproti tvrdým stínům je zde délka paprsků nastavena na předem danou hodnotu – tzv. *rozsah okluze*. Samotný výpočet ambient occlusion probíhá vysláním několika paprsků – podle požadované kvality, například 4 nebo 8 – kolem testovaného bodu. Směry těchto paprsků je nutné vhodně distribuovat ve 3D prostoru, k čemuž byl použit přístup podobný *SSAO* [13]. Generování náhodného směru je dosaženo pseudo-náhodným generováním bodů na hemisféře. Tato hemisféra je následně transformována tak, aby směřovala ve směru normály testovaného povrchu. Hodnoty jednotlivých paprsků – zastíněno nebo osvětleno – jsou postupně akumulovány a v posledním kroku algoritmu transformovány v průměrné zastínění daného bodu, které je získáno dělením celkové hodnoty akumulátoru počtem vyslaných paprsků a případným útlumem.

[3] Viz https://devblogs.nvidia.com/rtx-best-practices/ .

Kromě výše popsané realizace efektu ambient occlusion pomocí *stínových paprsků* obsahuje aplikace také alternativní implementaci, která využívá tzv. *vzdálenostní paprsky*. Tyto paprsky jsou rozdílné v tom, že jejich výsledkem není pouze binární hodnota zastínění, ale obsahují navíc vzdálenost okluderu. Tato hodnota je použita k útlumu výsledné hodnoty ambientní okluze, čímž vzniká jemnější efekt. Nevýhodou je vyšší cena jednotlivých paprsků.



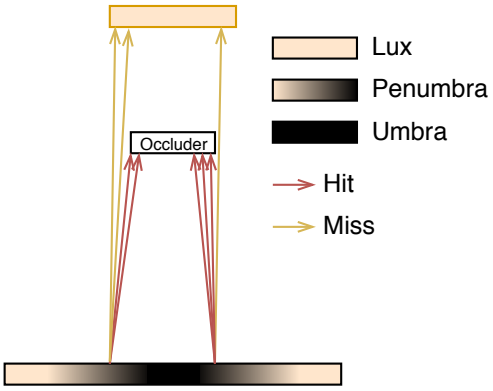
Obrázek 5.7: Diagram zobrazující implementaci efektu ambient occlusion za pomocí sledování paprsku. *Stínové paprsky* jsou použity podobně jako při generování tvrdých stínů, ale jejich délka je pevně nastavena.

5.2.3 Odrazy

Zajímavým efektem jsou spekulární odrazy na libovolně členitém povrchu. V zásadě jde o vysílání sekundárních paprsků, které se chovají podobně jako primární kamerové paprsky. Jejich implementace využívá stejné *Closest Hit* a *Miss* shadery. Struktura paprsků obsahuje kromě výstupní barvy, která je předána zpět volajícímu, také specifikaci efektů, které mají být počítány – stíny, ambient occlusion a odrazy. Pro primární paprsky jsou všechny tyto parametry nastaveny na zapnuto, ale v případě reflekčních paprsků jsou zapnuty pouze stíny. Směr odražených paprsků je dán zákonem odrazu, který v případě tohoto efektu počítá se směrem primárního paprsku a normálou povrchu.

5.2.4 Měkké stíny

Posledním implementovaných efektem jsou měkké stíny. Tento efekt opět používá *stínové paprsky*, jejichž nastavení je stejné jako v případě tvrdých stínů. Hlavním rozdílem je vyslání většího množství paprsků a výpočet průměrného výsledku. Směry jednotlivých paprsků jsou vždy posunuty o pseudo-náhodnou hodnotu tak, že tvoří kužel s uniformní distribucí. Tento přístup je velmi podobný *Monte Carlo* integraci, která v tomto případě počítá průměrné zastínění světelného zdroje. Diagram zobrazující tento pricip lze vidět na obr. 5.8.



Obrázek 5.8: Diagram zobrazující implementaci měkkých stínů za pomocí sledování paprsku. Vysláním několika *stínových paprsků* je provedena nepřesná integrace příchozího světla a následným rozmazáním je možné generovat přesnější měkké stíny.

5.2.5 Výsledky

Výsledné obrazy generované implementovanými efekty lze vidět na obr. 5.9. Tvrdé stíny ukazují velmi vysokou kvalitu, kdy je hodnota zastínění počítána přesně pro každý pixel kamery. Druhý obrázek ukazuje efekt měkkých stínů, kde stín v levé části obrazu je tvořen objektem blízkým. Stín tyče je znatelně více rozmazaný, díky její vzdálenosti od země. Dalším efektem je ambient occlusion, na kterém lze vidět efekt hlubokých záhybů. Poslední obrázek ukazuje zrcadlení na libovolně složitém objektu, ve kterém lze vidět obraz scény mimo aktuálně viditelnou geometrii.

Obrázek 5.9: Ukázka implementovaných hybridních efektů – tvrdé stíny (**první**), měkké stíny (**druhý**), ambient occlusion (**třetí**) a odrazy (**čtvrtý**).

5.3 Rasterizační efekty

Rasterizační efekty byly implementovány primárně z důvodu srovnání s jejich alternativami využívajícími sledování paprsku. Všechny implementované efekty jsou často používané ve vykreslovacích enginech, čímž je umožněno srovnání výkonu s výše popsanými alternativami.

Prvním z implementovaných efektů jsou tvrdé stíny, implementované technikou *Shadow Mapping* [6, Kapitola 15]. Jde o jednoduchou implementaci, která generuje mapy pro směrová světla. Metoda je implementována v základní variantě tak, jak byla popsána v sekci 2.5. Hlavním parametrem, udávajícím kvalitu a výkonnost, je rozlišení textury, do které je mapa generována.

Rasterizační efekt Ambient occlusion je implementován pomocí techniky *Screen-Space Ambient Occlusion*. Tato technika opět využívá hloubkovou mapu, tentokrát jde ale o výstup pořízený z pohledu hlavní kamery. Pro každý viditelný bod scény je následně detekováno jeho zastínění pomocí simulace vrhání paprsků v hloubkové mapě. Paprsky jsou, podobně jako u verze implementované pomocí sledování paprsku, generovány v hemisféře orientované podle normály v daném bodě. Pro každý paprsek ve světových koordinátech je provedena projekce do hloubkové mapy. Pokud je koncový bod paprsku hlouběji než odpovídající místo v hloubkové mapě – tedy složka z projektovaného paprsku je větší než hodnota v hloubkové mapě – považuje jej algoritmus za zastíněný. Po akumulaci všech vyslaných paprsků, jejichž počet určuje výslednou kvalitu, je opět vypočítáno průměrné zastínění, které je považováno za výslednou hodnotu *Ambient Occlusion*.

Zajímavým efektem, který nemá v případě rasterizace přímou alternativu, jsou spekulární odrazy. Pro rovné plochy – s nulovou změnou normálového vektoru, například rovinné zrcadlo – je možné použít algoritmus *Screen-Space Reflections* [19]. Tento algoritmus není v aplikaci implementován, ale jeho základním principem je opět simulace sledování paprsku v hloubkové mapě vykreslené scény. Po výpočtu odraženého vektoru se algoritmus snaží nalézt průnik tohoto paprsku se scénou, např. pomocí metody ray-marching a opakované detekce průniku s hloubkovou mapou. Po nalezení tohoto místa je získána barva pro daný fragment a použita jako barva reflekce.

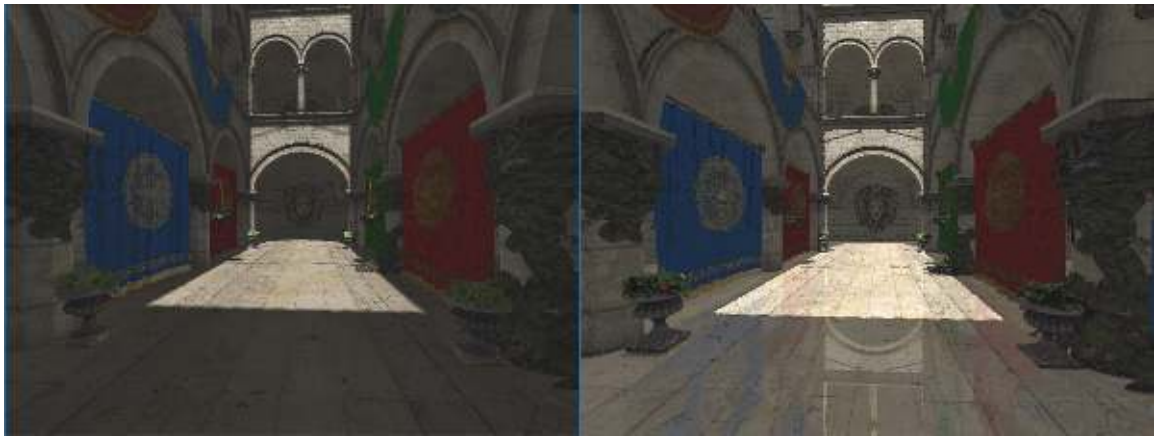
Posledním z implementovaných efektů jsou měkké stíny, implementované pomocí metody *Percentage Closer Filtering* [4]. Tato technika používá výstup již popsané metody pro generování tvrdých stínů a prování její několikanásobné vzorkování. Výsledek je lepší než jednoduché rozmazání, protože jsou hloubkové hodnoty porovnávány před samotnou filtrací. Hlavním parametrem této techniky je velikost *PCF* jádra, které udává kolik vzorků je testováno.

Příklady vygenerovaných rasterizačních efektů lze vidět na obr. 5.10. U tvrdých stínů lze vidět pixelizace, která je nejvíce znatelná pro vzdálené objekty. Měkké stíny jsou rozmazány i v případech, kdy jsou zdroje okluze blízko. Efekt ambient occlusion je nejvíce viditelný na okrajích geometrie, ale není generován u všech záhybů, jako v případě sledování paprsku.

Obrázek 5.10: Ukázka implementovaných rasterizačních efektů – tvrdé stíny (**vlevo**), měkké stíny (**uprostřed**) a ambient occlusion (**vpravo**).

Kromě rasterizačních efektů je součástí výsledné aplikace také jednoduché *Physically Based Rendering* (*PBR*) stínování, které bere v potaz materiálové vlastnosti, jako kovovost a hrubost. Pro implementaci byl použit microfacet model *GGX* [22] a již existující implementace v jazyce *GLSL*[4]. Kvůli co možná nejvyšší podobnosti výstupů generovaných módy *Ray Tracing* a *Rasterization* byla veškerá shader logika zabývající-se výpočtem stínování a osvětlení sdružena do jednoduché knihovny, která je společná pro všechny módy. Příklad výstupů lze vidět na obr. 5.11 .

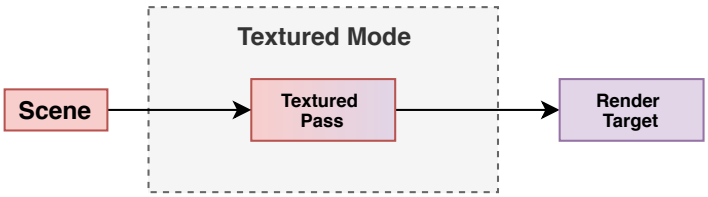
[4]Zdroj https://learnopengl.com/PBR/Theory .



Obrázek 5.11: Ukázka syntetizovaných obrazů, které jsou výstupem *Resolve* průchodů. Na obrázku lze vidět výstup *Rasterization* (**vlevo**) a *Ray Tracing* (**vpravo**) módů.

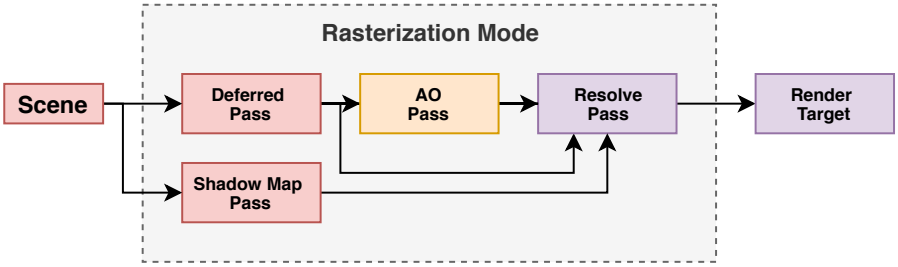
5.4 Hybridní vykreslování

Vykreslování je v enginu rozděleno do tzv. *vykreslovacích vrstev*, které jsou použity jako organizační jednotky pro jednotlivé vykreslovací operace. Každá vrstva obsahuje kromě samotného vykreslovacího průchodu (*rendering pass*) také přípravu zdrojů, jakými jsou např. akcelerační struktury nebo textury. Výsledná testovací aplikace, implementovaná za použití základního enginu, obsahuje tři vykreslovací módy: *Textured*, *Rasterization* a *Ray Tracing*. První z nich – *Textured* mód – je zde z důvodu testování a jako vizuální reference. Jeho práce se sestává pouze z jedné vykreslovací vrstvy, jejíž použití lze vidět na obr. 5.12. Vstupní scéna je zpracována výše popsaným způsobem a vykreslena pomocí jednoduchých shader programů, jejichž výstupem je model s nanesenými albedo texturami.



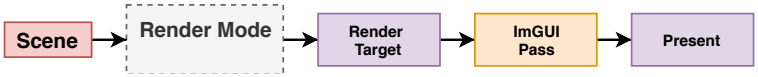
Obrázek 5.12: Diagram funkce vykrslovacího módu *Textured*, který se sestává pouze z jedné vykreslovací vrstvy. Vstupní scéna je vykreslena do cílového bufferu, který je následně zobrazen v hlavním aplikačním okně.

Druhý, již komplexnější vykreslovací mód nazvaný *Rasterization*, využívá čistě rasterizační techniky. Diagram obsahující diagram jeho komponent lze vidět na obr. 5.13. Prvním krokem je generování *Deferred G-Bufferu* (sekce 2.4), který je dále používán pro syntetizaci výsledného obrazu. Zároveň jsou naplněny stínové mapy využívané metodou *Shadow Mapping*, jejichž tvorba opět vyžaduje vykreslení scény. Dalším krokem je vykreslení efektu *Ambient Occlusion*, které používá výstupní *G-Buffer*. V závěru jsou všechny vygenerované výstupy kombinovány do výstupního obrazu a předány pro prezentaci v aplikačním okně.



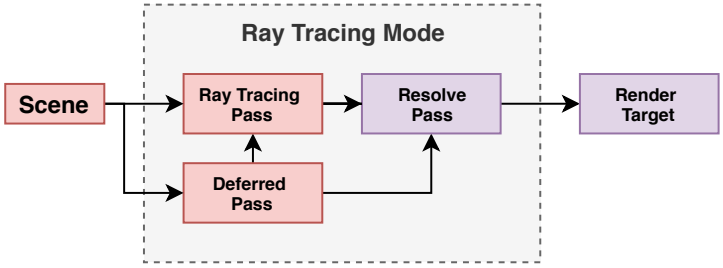
Obrázek 5.13: Diagram zobrazující postup použitý pro vykreslení konečného obrazu za pomocí čisté rasterizace. Scéna je před-vykreslena v *Deferred* průchodu, čímž je snížena cena finálního stínování. Výsledný syntetizovaný obraz je poskládán z vygenerovaných vstupů a předán pro zobrazení.

Průběh prezentace výsledného syntetizovaného obrazu lze vidět na obr. 5.14. Výstup ze zvoleného vykreslovacího módu je navíc obohacen o grafické uživatelské rozhraní, které je vykresleno za využití knihovny *ImGUI*. Následně je buffer obsahující finální obraz zařazen do tzv. *Swap Chain*, který je ve stejném pořadí prezentuje v aplikačním okně [5].



Obrázek 5.14: Diagram prezentace výsledného obrazu do aplikačního okna. Kromě samotné syntetizaci výstupního obrazu je zde také průchod vykreslující grafické uživatelské rozhraní.

Posledním módem je *Ray Tracing* vykreslovací mód, který využívá výše popsané metody hybridního vykreslování. Diagram obsahující postup syntetizace výsledného obrazu lze vidět na obr. 5.15. Prvním krokem je opět příprava *G-Bufferu*, čehož je docíleno vykreslením vstupní scény za pomocí rasterizačního průchodu. Po dokončení tohoto kroku je spuštěno samotné sledování paprsků, které využívá vygenerovaný *G-Buffer*, díky kterému je možné přeskočit vysílání primárních paprsků[6]. Vygenerované výstupy jsou nakonec opět kombinovány a předány pro prezentaci.

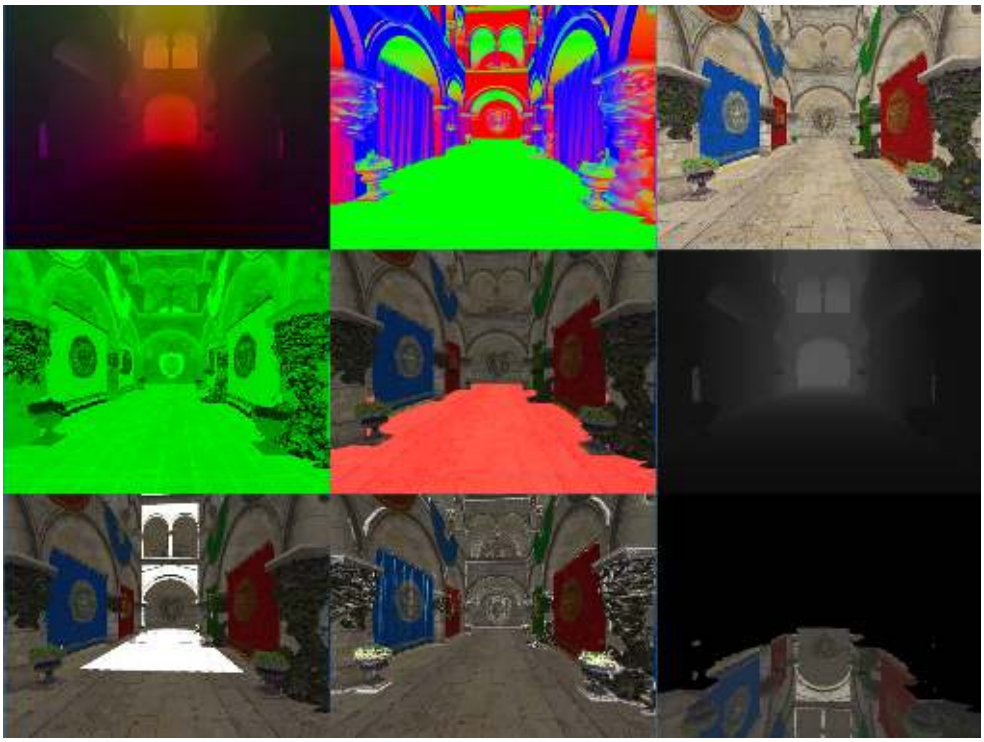


Obrázek 5.15: Vykreslovací mód *Ray Tracing* používá popsanou metodu hybridního vykreslování, která kombinuje výstupy rasterizace a sledování paprsku. Výstupní informace vygenerované z aktuální scény jsou předány *Resolve* průchodu, který z nich poskládá výsledný obraz a předá ho oknu pro zobrazení.

[5]Swap chain po vypnutí limitu vykreslovaných snímků za sekundu snímky automaticky přeskakuje.
[6]Implementace obsahuje i možnost využití „čistého“ sledování paprsků, které *G-Buffer* nevyužívá.

Generování *G-Bufferu* probíhá rasterizací aktuální scény z pohledy primární kamery. Jednotlivé složky jsou ukládány do bufferů, jejichž obsah lze vidět na obr. 5.16. Rozložení bufferů je následující [7]:

- Pozice – 4 hodnoty *float* pro pozici ve světě, poslední složka je použita pro detekci zápisu do *G-Bufferu*.
- Normála – 4 hodnoty *float* pro normálu v souřadném systému světa, poslední složka nevyužita.
- Albedo – 4 hodnoty *float* pro *RGBA* hodnoty z albedo textury pro daný materiál.
- Materiál – 4 hodnoty *float*, první dvě jsou použity pro *PBR* kovovost a hrubost. Další dvě hodnoty obsahují materiálové flagy, například zda materiál generuje spekulární odrazy.
- Hloubka – 1 hodnota *float*, hloubka uložená bez linearizace.
- Stíny – 1 hodnota *float*, generovaná skrz sledování paprsku.
- Ambient Occlusion – 1 hodnota *float*, generovaná skrz sledování paprsku.
- Odrazy – 4 hodnoty *byte*, generované skrz sledování paprsku.



Obrázek 5.16: Ukázka vygenerovaných bufferů při vykreslování v módu *Ray Tracing*. Postupně z levého horního rohu obsahují: pozice, normály, albedo, materiál, reflektivita, hloubka. Následují výstupy generované sledováním paprsku: stíny, ambient occlusion a reflekce.

[7] Z pohledu výkonnosti toto rozložení není zdaleka optimální. Jeho výhodou je možnost použití stejných bufferů ve všech průchodech.

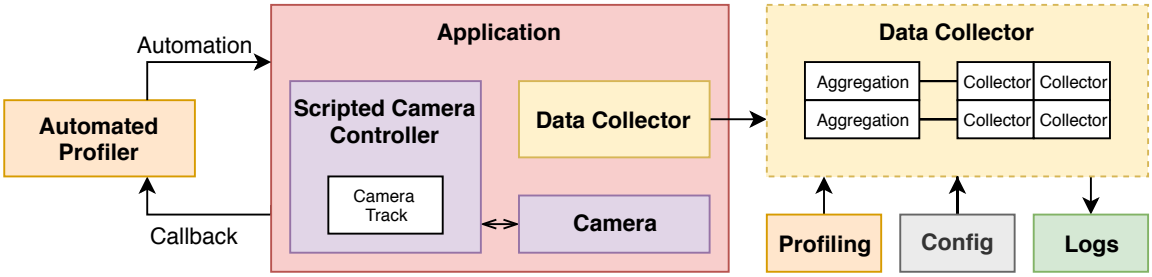
Hodnoty jsou před uložením do *G-Bufferu* kódovány, kdy například pozice ve světovém souřadném systému jsou uloženy relativně k pozici kamery, čímž je zvýšena jejich přesnost. Pro kódování a dekódování je vytvořena jednoduchá knihovna v jazyce *HLSL*, jejíž cílem je minimalizace rozdílu mezi uloženou a načtenou hodnotou.

5.5 Experimenty a testovací scénáře

Mezi testované parametry systému patří primárně *GigaRays/s* – miliardy generovaných paprsků za sekundu – a čas nutný pro stavbu akceleračích struktur. Kromě těchto základních parametrů lze také testovat cenu jednotlivých efektů a případně srovnání s jejich rasterizčními ekvivalenty. Dalším cílem, který si tato práce klade je zhodnocení náročnosti implementace této technologie do již existujícího čistě raterizačního enginu. Tato sekce se zabývá implementací jednotlivých testů a experimentů, jejichž cílem je získat data nutná pro následující kapitolu – vyhodnocení výsledků.

5.5.1 Automatické testy a logování

Z důvodu komplexnosti celého systému – množství testovatelných parametrů a nastavení ovlivňující výkonnost – jsou součástí enginu také možnosti automatického testování a generování logů. Diagram základních bloků, se kterými tento systém pracuje lze vidět na obr. 5.17. Každý z dále popisovaných testovacích scénářů implementuje rozhraní **Automated-Profiler**, které má přístup ke všem proměnným běžící aplikace. Při spuštění aplikace v módu automatického testování, je vytvořena instance odpovídající cílovému scénáři. Aplikace dále s touto instancí komunikuje pomocí zpětných volání, které jsou rozmístěny v hlavní aplikační smyčce.



Obrázek 5.17: Diagram systému automatického testování a generování logů. Každý z automatických testů je implementací rozhraní **Automated Profiler**, která automaticky ovládá aplikaci. Systém automaticky sbírá aktuální informace z aplikace a případně generuje aplikační logy.

Další důležitou funkcí, použitou v testovacích scénářích, je automatizace pohybu kamery. K tomuto účelu je v aplikaci možné použít skriptovaný ovladač kamery. Tento ovladač pracuje na základě záznamu tzv. *dráhy*, která udává klíčové snímky pozice a rotace kamery. Každý snímek obsahuje mimo jiné také čas jeho trvání čímž je umožněna plynulá animace, která využívá rotační kvaterniony a interpolaci typu *SLERP* z knihovny *DXTK*. Součástí je také možnost serializace a uložení / načtení ze souboru. Automatizační dráhy lze vytvářet přímo v aplikaci[8], nebo editací souborů s příponou „.trk“.

[8] Video obsahující příklad průchodu lze najít na přiložené SD kartě pod jménem „SponzaAutoPresetRayTracing.mp4“.

Poslední částí tohoto systému je tzv. **Data Collector**, jehož cílem je agregace datových toků a automatický sběr logů. Jde o relativně jednoduchou databázi hodnot, která umožňuje segmentaci datových toků do větších celků – *agregací*. Každou agregaci si lze představit jako tabulku v databázi, která obsahuje jeden nebo více řádků. Jednotlivé řádky obsahují řady hodnot generované jejich odpovídajícími kolektory. Jednotlivé kolektory lze nastavit pro sběr hodnot z různých částí aplikace – například z profilovacího systému, konfigurace aplikace nebo z uživatelem definovaných funkcí. Součástí generovaných logů je také aktuální nastavení aplikace, a argumenty předané z příkazového řádku, čímž je umožněno opakování experimentu se stejnými vstupními podmínkami. Příklad zkráceného výstupního logu lze vidět v příloze B.

5.5.2 Testovací scénáře

Již z prvotního testování bylo patrné, že pro přesné změření výkonnosti počtu paprsků vržených za sekundu bude nutný speciální přístup. Výkonnost systému je velmi závislá na poměru primárních paprsků, které zasáhnou geometrii – čímž dochází k vyvolání *Closest Hit* shaderu. První kategorie testů byla tedy navržena pro maximální přesnost hodnoty vyslaných paprsků za sekundu. Z tohoto důvodu byly vybrány scény a pozice kamery, pro které byl vypočítán počet *hit* pixelů a *miss* pixelů – ve kterých paprsek zasáhl, respektive minul, geometrii. Díky těmto koeficientům je následně možné vypočítat relativně přesné hodnoty paprsků vyslaných za sekundu, které jsou popsány v následující kapitole. Tyto testy nebylo možné jednoduchým způsobem automatizovat a proto byly provedeny manuálně.

Automatické profilování je rozděleno do čtyř kategorií: *Duplication*, *Build*, *Camera* a *Triangle*. Cílem *duplikačních* testů je získat časy nutné pro stavbu akceleračních struktur, pomocí několikanásobné duplikace celé scény. Jednotlivé duplikáty jsou vždy vloženy do vlastních spodních akceleračních struktur, čímž je možné také testovat závislost počtu akceleračnich struktur na celkovém čase nutném pro jejich stavbu. Build testy naopak testují jednotlivé modely, kde každá mesh z dané scény je vložena do vlastní akcelerační struktury. Výsledkem těchto testů je závislost doby stavby akceleračních struktur na počtu trojúhelníků a jejich distribuci.

Testy v kategorii *Camera* používají výše popsaný způsob automatizace kamery, který umožňuje průlet vybrané scény. V této kategorii jsou všechny testy srovnávající výkonnost jednotlivých módů a implementovaných efektů. Z důvodu lepší segmentace testů a srovnání kvality jsou v aplikaci k dispozici také profily kvality, které obsahují nastavení jednotlivých efektů. Mezi tyto profily patří například:

- **Bez efektů**: Všechny grafické efekty jsou vypnuty, pouze stínování.
- **Tvrdé stíny**: Pouze tvrdé stíny odpovídající kvalitě *Low* a stínování.
- **Ambient Occlusion**: Pouze Ambient Occlusion odpovídající kvalitě *Low* a stínování.
- **Odrazy**: Pouze odrazy odpovídající kvalitě *High* a stínování.
- **Měkké stíny**: Pouze měkké stíny odpovídající kvalitě *Medium* a stínování.
- **Low**: Tvrdé stíny s rozlišením stínových map 2048 pixelů, 4 vzorky Ambient Occlusion a stínování.
- **Medium**: Měkké stíny s rozlišení stínových map 4096 pixelů a 4 stínové vzorky, 4 vzorky Ambient Occlusion a stínování.

- **Medium**: Měkké stíny s rozlišení stínových map 4096 pixelů a 4 stínové vzorky, 4 vzorky Ambient Occlusion, odrazy a stínování.
- **Ultra**: Měkké stíny s rozlišení stínových map 8192 pixelů a 8 stínových vzorků, 16 vzorků Ambient Occlusion, odrazy a stínování.

Poslední kategorií je *Triangle*, která obsahuje testy jejichž cílem je získat závislost výkonnosti vyslaných paprsků na počtu trojúhelníků ve scéně. Tyto testy opět používají automatické ovládání kamery, které umožňuje nahlížet na scénu z různých úhlů. Navíc je zde přidána možnost duplikace scény, která podobně jako při testování stavby akceleračních struktur vytváří duplikace celé scény, čímž je získáno vysoké množství trojúhelníků.

5.5.3 Integrace sledování paprsku

Kromě implementace vlastního vykreslovacího enginu *Quark*, jsou součástí práce také zkušenosti s integrací technologie akcelerovaného sledování paprsků do již existujícího *in-house* enginu v herní vývojářské společnosti *Hangar13*[9]. Z důvodu podepsané dohodě o nezveřejnění (*NDA*) nemohou být některé implementační detaily zveřejněny v této práci, čímž ale nejsou omezeny zkušenosti se samotnou integrací.

Proces samotné integrace probíhal v několika fázích. První z nich byla zpracování vstupních dat, mezi které patří geometrie, textury a materiály. Jednotlivé buffery, obsahující vertex atributy jako například pozice, normála nebo texturovací koordináty, bylo nutné transformovat do formátu přístupného z shaderů. Textury a jejich mip-mapy byly použity ve stejné formě, jako v originálním enginu, čímž se snížila režie načítání scén. Největším problémem byly materiály, jejichž plná implementace by vyžadovala použití již existujících shaderů, které nejsou se systémem akcelerovaného sledování paprsku kompatibilní. Většina z testovaných efektů – kromě odrazů – plnou implementaci materiálů nepotřebuje a proto bylo převedeno pouze pouze pár základních materiálů pro vizuální kontrolu korektnosti. Výsledkem této fáze bylo zobrazení jednoduchých modelů za použití čistého sledování paprsků.

Druhá fáze, jejíž cílem bylo zobrazení komplexnějších scén, byla implementována za pomocí výše popsaných tříd pro reprezentaci scény. Po načtení modelů a jejich materiálů je vytvořen pomocný popis scény v interním formátu, ze kterého jsou sestaveny akcelerační struktury. Následně je možné scénu zobrazit za použití výše popsaného postupu. Akcelerační struktury jsou generovány postupně se změnami ve scéně. Při změnách pozice je vygenerována pouze vrchní vrstva akcelerační struktury. V případě přidání nových modelů je nutné sestavit odpovídající akcelerační struktury spodní vrstvy. Nové akcelerační struktury jsou potom přidány do cache, čímž je umožněno jejich znovupoužití.

Poslední fází byla experimentace s implementovanými efekty a stavbou akceleračních struktur. Díky integraci do již existujícího enginu bylo možné testovat velké množství scén a modelů, které odpovídají cílovému použití technologie v herním průmyslu. Výsledky těchto testů lze najít v následující kapitole.

[9] Viz https://hangar13games.com/ .

Kapitola 6

Vyhodnocení výsledků

Tato kapitola obsahuje zhodnocení dosažených výsledků, získaných provedením výše popsaných testovacích scénářů a experimentů. Výsledky ukazují limity technologie akcelerovaného sledování paprsků na výkonných grafických akcelerátorech *RTX 2080 Ti* a *GTX 970*. Kromě základních parametrů jsou také testovány implementované grafické efekty a závislost výkonnosti systému na jejich nastavení. Efekty používající sledování paprsku jsou také srovnány s jejich široce využívanými rasterizačními alternativami.

6.1 Testovací konfigurace

Při testování byla použita optimalizovaná spustitelná verze výsledné aplikace, přeložená za použití nejvyšší úrovně optimalizace. Testování bylo provedeno na dvou hardwarových sestavách, jejichž specifikace jsou popsány v tabulce 6.1.

Aby bylo možné uvést výsledky do souvislosti pro čtenáře této práce, byly v případě mnoha následujících testů použity dobře známé scény, které jsou často používané pro testování grafických efektů. Scény jsou volně k dispozici a lze je nalézt ve veřejném repositáři knihovny *glTF*[1]. Bližší informace k těmto modelům lze také nalézt v tabulce 6.2.

Tabulka 6.1: Specifikace konfigurace testovaných systémů. **PC1** využívá přímou hardwarovou akceleraci, zatímco **PC2** tuto funkci emuluje pomocí *Compute* shaderů.

ID	CPU	GPU	Compiler
PC1	Intel Xeon W 2135	Nvidia RTX 2080 Ti	MSVC++ 14.16
PC2	Intel i5 4670k	Nvidia GTX 970	MSVC++ 14.16

Tabulka 6.2: Informace o primárních testovacích scénách, používaných v testovacích scénářích. Mezi textury jsou započítány albedo, materiálové a normálové mapy.

Scene	Triangles [n]	Meshes [n]	Textures [n]
Box	12	1	2
Suzanne	3936	1	2
Sponza	262267	103	69

[1]Viz repozitář https://github.com/KhronosGroup/glTF-Sample-Models .

6.2 Výkonnost Ray Tracing jader

Tato sekce obsahuje výsledky testování hardwarové akcelerace sledování paprsku z pohledu počtu zpracovaných paprsků. Hlavní použitou metrikou je počet vyslaných paprsků za sekundu, pro kterou je použita jednotka *GigaRays/s* (zkratka GR/s). Pro výpočet této hodnoty je použit následující vzorec:

$$rps = fps \cdot width \cdot height \cdot rpp \tag{6.1}$$

Ve kterém mají jednotlivé proměnné následující význam:

- fps : Průměrný počet snímků za sekundu prezentovaných na obrazovku. Ve všech testech je vypnuta vertikální synchronizace s monitorem.
- *width* a *height* : Reprezentují rozměry aktuálně vykreslovaného rastru. Jejich násobením lze získat počet invokací prvotního *Ray Generation* shaderu.
- *rpp* : Průměrný počet paprsků vyslaných na jeden pixel.
- *rps* : Výsledná hodnota počtu paprsků za sekundu.

6.2.1 Zátěžové testy

Prvním z testovacích scénářů jsou zátěžové testy, jejichž cílem je zjistit výkonnost *Ray Tracing* jader a určit jejich limitní počet vyslaných paprsků. Z důvodu radikálně rozdílných výkonnostních parametrů, které závisí na pozici kamery ve scéně a jejím natočení, byly zvoleny pevně dané konfigurace kamery pro každou z primárních scén. Pohledy z těchto kamer lze vidět na obr. 6.1.

Obrázek 6.1: Pozice kamery použité pro přesné měření výkonnosti *Ray Tracing* jader. Použité scény jsou postupně: *Cube* (**vlevo**), *Suzanne* (**uprostřed**) a *Sponza* (**vpravo**).

Pro každou kombinaci pozice a testované rozlišení výstupního rastru byly přesně spočítány poměry paprsků, které scénu zasáhly a které scénu minuly – zjištěné hodnoty lze najít v tabulce 6.3.

Testování je dále rozděleno do dvou kategorií, první z nich obsahuje testy nazvané *Box*, *Suzanne* a *Sponza*. U těchto scénářů bylo cílem zjistit výkonnost sledování paprsků na jednoduchých scénách, bez složitějšího stínování. Jednotlivé scény jsou vykresleny s výše zobrazenými konfiguracemi kamery a za použití čistého sledování paprsků bez využití ostatních vykreslovacích průchodů. Pro každý pixel je po spuštění vyslán jeden primární paprsek. Pokud tento paprsek zasáhne geometrii, jsou dále vyslány následující paprsky: jeden *stínový paprsek* pro určení zastínění a 64 *stínových paprsků* pro Ambient Occlusion.

Tabulka 6.3: Tabulka obsahující informace o konfiguracích scény pro přesné měření výkonnosti *Ray Tracing* jader. Scény jsou vykreslovány ze stejných pozic s různým rozlišením rastru. Sloupce *Miss* a *Hit* obsahují procento pixelů, jejichž paprsky minuly, respektive zasáhly.

Scene	Resolution	Miss [%]	Hit [%]
Box	2560x1387	62	38
Suzanne	2560x1387	82	18
Sponza	2560x1387	0	100

V případě, že se primární paprsek vyhne veškeré geometrii scény, nejsou vysílány žádné další paprsky. Pro výpočet výsledné barvy je použit jednoduchý *Lambertovský* model osvětlení, bez použití *GGX* stínování.

Druhá kategorie obsahuje testy nazvané *Miss* a *Stress*. Cílem těchto testů je maximální vytížení *Ray Tracing* jader. Oba testovací scénáře jsou provedeny za použití scény *Sponza*, spolu s výše zobrazenou konfigurací kamery. V obou případech je pro každý pixel na rastru kamery vysláno 64 primárních paprsků. V prvním případě (*Miss*) je jejich maximální délka nastavena tak, aby nezasáhly scénu. Paprsky v druhém scénáři (*Stress*) jsou naopak nastaveny tak, aby všechny scénu zasáhly.

Tabulka 6.4: Tabulka obsahující výsledky pořízené za použití statické pozice kamery. Veškeré testy byly provedeny v rozlišení 1440p. Hodnota **Rays/pixel** je korigovaná pro danou pozici kamery a procento paprsků, které skutečně zasáhly geometrii. Sloupce FT_{Rt} obshují počet milisekund pro vykreslení jednoho snímku pomocí sledování paprsku, FT_{Ra} potom počet milisekund pro vykreslení pomocí referenční rasterizační implementace (*Textured* mód).

		PC1			**PC2**		
	Rays/pixel	FT_{Rt}	GR/s	FT_{Ra}	FT_{Rt}	GR/s	FT_{Ra}
Box	25.7	7.04	12.82	0.47	434.78	0.55	0.66
Suzanne	12.7	8.13	5.50	0.53	401.13	0.59	0.85
Sponza	66.0	32.26	7.09	1.86	4937.72	0.04	3.44
Miss	64.0	17.86	12.52	1.86	263.16	0.85	3.44
Stress	64.0	35.71	6.25	1.86	3372.54	0.08	3.44

Výsledky testování pro obě testovací sestavy lze nalézt v tabulce 6.4. Kromě počtu snímků za sekundu je možné v tabulce také nalézt přesný počet vyslaných paprsků na jeden pixel, k jejichž výpočtu byly použity výše popsané pozice kamer a jejich předpočítané parametry. Dále tabulka obsahuje průměrný počet GR/s a pro srovnání také počet snímků za sekundu při vykreslení pomocí jednoduchého *Textured* průchodu.

Z výsledků pro první sestavu – **PC1** využívající akcelerátor *RTX 2080 Ti* – lze vidět velmi vysokou výkonnost odpovídající oficiálním zdrojům[2], které uvádějí hodnotu 10 GR/s. Výsledky druhého systému, který nedisponuje hardwarovou akcelerací sledování paprsku, ukazují relativně nízkou výkonnost, která je ale při daném počtu paprsků stále relativně přívětivá. Zajímavým výsledkem je propad výkonnosti u jednoduššího modelu *Suzanne* oproti modelu *Sponza*, který koreluje s nižším procentem paprsků zasahujících scénu. Tento

[2] Viz https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/ .

propad může být zapříčiněn nižší obsazeností *Ray Tracing* jader. Z výsledků lze také obecně říci, že paprsky které minou geometrii jsou ve výsledku výpočetně levnější, než v případě zásahu.

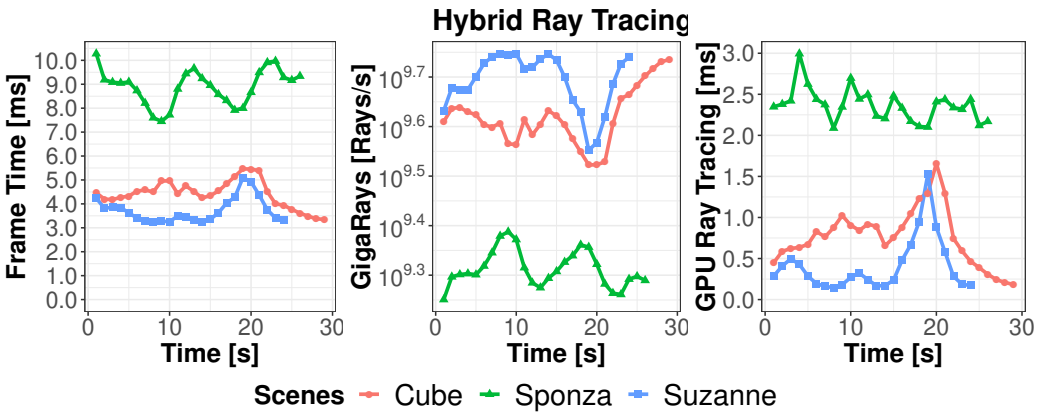
Všechny z následujících testovacích scénářů byly provedeny na obou testovacích sestavách (**PC1**, **PC2**), ale naměřené hodnoty vykazovaly v obou případech stejnou tendenci a jejich výkonnostní poměr byl vždy podobný těm v tabulce 6.4. Z tohoto důvodu nejsou výsledky ze sestavy **PC1** v následujících testovacích scénářích již prezentovány a jsou použity pouze výsledky z výkonnější sestavy **PC2**[3].

6.2.2 Testy hybridního vykreslování

Tato sekce popisuje výsledky testů, jejichž cílem bylo otestovat vlastnosti vykreslovacího systému v realistických podmínkách. Základem je opět měření počtu GR/s a počet milisekund na jeden snímek (FT). Při testování byly použity již dříve definované scény – *Cube*, *Suzanne* a *Sponza*.

První sada testů využívá automatizaci pohybu kamery k simulaci průchodu scénou. Pro každou scénu je použit předem připravený průchod[4]. Pro každou scénu je stejný průchod proveden v různých vykreslovacích módech – *Rasterization*, *Ray Tracing* a *Hybrid Ray Tracing* – a za použití dvou rozlišení výstupního rastru – 1024×768 a 2560×1440. Všechny testy byly provedeny za použití presetu kvality **Low** (viz sekce 5.5.2).

Grafy hodnot naměřených za použití hybridního sledování paprsků lze vidět na obr. 6.2. Jednotlivé grafy obsahují informace z jednoho průchodu scénou pro výše popsané scény. Z výsledků je patrné, že konfigurace kamery může v případě vykreslovaných scén výrazně ovlivnit výkonnost celého systému. Například nárůst doby výpočtu snímku, který lze vidět u scény *Suzanne* a *Cube*, koresponduje s přiblížením k modelu[5] kdy ho většina vyslaných paprsků zasáhne.



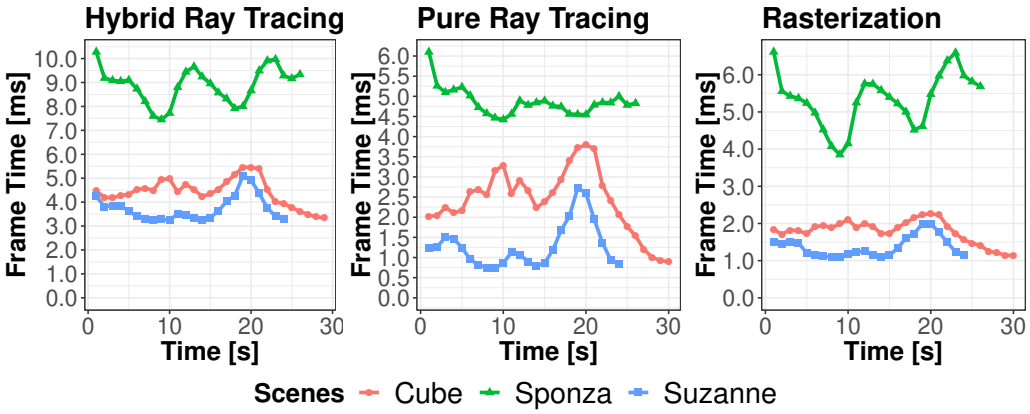
Obrázek 6.2: Grafy zobrazující výkonnost hybridního sledování paprsku v průběhu testování. Horizontální osa obsahuje aktuální čas automatizačního klipu kamery. Jednotlivé grafy zobrazují čas výpočtu na snímek (**vlevo**), vyslané paprsky za sekundu (**uprostřed**) a čas výpočtu sledování paprsku na GPU (**vlevo**).

[3] Záznamy ze všech provedených testů (PC1 i PC2) lze nalézt na přiloženém médiu.
[4] Ukázku těchto průchodů lze nalézt na přiloženém médiu ve formě videa.
[5] Průchod touto scénou lze vidět na videu „SuzanneAutoRayTracing.mp4“ na přiloženém médiu.

Hodnoty zobrazené na grafu vyslaných paprsků nejsou upraveny podle počtu *hit* a *miss* paprsků. Na grafu lze pozorovat, že i v případě složitější scény *Sponza* je výkonnost stále kolem 2 GR/s. Tento výsledek je obzvláště dobrý, protože při automatickém průchodu scény je téměř po celou dobu znatelná převaha paprsků zasahujících geometrii.

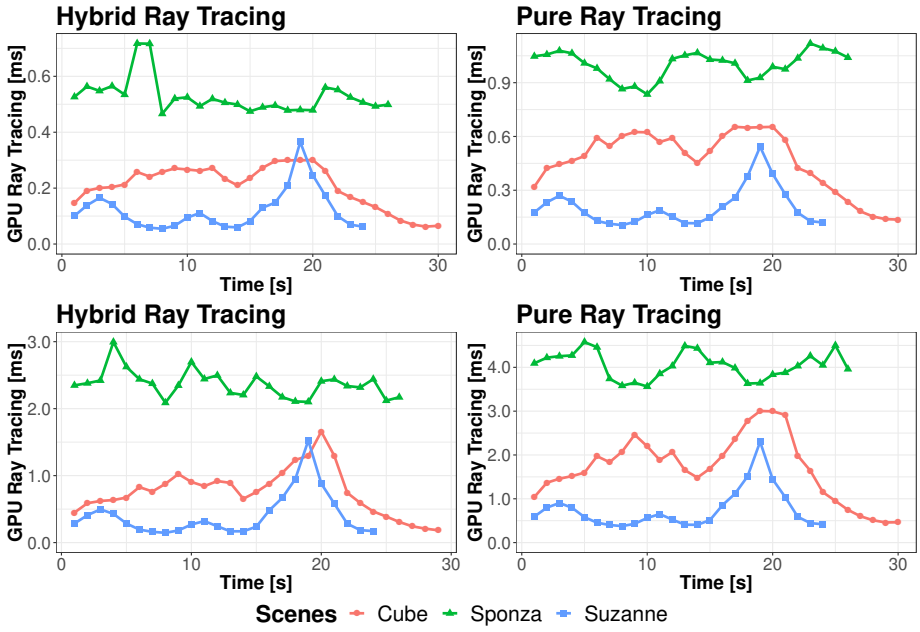
Na grafu výpočetního času lze opět vidět propady ve výkonnosti, které byly zmíněny již výše. Do hodnot vykreslených v tomto grafu je započítán pouze čistý čas, který zabral výpočet sledování paprsku na grafickém akcelerátoru – bez generování *G-Bufferu* a finálního *Resolve* průchodu.

Díky provedení stejných testů ve všech vykreslovacích módech – *hybridní sledování paprsku*, *čisté sledování paprsku* a *rasterizace* – je možné také provést srovnání jejich výkonnosti. Výsledky tohoto srovnání lze vidět na obr. 6.3. Na první pohled je možné z grafů usoudit, že *čisté sledování paprsků* má ze všech metod nejvyšší výkonnost. Ve skutečnosti tomu ale tak je pouze v případě některých konfigurací kamery, kdy většina paprsků nezasáhne vykreslovanou geometrii. Tento efekt lze vidět v poklesu výkonnosti u scén *Cube* a *Suzanne*, které korelují s přiblížením kamery k modelu. V případě scény *Sponza* jsou již výsledky *rasterizace* a *čistého sledování paprsků* srovnatelné.



Obrázek 6.3: Grafy obsahující srovnání výkonnosti mezi metodami sledování paprsku a čistě rasterizačním přístupem. Horizontální osa obsahuje čas automatizačního klipu kamery, vertikální potom celkový čas snímku v milisekundách.

Při porovnání *čistého* a *hybridního* přístupu je opět *čisté* sledování paprsků znatelně rychlejší. Rozdíl vzniká díky rasterizaci *G-Bufferu*, které musí být dokončeno před hybridní sledováním paprsku. Na grafu lze vidět také stejné poklesy a vrcholy ve výkonnosti jako v případě rasterizace. Při reálném využití hybridního sledování paprsku závislost na *G-Bufferu* již nebude problém, protože jeho generování je vyžadováno i pro jiné grafické efekty a proto je cena jeho výpočtu amortizována.

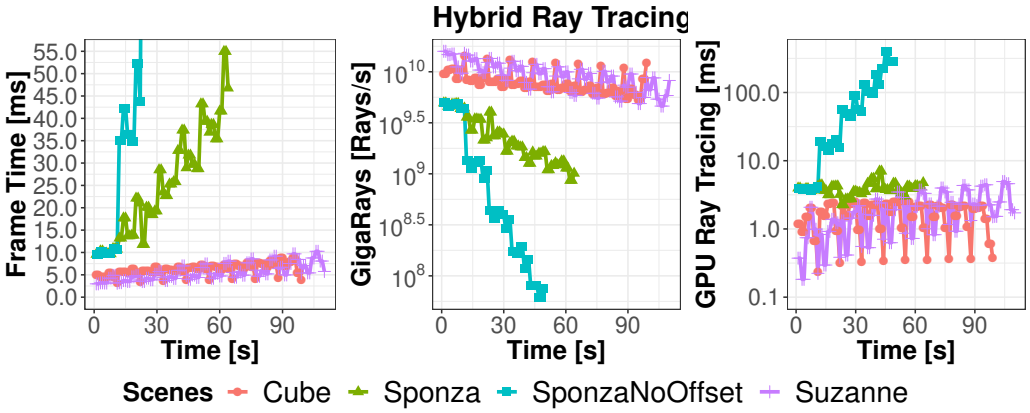


Obrázek 6.4: Grafy zobrazující dobu výpočtu sledování paprsku na GPU. Na grafech lze vidět výsledky pro hybridní přístup (**vlevo**), čisté sledování paprsku (**vpravo**) a dvě různá rozlišení rastru 1024×768 (**nahoře**) a 2560×1440 (**dole**).

Pokud by byl srovnán pouze čas nutný pro výpočet samotného sledování paprsku, potom lze vidět znatelně nižší časovou náročnost celé operace v případě *hybridního* přístupu. Srovnání časů nutných pro výpočet čistě sledování paprsků na GPU lze najít na obr. 6.4.

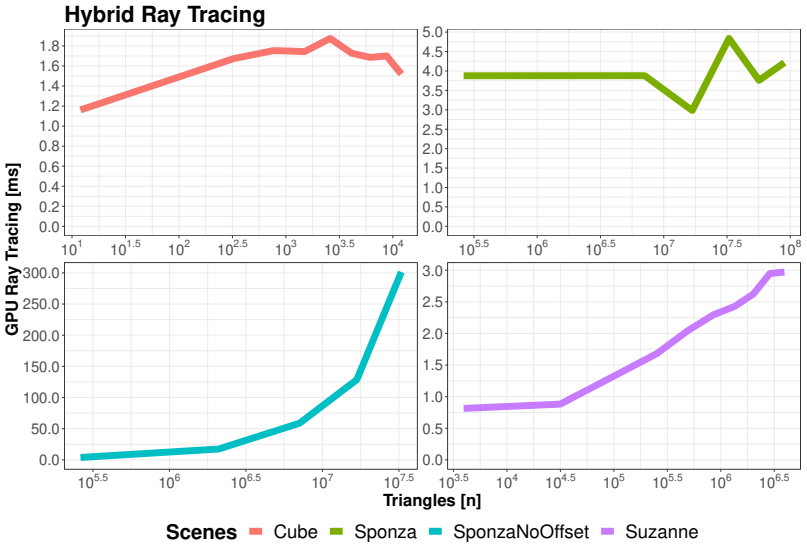
Následující sada testů také využívá automatizaci pohybu kamery k simulaci průchodu scénou. V tomto případě je ale po dokončení jednoho průchodu provedena duplikace celé scény v tří-dimenzionální mřížce – první úroveň obsahuje celou scénou jednou, druhá úroveň již devětkrát, následující 27-krát atp. Po dokončení duplikace následuje další průchod. Rozestup jednotlivých modelů v mřížce je pevně nastaven na začátku testu. Testovány jsou opět tři výše popsané primární scény – *Cube*, *Suzanne* a *Sponza*. Poslední z nich (*Sponza*) je testována dvakrát, jednou s původním rozestupem duplikace a podruhé s nulovým rozestupem, kdy se všechny instance scény navzájem překrývají.

Naměřené hodnoty, které lze najít na grafech na obr. 6.5, ukazují závislost výkonnosti sledování paprsku na počtu trojúhelníků ve scéně. Na grafech lze postupně vidět průběh výkonnosti při zvyšující-se úrovni duplikace. Mezi měřené hodnoty opět patří snímky za sekundu, počty vyslaných paprsků a dobu výpočtu na GPU. Jednotlivé průchody scénou lze na grafu rozpoznat periodickým opakováním poklesů ve výkonnosti.



Obrázek 6.5: Graf zobrazující postupnou degradaci výkonu při zvyšování počtu trojúhelníků ve scéně. Grafy postupně zobrazují čas výpočtu na snímek (**vlevo**), vyslané paprsky (**uprostřed**) a dobu výpočtu na GPU (**vlevo**).

Z výše popsaného grafu není příliš patrné, jakým způsobem je výkonnost na počtu trojúhelníků závislá a proto lze na obr. 6.6 vidět také zjednodušenou verzi, využívající stejná data. Hodnoty na vertikální ose grafu byly získány jako průměr časů nutných pro výpočet sledování paprsku za celý průchod. U obou případů využití scény *Sponza* je použita nižší maximální duplikace z důvodu odstranění zařízení při příliš dlouhém výpočtu. Zajímavý výsledek ukazuje graf v levém spodním rohu, kdy po počáteční nulové duplikaci výpočet vyskočí na hodnotu kolem 30 milisekund. V tomto případě byla použita stejná scéna jako pro graf v pravém horním rohu, ale rozestup v duplikační mřížce byl nastaven na nulu. Tento výsledek odpovídá očekávání, protože každý paprsek po průchodu akcelerační strukturou nakonec musí lineárně prohledávat trojúhelníky v listovém uzlu. Při nulovém rozestupu je počet testovaných trojúhelníků radikálně vyšší, protože je není možné vyřadit při průchodu akcelerační strukturou.



Obrázek 6.6: Grafy zobrazující vývoj výkonnosti v závislosti na počtu trojúhelníků ve scéně. Postupně grafy zobrazují jednotlivé scény – *Cube*, *Sponza*, *Suzanne*. Model *Sponza* je použit dvakrát, kdy ve **spodním** případě je nastaven offset duplikátů na nulu.

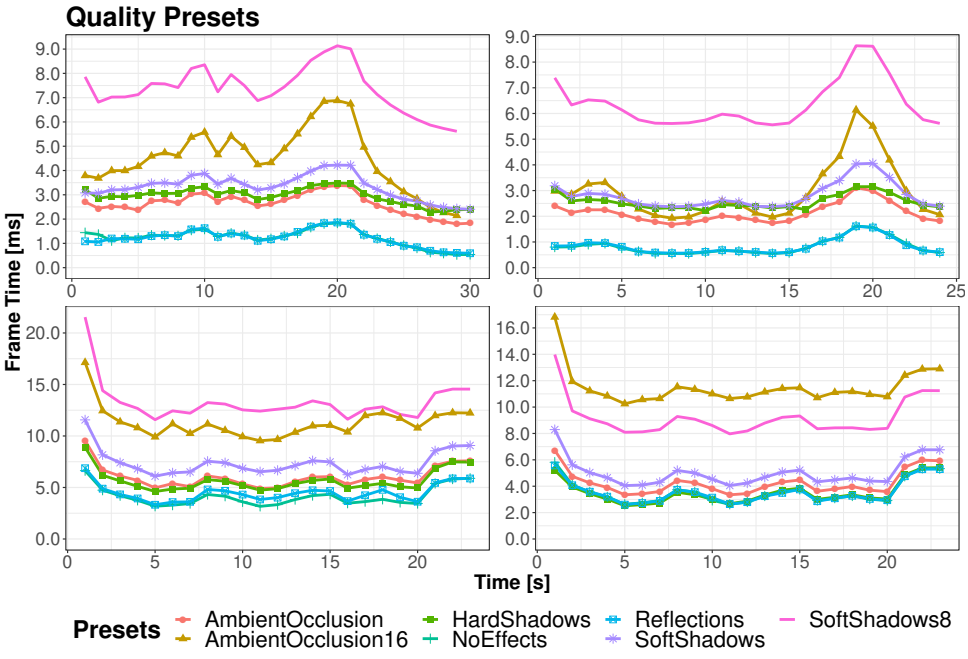
6.2.3 Testy grafických efektů

Poslední z testovacích kategorií zaměřených na výkonnost *Ray Tracing* jader jsou testy grafických efektů, kterou se zabývá tato sekce. Jednotlivé presety, jejichž specifikace lze najít v tabulce 6.5, jsou testovány podobným způsobem jako v předchozích testovacích kategoriích. Po dokončení přípravy scény je spuštěno ovládání kamery, které provede průchod scény. Testování je opět provedeno na výše popsaných scénách – *Cube*, *Suzanne* a *Sponza*.

Tabulka 6.5: Tabulka obsahuje specifikaci presetů kvality, včetně informací o počtu vysílaných paprsků. Při zapnutí odrazů je výsledný počet paprsků zvýšen o jedna.

Name	SM Resolution	PCF	AO Rays	Shadow Rays	Reflections
NoEffects	0	N/A	0	0	No
HardShadows	2048	N/A	0	1	No
AmbientOcclusion	0	N/A	4	0	No
AmbientOcclusion16	0	N/A	16	0	No
Reflections	0	N/A	0	0	Yes
SoftShadows	4096	4	0	4	No
SoftShadows8	8192	8	0	8	No
Low	2048	N/A	4	1	No
Medium	4096	4	4	4	No
High	4096	4	4	4	Yes
Ultra	8192	8	16	8	Yes

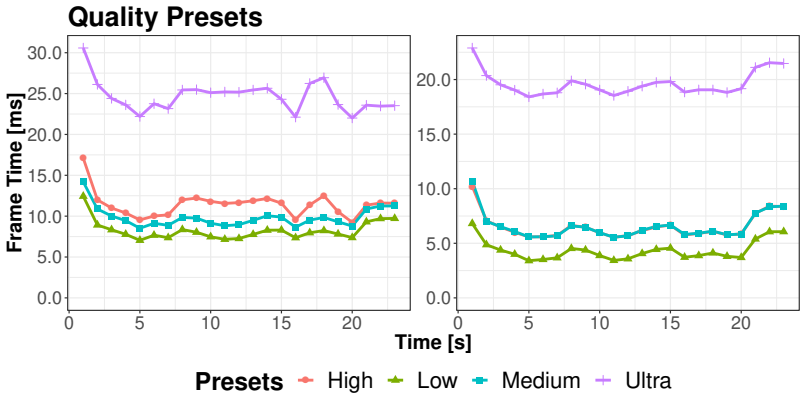
Prvním z cílů těchto testů je srovnání výkonnosti jednotlivých implementovaných efektů s jejich rasterizačními ekvivalenty. Jako metrika pro toto srovnání byl zvolen čas vykreslení snímků, který zahrnuje cenu vykreslovacích průchodů daného módu. Ze získaných dat byl vytvořen graf zobrazující všechny měřené presety, který lze najít na obr. 6.7.



Obrázek 6.7: Grafy zobrazující výkonnost při použití různých kvalitativních presetů. Grafy obsahují výsledky z hybridního sledování paprsků – *Cube* (**vlevo nahoře**), *Suzanne* (**vpravo nahoře**) a *Sponza* (**vlevo dole**). Poslední graf (**vpravo dole**) obsahuje výsledky z rasterizačního módu zobrazujícího scénu *Sponza*.

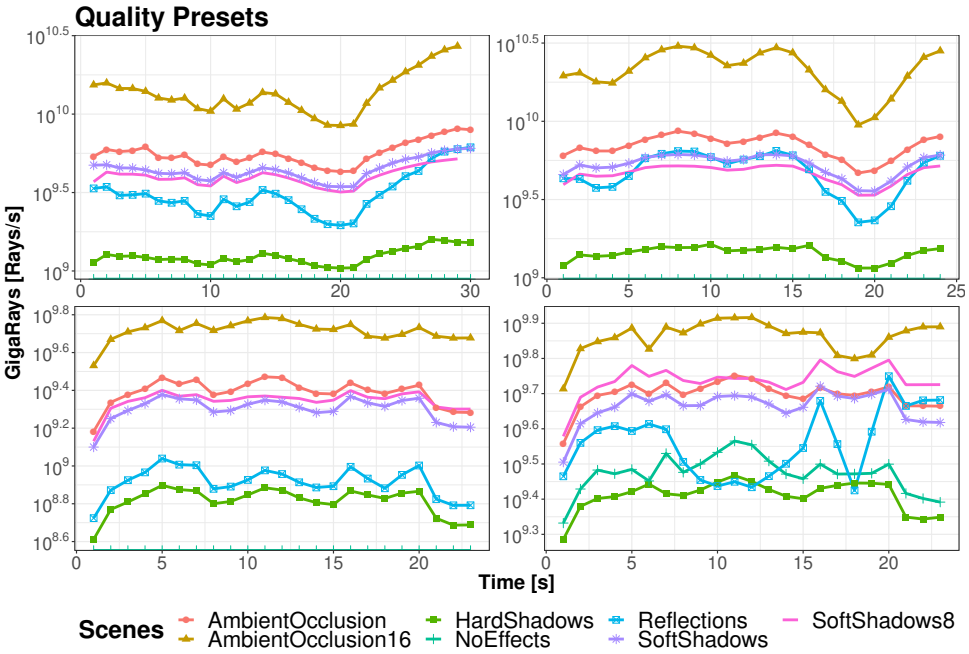
Z výsledků testování je patrné, že nejnáročnějším efektem jsou *měkké stíny*, jejichž aplikace redukuje výkon na zhruba 50%. Podle grafu potom následují efekty *Ambient Occlusion*, *tvrdé stíny* a *reflekce*. Bližšímu srovnání jednotlivých efektů je věnována sekce 6.3.

Při srovnání *Ambient Occlusion* a *tvrdých stínů* je zajímavé, že pokles výkonnosti při jejich použití je zhruba stejný i přes čtyřnásobné zvýšení počtu vyslaných paprsku v případě efektu *Ambient Occlusion*. Tento výsledek ukazuje, že paprsky s krátkou délkou jsou násobně levnější než ty bez omezení. Zároveň lze vidět, že i přes použití optimalizovaných paprsků v případě stínu (*Shadow Rays*), jsou krátké vzálenostní paprsky (*Distance Rays*) u *Ambient Occlusion* stále levnější. U efektu reflekce je nutné říci, že v první dvě scény neobsahují lesklé povrchy a proto je výsledek s reflekcí stejný jako ten bez efektu. U scény *Sponza* se rovněž efekt reflekce jeví jako velmi levný, ale toto lze přisoudit omezenému množství objektů ve scéně, které generují odrazové paprsky. Srovnání složitějších presetů, které již obsahují více druhů grafických efektů lze vidět na obr. 6.8.



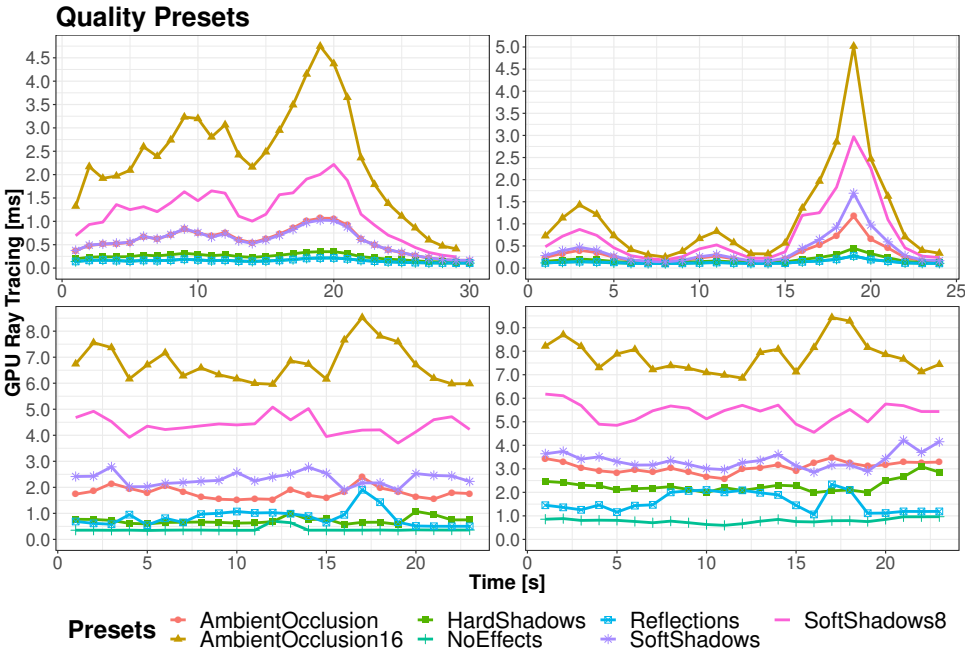
Obrázek 6.8: Srovnání složitějších presetů, které obsahují více grafických efektů. Popis jednotlivých presetů lze najít v tabulce 6.5. Vertikální osa obsahuje čas nutný pro vykreslení jednoho snímku.

Mezi měřené hodnoty patří také výkonnost v počtu GR/s, jejichž vizualizaci lze vidět na grafu 6.9. Graf je rozdělen podobně, jako v minulém případu, s rozdílem že v posledním grafu je zachycen průchod za použití módu *čistého sledování paprsku*. Zobrazené výsledky potvrzují výše uvedené tvrzení, že krátké *Ambient Occlusion* paprsky jsou nejlevnější, protože nejlépe vytěžují *Ray Tracing* jádra. Nejdražší jsou naopak *reflekční* paprsky, které spolu s malým počtem stínových paprsků vytěžují systém akcelerovaného sledování paprsku nejméně. Při srovnání s čistým sledováním paprsku lze opět vidět vyšší výkonnost v počtu vyslaných paprsků. Z grafů je také patrné, že při využití presetu *No Effects* je v případě *hybridního* přístupu počet vysílaných paprsků nulový, ale pro *čisté sledování paprsku* je nutné počítat s primárními paprsky.

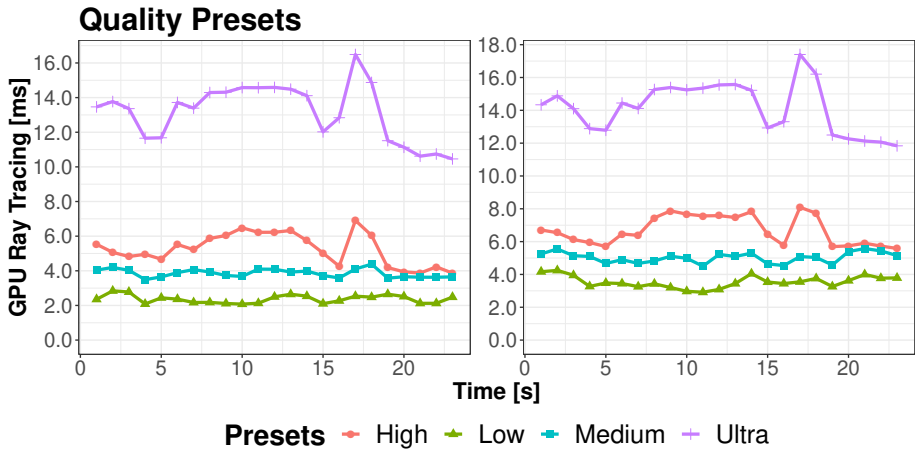


Obrázek 6.9: Grafy obsahující hodnoty naměřených vyslaných paprsků za sekundu pro hybridní sledování paprsku. Opět jde postupně o scény: *Cube*, *Suzanne* a *Sponza*, která je použita i pro poslední graf v kombinaci s *čistým* sledováním paprsků.

Vizualizaci délky výpočtu sledování paprsku na GPU lze vidět na obr. 6.10. Nejlevnějším efektem jsou opět *tvrdé stíny*, jejichž výpočet se v průměru pohybuje kolem $1ms$. Srovnání času pro složitější presety lze opět najít na oddělených grafech na obr. 6.11



Obrázek 6.10: Grafy zobrazující výpočetní čas sledování paprsku na GPU. Podobně jako v minulých případech jde postupně o scény: *Cube*, *Suzanne* a *Sponza*. Hodnoty posledního grafu jsou naměřeny ve scéně *Sponza* za použití *čistého* sledování paprsku.



Obrázek 6.11: Srovnání složitějších presetů, které obsahují více grafických efektů. Popis jednotlivých presetů lze najít v tabulce 6.5. Vertikální osa obsahuje čas výpočtu sledování paprsku na GPU.

6.3 Srovnání grafických efektů

Tato sekce obsahuje srovnání grafických efektů z pohledu náročnosti implementace, výkonnosti a vizuální kvality. Ze začátku je nutné zdůraznit, že srovnání je provedeno s výše popsanými (viz sekce 5.3) rasterizačními efekty. Tyto efekty reprezentují často používané přístupy při jejich výpočtu, ale nejsou reprezentativní z pohledu jejich efektivity. V reálných případech jsou tyto techniky používány s mnoha dalšími vylepšeními. Krátké srovnání s implementacemi v reálném herním enginu lze najít v sekci 6.5.

6.3.1 Tvrdé stíny

Prvním z porovnávaných efektů jsou *tvrdé stíny*. Vizuální srovnání obou variant lze vidět na obr. 6.12. V levé části obrázku lze vidět efekt generovaný pomocí sledování paprsku, který využívá jeden stínový paprsek. Pravá část obsahuje výstup z rasterizačního módu, využívající rozlišení stínových map 8192 pixelů. Doba výpočtu na GPU se v obou případech pohybovala kolem $1ms$.

Z pohledu implementační náročnosti jednoznačně vítězí přístup sledování paprsku, kdy je pouze nutné vyslat jeden paprsek ke světlu. Tento výpočet je navíc možné spojit s jinými efekty využívajícími sledování paprsku, čímž je možné redukovat počet průchodů pro vykreslení jednoho snímku. Z pohledu výkonnosti je možné dosáhnout lepších výsledků za použití rasterizace, protože generování jedné stínové mapy je výpočetně levné. Výhodou přístupu využívajícího sledování paprsků je jeho pomalu rostoucí doba výpočtu se zvyšujícím se počtem vykreslovaných trojúhelníků.

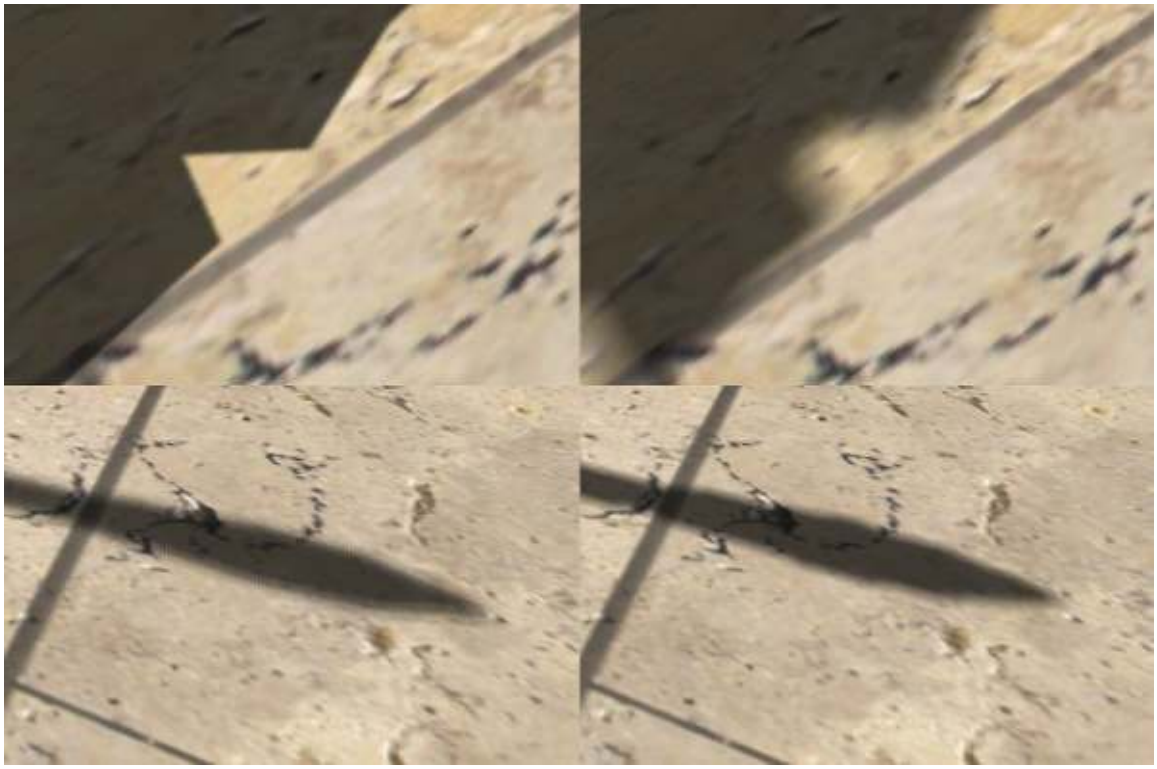
Obrázek 6.12: Srovnání tvrdých stínů při použití hybridního sledování paprsků (**vlevo**) a rasterizace (**vpravo**). Doba výpočtu na GPU se u obou případů pohybuje okolo $1ms$.

Na první pohled je z obrázku 6.12 patrné, že sledování paprsku generuje mnohem ostřejší a lépe definované stíny. Je nutné ale podotknout, že metoda stínových map je implementována ve velmi základní realizaci a pokročilejší metody vypadají mnohem lépe. Stíny generované sledování paprsku budou ale vždy tzv. *pixel-perfect* a nebudou ztrácet rozlišení z libovolné vzdálenosti.

6.3.2 Měkké stíny

Druhým srovnávaným efektem jsou měkké stíny, pro které lze opět vidět srovnání na obr. 6.13. Levá část obrázku obsahuje výstup metody využívající sledování paprsku, které používá 3 stínové paprsky. V pravé části je pro srovnání rasterizační přístup, který používá rozlišení stínových map 8192 pixelů a velikost kernelu 3×3. Doba GPU výpočtu se pro oba efekty pohybuje kolem $2ms$.

Zhodnocení z pohledu náročnosti implementace je podobné, jako v případě tvrdých stínů. Obě metody generující měkké stíny využívají v základu stejný přístup jako výše popsané tvrdé stíny. Drobnou nevýhodou přístupu využívající sledování paprsků je nutnost pseudo-náhodného generování jejich směrů pro optimální pokrytí zdroje světla. Výhodné je výborné škálování se vzrůstajícím počtem paprsků, které udává kvalitu výsledného efektu – s vyšším počtem paprsků je možné lépe pokrýt světelný zdroj a přesněji detekovat zastínění.



Obrázek 6.13: Srovnání měkkých stínů při použití hybridního sledování paprsků (**vlevo**) a rasterizace (**vpravo**). Objekt generující stín ve **vrchní** části je mnohem blíže než objekt v části **spodní**. Doba výpočtu na GPU se u obou případů pohybuje okolo $2ms$.

Z pohledu výkonnosti je lepší rasterizační přístup, který navíc oproti tvrdým stínům aplikuje pouze rozmazání jádrem o dané velikosti. Maximální velikost *penumbry* je udána právě velikostí jádra, nebo počtem paprsků. Pro přístup využívající sledování paprsku je navíc nutné výsledné stíny rozmazat, protože při malém počtu paprsků vzniká *Moaré* efekt. I přes výše zmíněné nevýhody je stále přístup využívající sledování paprsků, vizuálně věrohodnější – i za zachování stejného času výpočtu na GPU. Z obrázku lze vidět, že stíny generované pro bližší okluder si zachovávají ostrost. Naproti tomu rasterizační stíny jsou rozmazány v obou případech.

6.3.3 Ambient Occlusion

Dalším grafickým efektem je *Ambient Occlusion*, jehož srovnání lze vidět na obr. 6.14. V levé části je opět vidět výsledek generovaný za pomocí sledování paprsků, který používá 4 *vzdálenostní* paprsky. Pravá část obsahuje realizaci za použití rasterizace, která používá 8 vzorků. V obou případech byl výpočet na GPU dokončen za $2ms$.

Implementace efektu za využití sledování paprsku byla opět jednodušší a nevyžadovala tvorbu dalších vykreslovacích průchodů. Rasterizační přístup měl navíc problémy s nepřesností hloubkového bufferu, ve kterém je simulováno sledování paprsku. Obě metody používají stejné pseudo-náhodné generování paprsků a proto jsou v tomto ohledu rovnocenné.

Obrázek 6.14: Srovnání grafického efektu *Ambient Occlusion* při použití sledování paprsku (**vlevo**) a rasterizace (**vpravo**). Doba výpočtu na GPU se v obou případech pohybuje okolo $2ms$.

Nastavení kvality je v obou případech realizováno podobným způsobem – zvýšením počtu vysílaných parsků. Výsledkem přístupu využívajícího sledování paprsků i rasterizace je výstup, který obsahuje velké množství šumu, který je v obou případech nutné odstranit rozmazáním. Zajímavým výkonnostním parametrem je také rozsah okluze, který v případě sledování paprsku ovlivňuje zvyšuje cenu výsledného.

Vizuální srovnání, provedené s pomocí obr. 6.14, ukazuje nedostatky rasterizační metody. V několika případech lze vidět tzv. *falešné zastínění* (spodní obrázky v levé části oblouku), které vzniká díky generování efektu až v prostoru obrazovky, be kompletní znalosti scény.

6.3.4 Reflekce

Posledním z implementovaných efektů jsou reflekce, jejichž implementace byla ze všech efektů sledování paprsků nejnáročnější. Kromě generování akceleračních struktur nutných pro všechny výše popsané efekty, bylo také nutné umožnit shaderům přístup k vertex a index datům a texturám (viz sekce 5.1). Samotný výpočet odrazu je již jednoduchý, ale je navíc nutné počítat kompletní barvu zasaženého místa scény. Kromě barvy je také nutné vypočítat zastínění a případně *Ambient Occlusion*, aby byly tyto efekty v odraženém obraze

také vidět. Výkonnostně je tento efekt ze všech nejdražší, protože je nutné provést výpočet finální barvy dvakrát – nebo vícekrát pro několikanásobný odraz.

6.3.5 Zhodnocení

Nejvhodnějšími kandidáty pro použití sledování paprsku jsou efekty *tvrdých stínů* a *Ambient Occlusion*. Generování stínu je pro rasterizaci obzvlášť náročné v otevřených scénách. Výše představená technika využívající sledování paprsku pracuje velmi dobře i pro rozsáhlé scény. V případě *Ambient Occlusion* jde o kombinaci doby výpočtu její rasterizační alternativy a grafických artefaktů, které produkuje. Velkou výhodou je také relativně nízká cena jednotlivých paprsků, které i při nízkém počtu generují realistický výstup.

6.4 Stavba akceleračních struktur

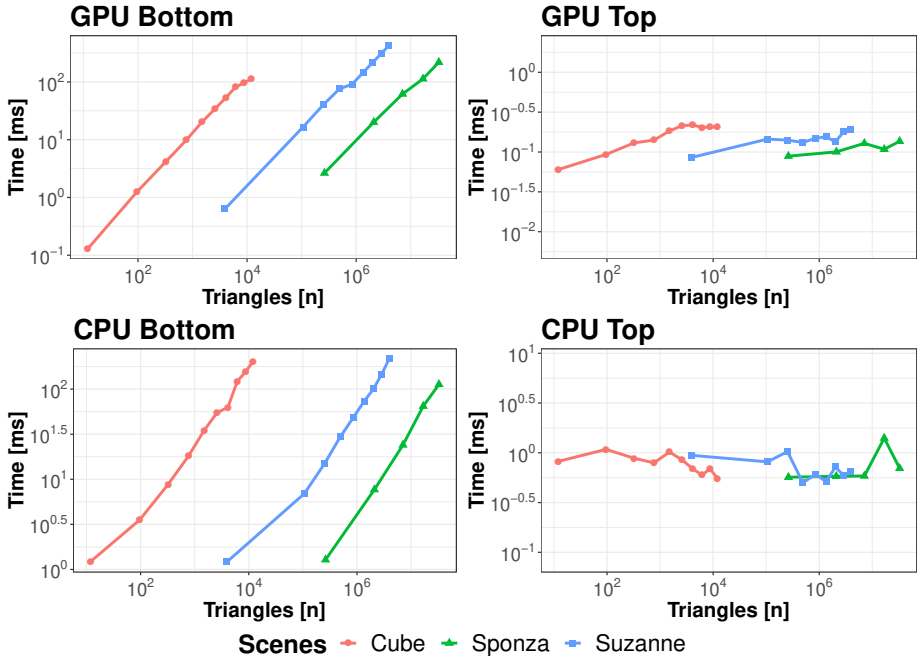
Kromě výkonnosti *Ray Tracing* jader je dalším důležitým parametrem také čas nutný pro stavbu akceleračních struktur. Tento parametr je obzvláště zajímavý v případě pohybujících-se objektů, kvůli kterým je nutné akcelerační struktury velmi často aktualizovat. Testy, jejichž výsledky jsou prezentovány v jednotlivých pod-sekcích, používají jako metriku časy naměřené při stavbě obou typů akceleračních struktur – vrchní i spodních vrstev – přičemž oddělené hodnoty jsou získány pro výpočet probíhající na CPU a GPU. Měřeny jsou také velikosti výsledných akceleračních struktur.

6.4.1 Duplikační testy

Cílem této kategorie testů je získat závislost času nutného pro stavbu akceleračních struktur a jejich velikosti na počtu trojúhelníků ve scéně. Z tohoto důvodu byla použita již výše popsaná duplikace celé scény, kterou je možné zvýšit aktuální počet trojúhelníků. Každý duplikát byl umístěn do vlastní akcelerační struktury spodní úrovně, čímž je simulován vyšší počet modelů. Testování bylo opět provedeno na primárních testovacích scénách – *Cube*, *Suzanne* a *Sponza* – jejichž parametry lze najít v tabulce 6.2. Samotné testy jsou opět automatizované a jejich cílem je postupné navyšování koeficientu duplikace až po předem danou hladinu.

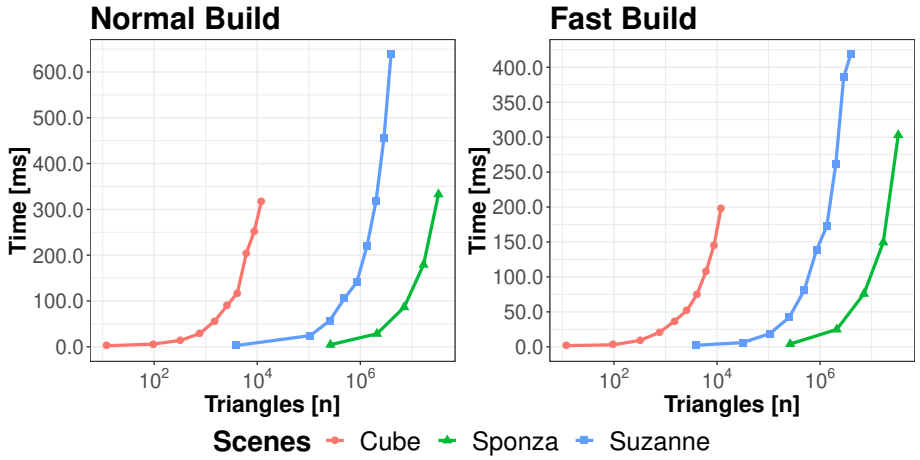
Prvním z testovaných parametrů byla časová náročnost jednotlivých operací nutných pro stavbu akceleračních struktur. Vizualizaci výsledných hodnot lze vidět na grafech na obr. 6.15. Jednotlivé body v grafech vždy reprezentují určitou úroveň duplikace – nulová v levé části grafu a maximální v pravé části grafu.

Z grafů lze rozpoznat závislost na počtu trojúhelníků při stavbě na grafickém akcelerátorů, která je lineární. V případě výpočtu na CPU je výpočet více zatížen celkovým počtem akceleračních struktur spodní úrovně. Pro vrchní vrstvu akceleračních struktur lze vidět mírný nárůst v době výpočtu pro zvyšující-se počet instancí ve scéně.



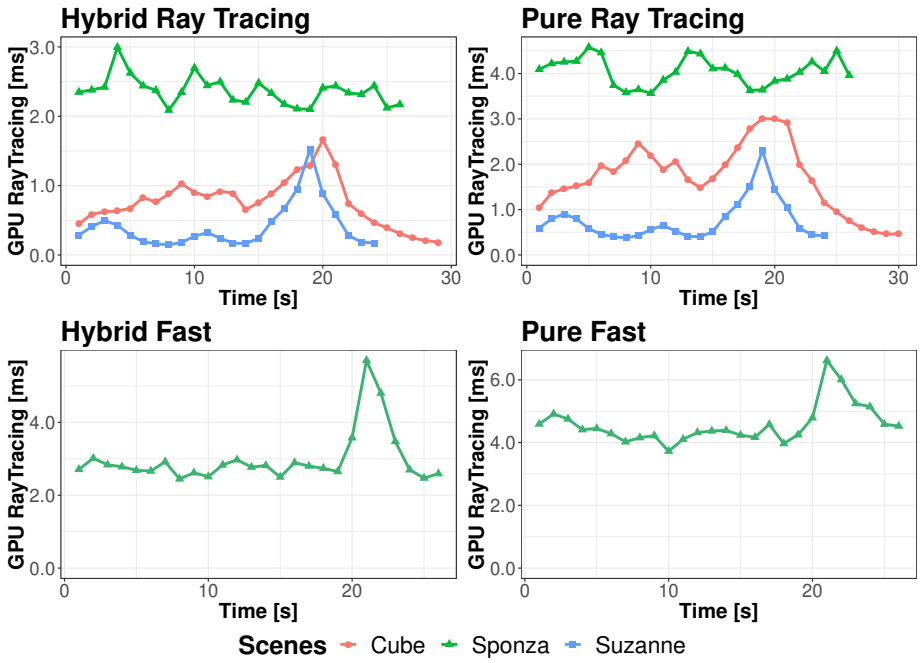
Obrázek 6.15: Grafy zobrazují dobu stavby akceleračních struktur na GPU (**nahoře**) a na CPU (**dole**). Časy jsou také rozděleny podle typu akcelerační struktury – vrchní vrstva (**vlevo**) a spodní vrstva (**vpravo**).

Druhým testovaným parametrem je přepínač stavby akceleračních struktur do „rychlého módu“, který produkuje výsledky za kratší čas, ale výsledná akcelerační struktura není tak efektivní při operaci sledování paprsků. Získané hodnoty lze vidět na obr. 6.16, který v levé části obsahuje graf hodnot bez použití rychlého módu a v pravé s rychlým módem. Naměřené časy jsou v tomto případě součtem všech výše popsaných časů – stavba obou typů akceleračních struktur na CPU i GPU. Na výsledcích lze vidět 30% až 40% zrychlení pro celkovou dobu stavby, v případě využití „rychlého módu“.



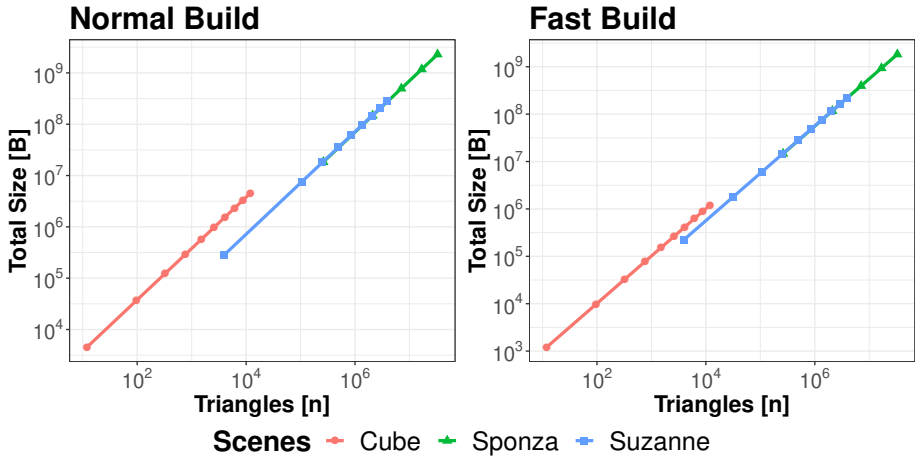
Obrázek 6.16: Graf obsahuje srovnání času nutného pro stavbu akceleračních struktur s výchozím nastavením (**vlevo**) a s nastavením pro maximální rychlost (**vpravo**).

Při použití rychlého módu pro stavbu akceleračních struktur je také zajímavé jakým způsobem tento přepínač ovlivní výkonnost sledování paprsku. Z tohoto důvodu je zde také přidán test ukazující měření výkonnosti na scéně *Sponza*, který lze vidět na obr. 6.17. Z výsledků lze vidět, že akcelerační struktury postavené v rychlém módu mají zhruba o 20% - 30% nižší výkonnost než při využití akceleračních struktury postavených bez rychlého módu.



Obrázek 6.17: Srovnání výkonnosti vysílání paprsků za použití akcelerační struktury s výchozím nastavením (**nahoře**) a s nastavením pro maximální rychlost stavby (**dole**). Zobrazeny jsou oba módy pro sledování paprsku – *hybridní* (**vlevo**) i *čistý* (**vpravo**).

Dalším důležitým parametrem jsou velikosti výsledných akceleračních struktur, které lze vidět na obr. 6.18. Z výsledků je patrná lineární závislost, která je pravděpodobně způsobena pevně daným výpočtem velikosti.

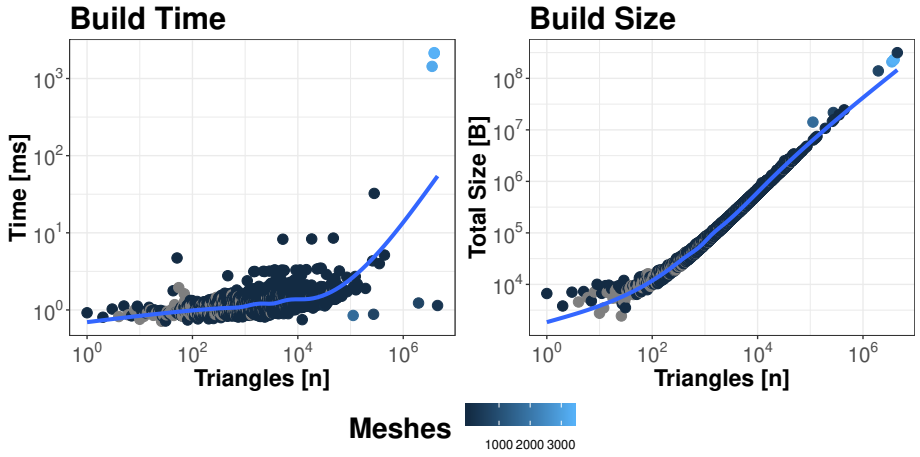


Obrázek 6.18: Srovnání paměťové náročnosti bufferů obsahujících akcelerační struktury. Opět lze vidět výsledky s výchozím nastavením (**vlevo**) a s nastavením pro maximální rychlost (**vpravo**). Udávané hodnoty zahrnují všechny akcelerační struktury.

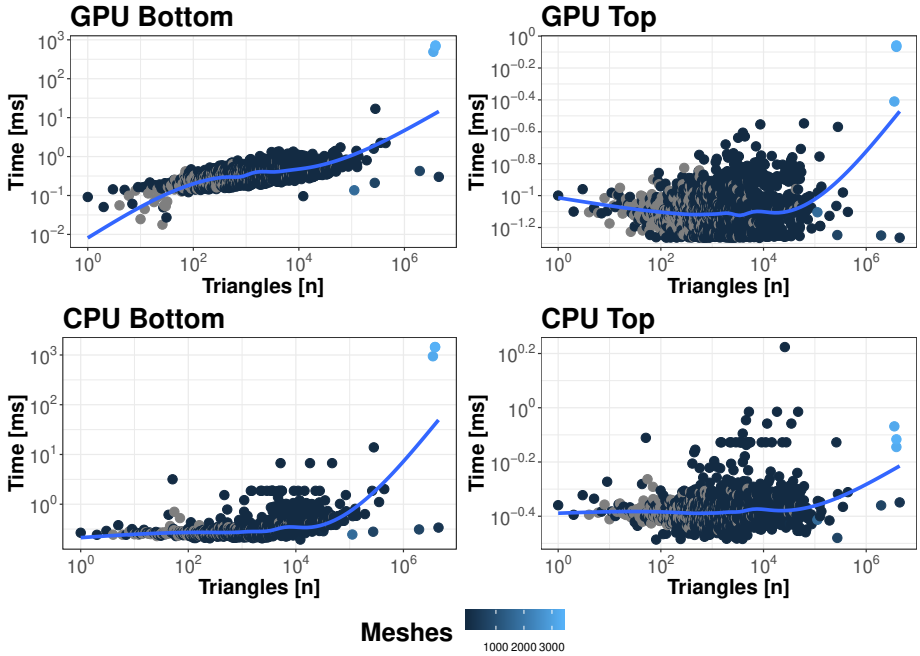
6.4.2 Testování modelů

Pro širší testování jakým způsobem ovlivňují parametry modelu dobu stavby akceleračních struktur byly provedeny testy mnoha různých scén a modelů. Základem těchto testů byla integrace do existujícího enginu, díky které bylo možno načítat velké množství scén a modelů (viz sekce 6.5). Kromě stavby akcelerační struktury pro celý model, bylo také provedeno testování na jednotlivých meshích modelu. Tento postup umožnil sesbírat mnoho testovacích dat, jejichž počet se pohybuje okolo 150.000.

Základem testů bylo opět měření doby nutné pro stavbu akceleračních struktur a jejich velikosti. Výsledky těchto testů lze vidět na obr. 6.19. Graf s oddělenými časy výpočtu lze vidět na obr. 6.20.



Obrázek 6.19: Grafy obsahující data ze stavby akceleračních struktur získaná testováním mnoha různých modelů. Do časů stavby akceleračních struktur (**vlevo**) byl započítán čas výpočtu na CPU i GPU a vrchních i spodních vrstev. Graf velikosti struktur (**vpravo**) rovněž počítá z celkovými paměťovými náklady vrchních i spodních vrstev.



Obrázek 6.20: Grafy zobrazující časy stavby různých testovaných modelů. Body jsou obarveny podle počtu meshů v daném modelu.

6.5 Náročnost integrace

Posledním z provedených experimentů byla integrace technologie hardwarové akcelerace sledování paprsků do již existujícího herního enginu. Tato část práce byla realizována jako součást stáže ve firmě *Hangar13*. Cílem byla implementace vykreslování grafických efektů za pomocí akcelerovaného sledování paprsku za cílem zjištění budoucího potenciálu této technologie. Samotná práce zahrnovala také srovnání s grafickými efekty implementovanými v použitém *in-house* enginu z pohledu vizuální kvality a náročnosti jejich výpočtu.

Prvním zjištěním, které vyplynulo z této spolupráce, je otázka jakým způsobem efektivně dostat data modelů a scén do akceleračních struktur. Podpora pohybujících-se objektů je relativně jednoduše implementovatelná pomocí generování vrchních vrstvy akceleračních struktur pro každý vykreslovaný snímek, což je reálné i pro větší scény (viz výsledky v sekci 6.4). Objekty, které se deformují, nebo využívají technik kosterní animace, nanášení kůže atp. jsou problematické z důvodu opakované tvorby akceleračních struktur spodní úrovně, které jsou již relativně výpočetně náročné. Systém akcelerovaného sledování paprsků sice umožňuje aktualizaci akceleračních struktur, čímž je redukována doba stavby, ale požadavkem je zachování stejného počtu primitiv [1]. Jednou možností, jak tento problém řešit je asynchronní výpočet těchto struktur, spolu s ukládáním již vytvořených struktur do databáze pro pozdější znovu-použití. Tento přístup lze také kombinovat s výše popsaným rychlým módem stavby akceleračních struktur, který by pro omezené množství objektů ve scéně nemusel výkonnost příliš omezovat. Výsledná akcelerační struktura potom může reprezentovat model v některé dřívější fázi animace, ale pokud frekvence aktualizace bude dostatečně vysoká, potom může výsledek působit dostatečně věrohodně.

Při experimentování byly opět nejvýhodnějším efektem tvrdé stíny. Cílovým použitím je primárně globální světlo – například slunce – které ve volném prostoru osvětluje prostor do velké vzdálenosti. Při řešení tohoto typu stínů v případě rasterizace je nutné použít speciální metody – například kaskádové stíny kombinované s perspektivním mapováním – jejichž cena je již srovnatelná s metodou využívající sledování paprsku. Pro velké množství lokálních světel by také bylo možné použít přístup podobný *Clustered Shading* spolu s výběrem světel, které do daného segmentu zasahují. Pokud by v tomto případě došlo k plnému využití všech *Ray Tracing* jader, bylo by také možné použít náhodný výběr omezeného množství světel v každém snímku. Výsledek pro jednotlivá světla by potom bylo možné mezi snímky akumulovat. Dalším možným přístupem může být výpočet zastínění pro nižší rozlišení výstupního rastru, čímž by došlo k radikálnímu snížení počtu paprsků.

Mezi ostatními implementovanými efekty byl zajímavým také efekt *Ambient Occlusion*, který je výpočetně srovnatelný se standardně používanými metodami. Jeho implementace je také velmi jednoduchá a nevyžaduje příliš vstupních informací o vykreslované geometrii – pouze *G-Buffer* ze kterého je získána pozice ve scéně a normála. Efekt měkkých stínů je v aktuálně implementované podobě příliš drahý pro reálné nasazení pro plošná světla. Pro zlepšení jeho vlastností by bylo možné opět použít akumulaci mezi snímky a techniku *Monte Carlo*. Nejnáročnějším efektem jsou odrazy, pro které by bylo nutné implementovat shadery materiálů všech použitých materiálů. V reálném použití budou pravděpodobně reflekce pouze napodobovat odražený objekt s jednodušším stínováním.

Ve výsledném zhodnocení došlo ke shodě, že akcelerované sledování paprsků má velmi dobrý potenciál a pravděpodobně bude v následujících letech často používán pro realizaci některých grafických efektů – například stínů a globálního osvětlení.

6.6 Možnosti vylepšení

Cílem této sekce je souhrn dalších směrů, kterými by bylo možné tuto práci rozšířit a možnosti vylepšení technologie akcelerovaného sledování paprsků skrz API *DirectX Ray Tracing*. Mezi možnosti dalšího vývoje patří využití aktualizace akceleračních struktur. Kromě měření parametrů této funkcionality, by bylo také možné implementoval plně animované scény s deformovatelnou geometrií. Dalším potenciálním směrem je implementace grafického efektu globální osvětlení, která by mohla díky hardwarové akceleraci sledování paprsku mohla nahradit dříve používané metody.

S globálním osvětlením také souvisí testování rekurzivního sledování paprsků, pro které by bylo zajímavé zjistit cenu jednotlivých paprsků při zvyšování úrovně rekurze. Další parametr zajímavý z pohledu výkonnosti výsledného systému je velikost *payload* struktury spojené s každým paprskem. Zde by bylo také možné zjistit, jakým způsobem ovlivňuje počet „živých“ proměnných používaných před i po vyslání paprsku výslednou výkonnost celého procesu. Toto měření je obzvláště zajímavé, protože všechny „živé“ proměnné je nutné při předání paprsku *Ray Tracing* jádrům uložit na zásobník a opět je načíst po dokončení akce výpočtu[6].

Při implementaci akcelerace sledování paprsku pomocí *DirectX Ray Tracing* jsem se v mnoha situacích setkal s problémy s vykreslováním, u kterých bylo velmi časově náročné zjistit která část procesu nepracuje správně. Využití *ladících vrstev*, které jsou součástí *DirectX 12*, v některých situacích chybu neodhalilo i když šlo podle standardu o nekorektní použití aplikačního rozhraní – například neplatné nastavení v předávané struktuře. Kromě pomocných vrstev byl při vývoji také použit nástroj *PIX*, který i přes podporu *DXR* většinu chybných stavů opět neodhalil. Mnoho nástrojů akceleraci sledování paprsku navíc vůbec nepodporuje – například často používaný nástroj *RenderDoc*. Pro rozšíření této technologie bude jistě důležitá kvalita ladících nástrojů, jejichž vývoj by měl mít vysokou prioritu.

V průběhu práce s rozhraním a implementací jednotlivých efektů jsem také došel k několika vylepšením, které by bylo užitečné integrovat do následujících revizí *DXR*. První z nich je umožnit uživateli nastavit sledování paprsku v shaderu (například *flag*) tak, aby docházelo k postupné invokaci *closest-hit* shaderů podle vzdálenosti geometrie od počátku paprsku. Příkladem použití této vlastnosti je například akumulace barev pro průhledné materiály. Další užitečnou vlastností by byla také možnost získat sousední geometrická primitiva pro daný zásah geometrie – například pokud jsou ve stejné části akcelerační struktury.

Při implementaci grafického efektu *Ambient Occlusion* bylo kvůli útlumu efektu se vzdáleností nutné vysílat výše popsané *vzdálenostní paprsky*, jejichž vyhodnocení vyžadovalo spuštění *closest-hit* shaderů. Implementace těchto shaderů ale obsahuje pouze nastavení vzdálenosti zásahu do struktury paprsku, která by mohla být provedena i bez jeho invokace. Tato funkce by v případech vysílání velkého množství těchto paprsků umožnila zvýšení výkonnosti. Poslední z navrhovaných vylepšení je možnost nastavit multiplikátor prvku *InstanceConributionToHitGroupIndex*, s jehož pomocí se vybírá vyvolaný *closest-hit* shader. Existence tohoto multiplikátoru by umožňovala tvorbu společných *closest-hit* shaderů pro všechny instance ve scéně – například pro *vzdálenostní paprsky* – čímž by se zmenšila velikost tabulky shaderů.

[6] Viz https://devblogs.nvidia.com/rtx-best-practices/ .

Kapitola 7

Závěr

Tato práce se zabývala použitelností systému hardwarové akcelerace sledování paprsku a jeho kombinací s rasterizací do hybridního vykreslovacího enginu využívajícího rozhraní *DirectX Ray Tracing*. Práce obsahuje rozbor důležitých konceptů z oblasti sledování paprsku a výběr množiny grafických efektů s jejichž pomocí je systém testován.

Hlavními přínosy práce jsou návrh a implementace hybridního vykreslovacího enginu nazvaného *Quark*, který kombinuje rasterizaci a hardwarově akcelerované sledování paprsku. Engine je navržen modulárně a pro obecné použití, čímž je umožněno jeho využití k dalším experimentům. Implementovány jsou také možnosti testování a profilování na CPU a GPU.

Samotné zhodnocení technologie bylo realizováno z několika pohledů – srovnání grafických efektů, testování parametrů systému a integrace technologie do existujícího enginu.

V odpovědi na otázku položenou v úvodu této práce – zda je tato technologie již použitelná v dnešních zobrazovacích enginech – je za použití výše popsaných výsledků nutné odpovědět pozitivně. I přes zjevné nevýhody velmi náročné implementace a omezené hardwarové podpory na současných zařízeních, vykazuje výsledný systém velmi dobré parametry, které tyto nevýhody převažují. Výkonem se akcelerátor *RTX 2080 Ti* v reálných podmínkách blíží hodnotě 7 $GigaRays/s$, která odpovídá výsledkům udávaným výrobcem. Z testovaných grafických efektů byly vybrány dva – *Tvrdé stíny* a *Ambient Occlusion* – které jsou obzvláště výhodné, díky jejich relativně jednoduché implementaci a efektivnímu využití *Ray Tracing* jader.

Součástí práce byla také integrace technologie akcelerovaného sledování paprsků do herního enginu ve firmě *Hangar13*, která se dlouhodobě zabývá vývojem her. Díky této zkušenosti je možné odpovědět i na druhou z položených otázek, která se dotazuje zda je technologie využitelná i v real-time grafických systémech. Odpověď je opět pozitivní, ale kromě výše zmíněných problémů se v tomto případě ukazují také další nevýhody ve formě výpočetně náročné stavby akceleračních struktur a složité reprezentace vykreslovaných scén.

Mezi možnosti dalšího vývoje projektu patří například: využití aktualizace akceleračních struktur, testování modelů s vysokým počtem trojúhelníků, implementace efektu globálního osvětlení, nebo měření na grafických akcelerátorech s architekturou *Pascal*, které nedávno získaly podporu akcelerace sledování paprsku využívající softwarového řešení.

Práce byla úspěšně prezentována na studentské konferenci **Excel@FIT 2019**, kde byla oceněna odborným panelem za použití moderních technologií. Dosažené výsledky ukazují velký potenciál využití této technologie v herních i obecných zobrazovacích enginech a bude pravděpodobně, i přes výše popsané nevýhody, značně využívána v následujících letech.

Literatura

[1] DirectX Specs. Online, Květen 2019.
URL https://microsoft.github.io/DirectX-Specs/

[2] Appel, A.: Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference on - AFIPS 68 (Spring)*, ACM Press, 1968, doi:10.1145/1468075.1468082.

[3] Bavoil, L.; Sainz, M.; Sainz, M.: Image-Space Horizon-Based Ambient Occlusion. SIGGRAPH 2008, 2008.
URL https://developer.download.nvidia.com/presentations/2008/SIGGRAPH/HBAO_SIG08b.pdf

[4] Chan, E.: Rendering Fake Soft Shadows with Smoothies. SIGGRAPH 2004, 2004.
URL http://jankautz.com/courses/ShadowCourse/06-Smoothies.pdf

[5] Cook, R. L.; Porter, T.; Carpenter, L.: Distributed ray tracing. *ACM SIGGRAPH Computer Graphics*, ročník 18, č. 3, jul 1984: s. 137–145, doi:10.1145/964965.808590.

[6] Foley, J. D.; van Dam, A.; Feiner, S. K.; aj.: *Computer Graphics*. Addison Wesley, 2013, ISBN 0321399528.

[7] Fujimoto, A.; Tanaka, T.; Iwata, K.: ARTS: Accelerated Ray-Tracing System. *IEEE Computer Graphics and Applications*, ročník 6, č. 4, 1986: s. 16–26, doi:10.1109/MCG.1986.276715.

[8] Havran, V.: Zobrazovací rovnice a formy jejího vyjádření. 2014.
URL http://cw.fel.cvut.cz/b172/_media/courses/b4m39rso/lectures/04-x39rso-rendering_eq.pdf

[9] Immel, D. S.; Cohen, M. F.; Greenberg, D. P.: A radiosity method for non-diffuse environments. *ACM SIGGRAPH Computer Graphics*, ročník 20, č. 4, aug 1986: s. 133–142, doi:10.1145/15886.15901.

[10] Kajiya, J. T.: The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH 86*, ACM Press, 1986, doi:10.1145/15922.15902.

[11] Kay, T. L.; Kajiya, J. T.: Ray tracing complex scenes. *ACM SIGGRAPH Computer Graphics*, ročník 20, č. 4, aug 1986: s. 269–278, doi:10.1145/15886.15916.

[12] Kilgariff, E.; Moreton, H.; Stam, N.; aj.: NVIDIA Turing Architecture In-Depth. Online, Září 2018.
URL https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/

[13] Kvarfordt, D.; Lillandt, B.: Screen Space Ambient Occlusion. Online, 2017.
URL http://www.cse.chalmers.se/edu/course/TDA361/Advanced%20Computer%20Graphics/SSAO.pdf

[14] Křivánek, J.: Rendering equation and its solution. 2015.
URL https://cgg.mff.cuni.cz/~jaroslav/teaching/2015-npgr010/slides/07%20-%20npgr010-2015%20-%20rendering%20equation.pdf

[15] Nah, J.-H.; Park, J.-S.; Park, C.; aj.: T&I engine. In *Proceedings of the 2011 SIGGRAPH Asia Conference on - SA '11*, ACM Press, 2011, doi:10.1145/2024156.2024194.

[16] NVIDIA: NVIDIA Turing GPU architecture. 2018.
URL https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[17] Owens, B.: Forward Rendering vs. Deferred Rendering. Online, Říjen 2013.
URL https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342

[18] Polášek, T.: *Komponentní systém pro herní grafický engine*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=19918

[19] Stachowiak, T.: Stochastic Screen-Space Reflections. SIGGRAPH 2015, 2015.
URL
https://www.ea.com/frostbite/news/stochastic-screen-space-reflections

[20] Umenhoffer, T.; Patow, G.; Szirmay-Kalos, L.: Robust Multiple Specular Reflections and Refractions. Online.
URL https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch17.html

[21] Veach, E.; Guibas, L. J.: Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH 97*, ACM Press, 1997, doi:10.1145/258734.258775.

[22] Walter, B.; Marschner, S. R.; Li, H.; aj.: Microfacet Models for Refraction through Rough Surfaces. 2007, doi:10.2312/egwr/egsr07/195-206.
URL https://www.cs.cornell.edu/~srm/publications/EGSR07-btdf.pdf

[23] Whitted, T.: An improved illumination model for shaded display. *Communications of the ACM*, ročník 23, č. 6, jun 1980: s. 343–349, doi:10.1145/358876.358882.

Příloha A

Plakát

Hybrid RayTracing in DXR

Autor: Tomáš Polášek
Vedoucí práce: Ing. Jozef Kobrtek

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019

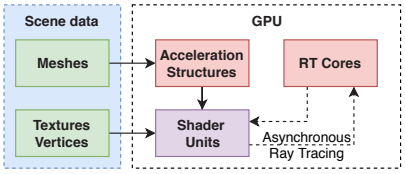
1. Motivace

Cílem této práce je zhodnotit míru použitelnosti hardwarové akcelerace sledování paprsků na grafických akcelerátorech *Nvidia Turing*, za využití rozhraní *DirectX Ray Tracing*.
Hodnocení je realizováno z několika pohledů - výkonnost systému, náročnost integrace a jednoduchost implementace grafických efektů. K dosažení tohoto cíle je součástí práce návrh a implementace **Hybridního vykreslovacího enginu Quark**, využívajícího grafického rozhraní **DirectX 12**.

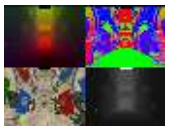
Quark
Hybrid RayTracing

2. Akcelerace Sledování Paprsku

Grafické akcelerátory *Nvidia Turing* obsahují tzv. **Ray Tracing** jádra, ve kterých dochází k asynchronnímu výpočtu průniku paprsků s geometrií.



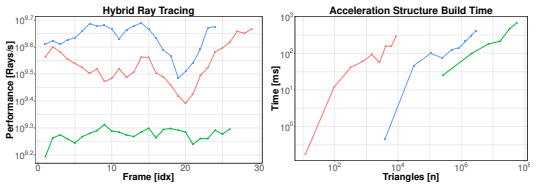
3. Hybridní Vykreslování



Hybridní vykreslování je metoda kombinující výhody rasterizace s možnostmi generování efektů pomocí sledování paprsku. Jejím cílem je snížení počtu prozatím stále drahých operací vyslání paprsku. V principu je podobná dříve používaným *Deferred Rendering* technikám, kdy je scéna před stínováním vykreslena do tzv. **G-Bufferu** (**vpravo**), který je použit při sledování paprsku.

4. Experimenty a Testování

Mezi hlavní testované parametry patří počet GigaRays/s (**vlevo**, průchod scénou) a časy pro stavbu akceleračních struktur (**vpravo**). Prezentované výsledky byly pořízeny na zařízení **RTX 2080 Ti**; hodnocení a další testy lze nalézt pod odkazem **Paper**.



5. Výsledky

Systém akcelerovaného sledování paprsku dosahuje očekávaných výsledků – **5 až 12 GigaRays/s**. Doba stavby akceleračních struktur se blíží **jednotkám milisekund** i pro menší vstupy – 50 tisíc trojúhelníků – čímž může být omezena použitelnost pro dynamické modely. Technika **Hybridního vykreslování** umožňuje věrnou realizaci efektů – stíny, ambient occlusion a odrazy – v reálném čase.

Odkazy

Repository | Paper | Video

NVIDIA GEFORCE RTX

16

Příloha B

Ukázka logu

```
Configuration:
Command line parameters  = Measurement.exe --scene
Suzanne/Suzanne.gltf --profile-automation
--profile-camera-track Suzanne/Suzanne.trk --rt-mode
--width 2560 --height 1440 --profile-output
PresetSuzanne1440RtUltra --quality-preset Ultra
Build type               = Release
Device name              = NVIDIA GeForce RTX 2080 Ti
Device memory MB         = 11049
Ray Tracing backend      = Hardware
Window width             = 2560
Window height            = 1421
Window title             = Direct3D App
Back-buffer count        = 3
Fill Triangles           = enabled
Triangles Font Clock-Wise= enabled
Fast Build AS            = disabled
Target FPS               = 0
Target UPS               = 60
Tearing                  = enabled
VSync                    = disabled
GUI rendering            = disabled
Debug layer              = disabled
GPU validation           = disabled
GPU profiling            = enabled
Render mode              = Ray Tracing
Loaded scene file        = Suzanne/Suzanne.gltf
Using testing scene      = disabled
Duplication              = 1
Duplication in cube      = disabled
Duplication offset       = 10
BLAS duplication         = disabled
Rays per pixel           = 33
Quality preset           = Ultra
Rasterization Quality    :
  AO                     = enabled
  AO kernel size         = 16
  Shadows                = enabled
  Shadow map resolution  = 8192
  Shadow PCF kernel size = 8
  Reflections            = enabled
Ray Tracing Quality      :
  AO                     = enabled
  AO rays                = 16
  AO kernel size         = 8
  Shadows                = enabled
  Shadow rays            = 8
  Shadow kernel size     = 8
  Reflections            = enabled

ProfilingAggregator:
Render , 10.3575 , 11.2371 , ...
Update , 0.0674 , 0.0707 , ...
RayTracing , 0.1745 , 0.1747 , ...
RayTracingGpu , 9.34288 , 10.0515 , ...
DeferredRLGpu , 0.137888 , 0.1824 , ...
RayTracingRLGpu , 1.42186 , 2.22701 , ...
ResolveRLGpu , 7.78003 , 7.63955 , ...
Index , 0 , 1 , ...

ProfilingBuildAggregator:
BuildBottomLevelAS , 0.8794 , ...
BuildBottomLevelASGpu , 0.257728 , ...
BuildTopLevelAS , 0.8445 , ...
BuildTopLevelASGpu , 0.059776 , ...
Duplication , 1 , ...
Triangles , 3936 , ...
Meshes , 1 , ...
BottomLevels , 1 , ...
TopLevelSize , 64 , ...
BottomLevelsSize , 222592 , ...
Index , 0 , ...

FpsAggregator:
FPS , 81 , 91 , ...
UPS , 59 , 60 , ...
FrameTime , 12.3982 , 10.9957 , ...
GigaRays/s , 9.68254 , 10.9176 , ...
Index , 0 , 1 , ...


AccelerationStructure:
Acceleration Structure building statistics:
    Bottom Level Acceleration Structure:
        Maximum [B]: 222592
        Minimum [B]: 222592
        Currently Allocated [B]: 222592
        Currently Created [num]: 1
        Current Average [B]: 222592
        Maximum Triangles [num]: 3936
        Minimum Triangles [num]: 3936
        Total Triangles [num]: 3936
        Maximum Meshes [num]: 1
        Minimum Meshes [num]: 1
        Total Meshes [num]: 1
    Top Level Acceleration Structure:
        Maximum [B]: 64
        Minimum [B]: 64
        Currently Allocated [B]: 64
        Total Allocated [B]: 64
        Total Created [num]: 1
        Average Allocated [B]: 64
        Maximum Instances [num]: 1
        Minimum Instances [num]: 1
        Total Instances [num]: 1
    Scratch Buffer:
        Maximum [B]: 32896
        Minimum [B]: 32896
        Currently Allocated [B]: 32896
        Total Allocated [B]: 32896
        Total Created [num]: 1
        Average Allocated [B]: 32896
ProfilingStack:
Profiling statistics:
    Scopes entered : 121868
    Scopes exited : 121867
    Overhead per scope [ticks] : 104.5
    Threads entered : 3
    Threads exited : 0
    ============================
    ... Profiling Data ...
    ============================
```

Příloha C

Obsah přiložné SD karty

Přiložená karta obsahuje:

- `archive/` – Obsahuje záložní archivy s obsahem této karty.
- `exec/`:
  - Preložená verze testovací aplikace.
  - `AppManual.pdf` – Manuál ovládání grafické části aplikace.
- `src/`:
  - Zdrojové texty programové části práce.
  - `Apps/` - Zdrojové kódy testovací aplikace.
  - `DX12-Tutorial/` - Ukázkové aplikace využívající vytvořený engine.
  - `Engine/`:
    - Hybridní vykreslovací engine Quark.
    - `Lib/` - Knihovny použité při vývoji engine.
  - `EngineTests/` - Testy některých částí vykreslovacího engine.
  - `Lib/` - Obecně používané knihovny.
  - `Models/` - Ukázkové modely a scény použité při testování.
  - `Prof/`:
    - Profilování a výsledné logy.
    - `Automated/` - Skripty pro automatické profilování.
    - `Automated/Graphs/` - Skripty pro generování grafů.
  - `Res/` - Další použité prostředky a modely.
- `src_poster/`:
  - Zdrojové texty posteru na konferenci Excel@FIT.
- `src_text/`:
  - Zdrojové texty technické zprávy.
- `video/` - Videa obsahující vizuální ukázky z testování.
- `README.txt` – Popis obsahu karty a jeho použití.
- `xpolas34-DXR.pdf` – Text písemné zprávy.
- `xpolas34-DXR-tisk.pdf` – Text písemné zprávy určený k tisku.

Příloha D

Ovládání aplikace