ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Bakalářská práce

Renderovací pipeline a podpora ray-tracing v Unity

Josef Bacík
Otevřená informatika (BS) - Počítačové hry a grafika

Leden 2022
Vedoucí práce: Ing. David Sedláček, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bacík** Jméno: **Josef** Osobní číslo: **483624**

Fakulta/ústav: **Fakulta elektrotechnická**

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

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Renderovací procesy v Unity pro virtuální realitu

Název bakalářské práce anglicky:

Rendering pipelines in Unity for virtual reality

Pokyny pro vypracování:

Popište renderovací procesy (rendering pipeline) dostupné v Unity (např. tzv. standardní, URP, HDRP). Dále prostudujte a popište podporu Unity pro realtime raytracing (omezení, požadavky) i s ohledem na využití pro zobrazování pro virtuální realitu (VR), vytvořte základní demonstrační testy pro ukázku možností dostupných efektů (Ambient Occlusion, Global Illumination, Reflections, Shadows, Recursive rendering), a pro výkonnostní posouzení.
Navrhněte proces konverze VR scény vytvořené v URP do HDRP i s podporou Raytracingu. Demonstrujte konverzní proces na scéně vybrané ve spolupráci s vedoucím práce. Porovnejte konvertované scény (URP, HDRP, HDRP + Raytracing) z výkonostního hlediska na různém HW (jak rozdílné CPU a graf. karta, tak různé brýle pro VR - např. HTC vive, HTC vive Pro, Xtal 8K, Oculus Quest/2 s kabelem).

Seznam doporučené literatury:

1) J. Žára, B. Beneš, J. Sochor, P. Felkel, Moderní počítačová grafika, 2. vydání. Computer Press, 2004.
2) Tomas Akenine-Moller et al., Real-Time Rendering (4th edition). CRC Press, 2018.
3) Haines et al. Ray Tracing Gems, Apress, 2019.
4) Unity documentation, HDRP - Ray Tracing, https://docs.unity3d.com/. 2021, Online.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Sedláček, Ph.D., katedra počítačové grafiky a interakce FEL

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

Datum zadání bakalářské práce: **13.09.2021** Termín odevzdání bakalářské práce: **04.01.2022**

Platnost zadání bakalářské práce: **19.02.2023**

Ing. David Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Děkuji Ing. Davidovi Sedláčkovi, Ph.D. za vedení mé bakalářské práce, ochotnou pomoc při praktické i teoretické části této práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

.....
V Praze dne 4. 1. 2022
Josef Bacík

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

.....
In Prague 4. 1. 2022
Josef Bacík

Abstrakt / Abstract

Předmětem této bakalářské práce je analýza různých renderovacích pipeline v Unity, popsání a otestování výkonnosti efektů ray-tracingu v Unity a popsání procesu konverze projektu z jedné renderovací pipeline do druhé

Klíčová slova: render pipeline; ray-tracing; výkon ray-tracingu; VR; Unity; HDRP; URP; Konverze.

The subject of this thesis is to analyze different rendering pipelines in Unity, describe and test performance of ray-tracing effects in Unity and describe conversion process of a project from one render pipeline to another

Keywords: render pipeline; ray-tracing; ray-tracing performance; VR; HDRP; URP; Conversion.

Title translation: Rendering pipeline and ray-tracing support in Unity

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Tato bakalářská práce se zabývá několika problémy.

Popisuje vykreslování a vykreslovací procesy v herním enginu Unity. Navrhuje a zjednodušuje proces konverze z vykreslovacího procesu URP do vykreslovacího procesu HDRP s podporou ray-tracingu. Popisuje ray-tracing a podporu různých efektů ray-tracingu v Unity a výkonnostně testuje jak jednotlivé efekty ray-tracingu, tak scény před konverzí a po konverzi. Scény před konverzí a po konverzi testuje na různých grafických kartách a různých headsetech pro virtuální realitu.

Převod z URP do HDRP řeším, jelikož se při vývoji aplikace může stát, že vývojáři nestačí funkce v jednodušším renderovacím procesu a potřebuje vylepšit projekt, aby mohl používat více efektů vykreslování. Zároveň otestuji vliv na výkon před a po vylepšení, abych zjistil vliv vykreslovacích procesů na výpočetní zátěž různých počítačů a případně brýlí pro virtuální realitu.

Ray-tracing je poměrně nová funkce v Unity. Proto popíši jednotlivé efekty ray-tracingu podporované v Unity a otestuji jejich výpočetní náročnost.

Kapitola 2
Teoretická část

Unity je herní engine používaný pro vývoj 2D i 3D her na různé platformy, byl vyvinut v společností Unity Technologies a představen v roce 2005. Tato kapitola je věnována vykreslovacímu enginu v Unity a vykreslování obecně.

2.1 Render pipeline

Render pipeline (vykreslovací proces) je model popisující kroky potřebné k vykreslení virtuální scény na obrazovku. Jelikož záleží na použitém software a hardware, není žádný unifikovaný render pipeline použitelný pro všechny aplikace. Render pipeline se (podle Unity dokumentace [1])rozděluje na redukování (culling), vykreslování (rendering) a následné zpracování (post-processing). Podrobněji se můžeme dočíst o počítačové grafice například v knize Moderní Počítačová grafika[2], kde je zároveň zmíněno mnoho pojmů z následujících kapitol.

2.1.1 Culling

Culling je obecně jakýkoliv proces, ve kterém se z následujícího vykreslování odebírají objekty, které nebude potřeba vykreslit, protože nebudou na kameře viditelné. Snažíme se tak co nejvíce snížit náročnost při vykreslování a ušetřit výkon.[3]

Frustrum culling je základní technika cullingu, při které se odstraní ze seznamu objektů k vykreslení (ořežou) všechny objekty, které nejsou v pohledovém kuželu kamery. Back-face culling se stará o odstranění odvrácených stran objektů ze seznamu vykreslovaných objektů. Například člověku otočenému k vám čelem nemusíte vykreslovat záda.

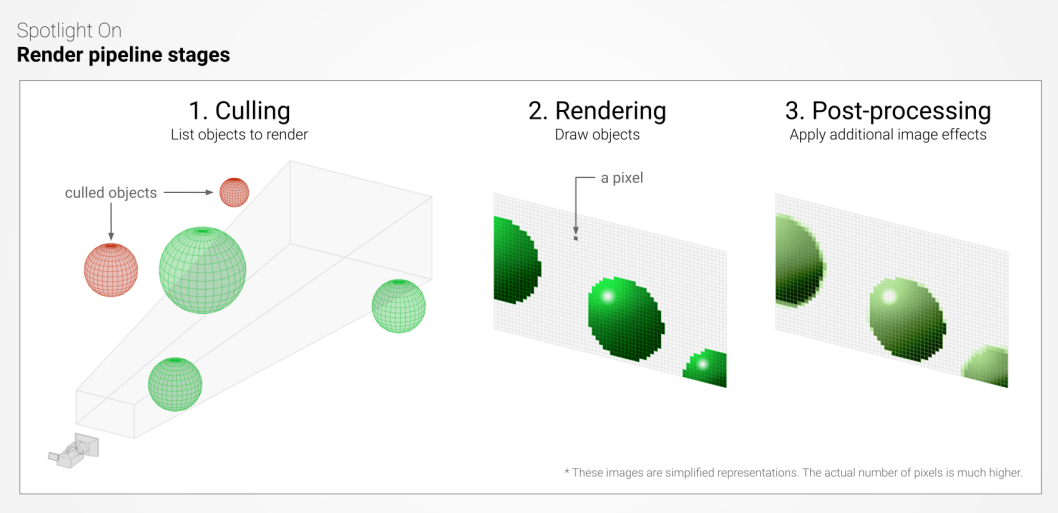
Později se v render pipeline provádí takzvaný Z-test, který se stará o odstranění překrytých objektů. Geometrie objektů musí být tvořena trojúhelníky. Pro každý pixel na obrazovce, který je zakryt nějakým trojúhelníkem, se v Z-bufferu uchová vzdálenost od kamery k nejbližšímu trojúhelníku, a jsou tak odstraněny všechny trojúhelníky, které jsou od kamery dále, než nějaký jiný trojúhelník ve stejném pixelu. Tím se elegantně a rychle vyřeší odstranění překrytých objektů v pohledovém kuželu.

2.1.2 Rendering

Po cullingu se na grafickou kartu odešlou informace, které jsou zbylými objekty referencovány, tedy jejich geometrie, textury a shadery. Tyto informace často zůstávají neměnné mezi snímky, a tak se posílají na grafickou kartu pouze jednou při prvním vykreslování objektu. Procesor si pak nechává jen odkazy, aby mohl grafické kartě říci ve chvíli,kdy má tato data použít. Než se odešlou i informace specifické pro daný snímek, procesor ještě seřadí objekty a určí tak jejich pořadí při vykreslování. V určování pořadí je důležitá vzdálenost od kamery a zda je či není objekt průhledný. Neprůhledné objekty jsou řazeny od nejbližších po nejvzdálenější, průhledné objekty od nejvzdálenějších po nejbližší. Na grafickou kartu se pak pošle zbytek informací a začne se vykreslovat. Na grafické kartě prochází každý objekt skrze vertex shader, rasterizaci a fragment shader

2.1.3 Post-processing

Pokud je zapnutý post-processing, tak se místo na obrazovku vykresluje nejdříve do bufferu na grafické kartě. Pak se použijí shadery pro aplikování filtrů post-processingu. Některé filtry potřebují několik průchodů. Post-processing dovoluje filtrům přístup k celému vykreslenému obrazu.[4]



Obrázek 2.1. Jednotlivé vykreslovací kroky [1]

2.2 Single pass a Multi pass rendering

Při použití brýlí pro virtuální realitu, které budu používat při testování v poslední kapitole této práce, rozlišujeme v Unity mezi různými sekvencemi vykreslování objektů z důvodu potřeby vykreslení obrazu dvakrát (pro každé oko). Multi pass rendering vykreslí obrazy postupně, nejdříve celé jedno oko, potom celé druhé oko. Oproti tomu single pass rendering vykresluje oba obrazy naráz, objekt po objektu. Toto je výhodné z hlediska výkonu. Jelikož obě obrazovky mají stejný culling a stíny, stačí tyto dvě věci udělat jen jednou a aplikovat je na obě obrazovky.[5] Třetí varianta je single pass instanced rendering, který je podobný standardnímu single pass renderingu. Rozdíl je ve volání funkce vykreslení.[6] Pro scénu s dvěma objekty (dvěma mesh) je rozdíl ve volání funkce následující:

Single pass

- Vykresli objekt 1 pro levé oko
- Vykresli objekt 1 pro pravé oko
- Vykresli objekt 2 pro levé oko
- Vykresli objekt 2 pro pravé oko

Single pass instanced

- Vykresli objekt 1 dvakrát, pro levé i pravé oko
- Vykresli objekt 2 dvakrát, pro levé i pravé oko

Tato změna sníží zatížení procesoru.

[1] `https://docs.unity3d.com/es/2019.4/Manual/BestPracticeLightingPipelines.html`

2.3 Forward vs. deferred rendering

Ve vykreslování používáme dvě vykreslovací posloupnosti. Forward rendering a Deferred rendering. Forward rendering je standardní technika vykreslování, která se dnes používá více než Deferred rendering.

Rozdíl je v čase aplikování světla na scénu. Forward renderer aplikuje světla standardně ve fragment shaderu . Oproti tomu deferred renderer je aplikuje až trochu později, po Z-testu a dalších funkcích, když už je geometrie připravená na vykreslení. Díky tomu je deferred rendering výkonnější, pokud má vykreslovaná scéna mnoho světelných zdrojů. A právě v těchto scénách se používá.[7]

2.4 Unity Render pipeline

Unity poskytuje Build-in render pipeline a možnost vytvořit si vlastní render pipeline prostřednictvím Scriptable render pipeline API. Tímto způsobem jsou také vytvořeny dvě další render pipeline, které Unity poskytuje.

2.4.1 Build-in render pipeline

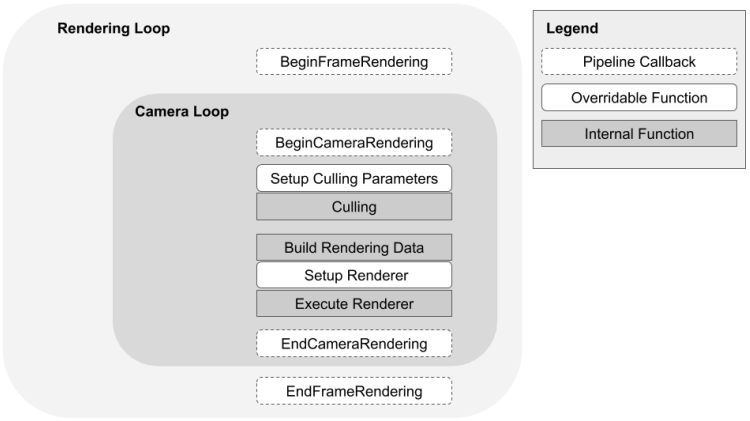
Build-in render pipeline je základní render pipeline v Unity. Je automaticky importována do projektu, pokud není zvoleno jinak. Poskytuje funkce pro vykreslování. Je možné ji modifikovat pro další funkce. Obsahuje forward render i deferred renderer. [8] BIRP je takzvaný general purpose renderer. Byla tedy vytvořena tak, aby se hodila na co nejvíce případů. S tím je však také spojen problém. Jelikož se snaží udělat vše najednou, nemůže nabídnout ty nejlepší grafické vlastnosti a nemůže být optimalizována pro potřeby každého jednoho projektu.

2.4.2 Scriptable render pipeline

Aby Unity vyřešilo problémy BIRP, udělalo nový renderer SRP. Je to modulární render systém, který dovoluje upravit vykreslovací proces pomocí skriptů. Zároveň to znamená, že Unity má více grafických možností, jako například používání shader grafu zmíněného v páté kapitole. Úprava SRP je však složitá a náročná na programování, proto Unity nabízí dvě předefinované Scriptable render pipeline nazvané Universal render pipeline a High definition render pipeline, které nabízí mnoho funkcí používaných při vývoji.

2.4.3 Universal render pipeline

URP je, jak její název napovídá, universální. Používá se pro vývoj aplikací ve 2D i ve 3D. Je méně výpočetně náročná a proto se používá i pro vývoj aplikací na zařízení s menší výpočetní kapacitou jako například mobilní telefony. Zároveň je více doporučována pro virtuální realitu než druhá předefinovaná SRP. Poskytuje forward renderer, deferred renderer a 2D renderer.



Obrázek 2.2. Universal Render Pipeline - zjednodušený vykreslovací cyklus

Obrázek 2.2 ukazuje, jak URP vykresluje. Můžeme vidět, že postupuje podle výše popsaného modelu vykreslování. URP nenabízí nutně více funkcí než BIRP, proto je používána na jednodušší aplikace. Jednotlivý výčet funkcí a porovnání s BIRP najdete v Unity URP dokumentaci[1].[9]

2.4.4 High definition render pipeline

HDRP je určená pro špičkové platformy jako například PC, PlayStation nebo Xbox. Dokáže vytvořit velmi věrohodnou grafiku a používá se pro velké hry a aplikace, kde je věrohodnost důležitá. Obecně je složitější na použití a nevhodná pro vývojáře, kteří nepracují ve větších skupinách. Nabízí stejné funkce jako URP a několik navíc. Narozdíl od URP podporuje ambient occlusion, screen space reflections (o kterých se píše v následující kapitole), decals, více možností v shaderech, auto exposure post processing efekt a ray-tracing. Ray-tracing je popsán v následující kapitole i se všemi dostupnými efekty. Je dostupný forward i deferred renderer, ale neposkytuje možnost vývoje 2D aplikací.[10]

[1] `https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@13.1/manual/universalrp-builtin-feature-comparison.html`

Kapitola 3
Ray-tracing

Ray-tracing je vykreslovací technika, která dokáže vyprodukovat velmi vysoký stupeň věrohodnosti, je však výpočetně náročná. Proto se ray-tracing používá více například pro filmy a počítačově generované obrázky, kde není potřeba vykreslovat v reálném čase. Pro filmy se začal používat v roce 2013. Až v roce 2019 se začaly prodávat uživatelům grafické karty schopné ray-tracingu v reálném čase. Nutno podotknout, že i když jsou toho dnešní grafické karty schopné, většinou se ray-tracing používá jen pro některé části vykreslování a o většinu se stále stará rastrovaná grafika.

Ray-tracing dokáže věrohodně simulovat množství optických efektů, jako například odraz, lom a rozptyl světla. Algoritmus (obrázek 2.1) vysílá z kamery paprsek/paprsky pro každý pixel na virtuální obrazovce. Tyto paprsky při každém nárazu vyšlou z místa nárazu další paprsky různými směry podle materiálu objektu, na který dopadly. Každý paprsek si drží svoji barvu, která se po výpočtu promítne do vykreslení.[11]

3.1 Vykreslovací rovnice

$$L_o(X, \widehat{\omega}_o) = L_e(X, \widehat{\omega}_o) + \int_{S^2} L_i(X, \widehat{\omega}_i) f_X(\widehat{\omega}_i, \widehat{\omega}_o) |\omega_i \cdot \widehat{n}| d\widehat{\omega}_i$$

- X - bod ve scéně
- $\widehat{\omega}_o$ - odchozí směr
- $\widehat{\omega}_i$ - příchozí směr
- $\widehat{n}$ - povrchová norma
- S^2 - všechny příchozí směry (koule)

Tato rovnice je základem pro ray-tracing. Popisuje, jak se má chovat paprsek v daném bodě. Popíšeme-li rovnici slovy, říká, že světlo $L_o(X, \widehat{\omega}_o)$ vyzařované bodem X se rovná světlu $L_e(X, \widehat{\omega}_e)$, které tento bod vysílá (pro světelné zdroje) a pro všechny příchozí směry $\int_{S^2} d\widehat{\omega}_i$ je potřeba přičíst množství příchozího světla $L_i(X, \widehat{\omega}_i)$ vynásobené funkcí odraznosti materiálu $f_X(\widehat{\omega}_i, \widehat{\omega}_o)$, jež říká, jak moc se příchozí světlo bude odrážet, a Lambertovým zákonem $|\omega_i \cdot n|$. Lambertův zákon se týká vlivu dopadajícího světla na povrch v závislosti na úhlu dopadu. Čím kolměji světlo dopadá, tím větší efekt má na povrch. Rovnice je rekurzivní.

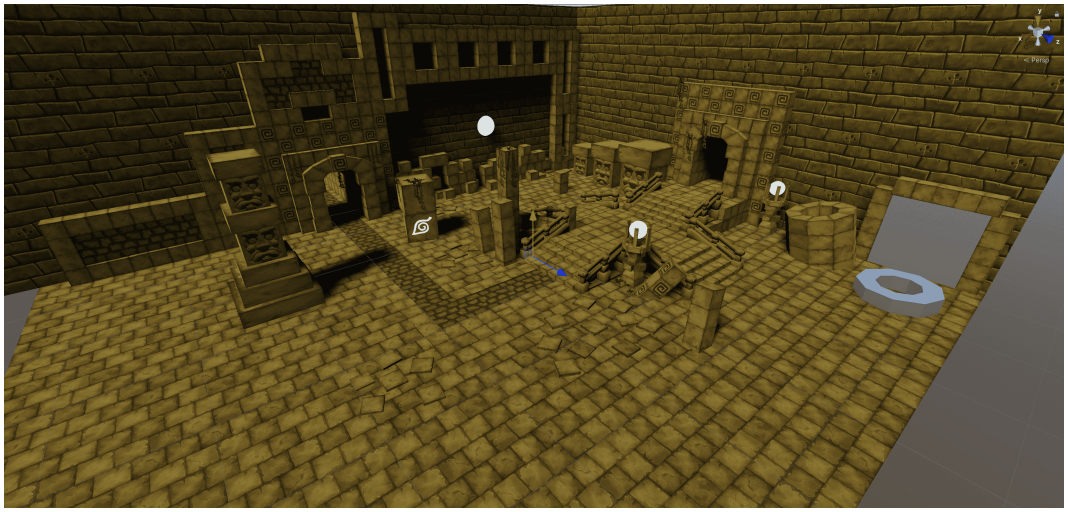
Tato forma rovnice se nazývá čisté sledování paprsku (pure path tracing). Paprsek vystřelený do scény se protne s nějakým objektem v nějakém bodě. Pro tento bod se spočítá rovnice a vystřelí se odtud další paprsek náhodným směrem. Proces není ukončen, dokud paprsek neprotne zdroj světla. To však může být velmi výpočetně náročné, proto dnešní metody ovlivňují směr vyslaného paprsku a v případě ray-tracingu vysílají z každého bodu více paprsků najednou.

Path tracing se dnes používá jako vzor. Nechá se vykreslit scéna skrze path tracing s velkým množstvím vystřelených paprsků pro každý pixel (v řádu tisíců), a pak se podle tohoto vzoru upravuje tak, aby scéna vypadala podobně, i když se vykresluje pomocí méně náročných metod.

Ray-tracing je podrobněji popsaný v knize Ray Tracing Gems od Apress[12]

3.2 Ray-tracing v Unity

HDRP v Unity podporuje ray-tracing několika částí vykreslování. K hlavním patří zastínění okolím (ambient occlusion), kontaktní stíny (contact-shadow), globální osvětlení (global ilumination), odrazy (reflections) a stíny (shadows). Tyto funkce lze zapnout skrze již zmíněný volume profil, či v případě stínů se zapínají přímo na světlech samotných. Níže budu demostrovat jednotlivé efekty ray-tracingu na scéně na obrázku 3.1.



Obrázek 3.1. Scéna pro demonstraci efektů ray-tracingu.

3.2.1 Možnosti nastavení

Unity umožňuje nastavit několik parametrů při používání ray-tracingu. Hlavním parametrem, který jde nastavit u všech níže popsaných metod, je počet vzorků (sample), tedy kolikrát se paprsek vystřelí do scény pro každý pixel. Počet vzorků je obecně potřeba větší v místech s nižší úrovní osvětlení, na místech s přímým (slunečním) světlem běžný uživatel nepozná rozdíl, pokud použijeme jen jeden vzorek. U většiny metod pak najdeme číslo určující maximální počet odrazů každého paprsku a denoiser.

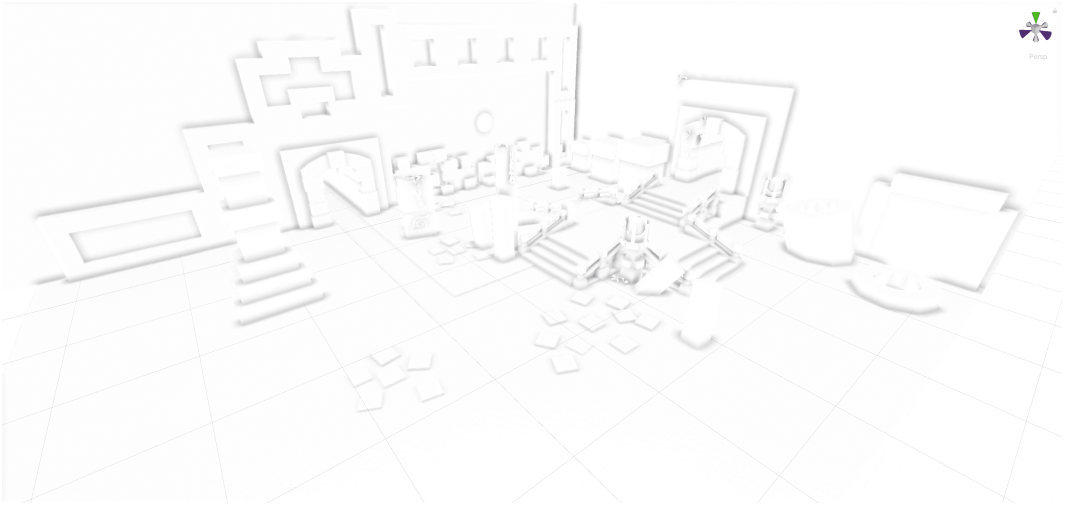
Množství odrazů paprsků je důležité u Global illumination a u reflektivních efektů ray-tracingu, které jsou popsány níže.

Denoiser je schopen vyplnit mezery v zrnitém obrazu a vytvořit tak lepší výsledek s malým počtem vzorků. Díky tomu dokážeme mnohem výkonněji používat ray-tracing v reálném čase.

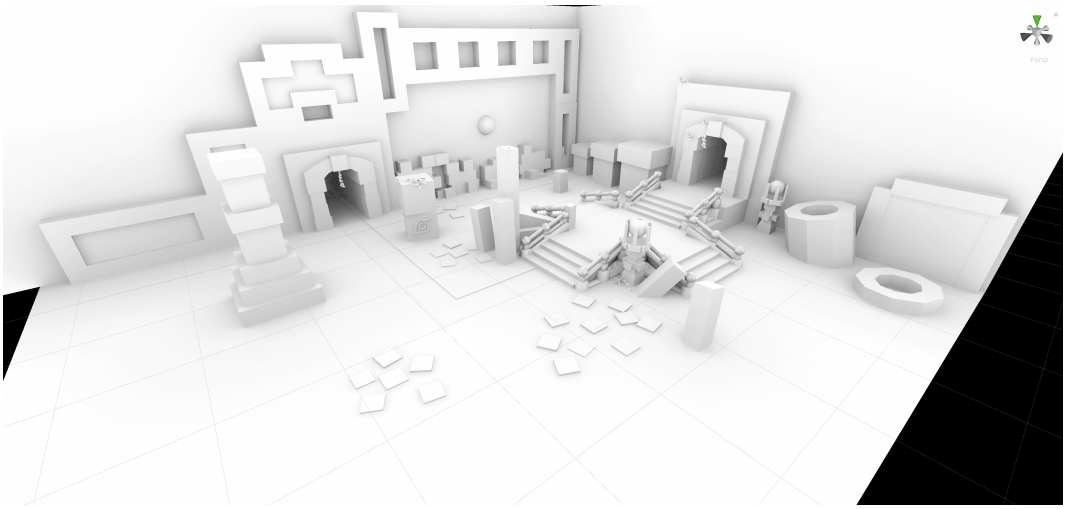
3.2.2 Ambientní okluze

AO je globální metoda stínování. Pomáhá scéně získat realistický dojem tím, že napodobuje chování reálného světla a započítává tlumení světla zastíněním. Pomáhá nám lépe rozpoznat tvar objektů. Například screen space ambient occlusion (obrázek 3.2) je často používaná pro její nenáročnost ve výpočtech a účinnost. Obvykle se určuje vysláním paprsků z modelu. Sleduje, zda paprsek narazí do jiného objektu, či nenarazí (omezujeme maximální délku paprsku). Pokud paprsek nenarazí, pak bude bod světlejší. [13]

Pro AO je možné použít ray-tracing pro větší věrohodnost, jak je ukázáno na obrázku 3.3. Ray-traced ambient occlusion dokáže lépe simulovat realitu pomocí započítávání geometrie scény a pozice kamery. Také dokáže, na rozdíl od screen space ambient occlusion, zahrnout do výpočtu i objekty, které jsou mimo obrazovku. [14]



Obrázek 3.2. Čisté Screen space ambient occlusion



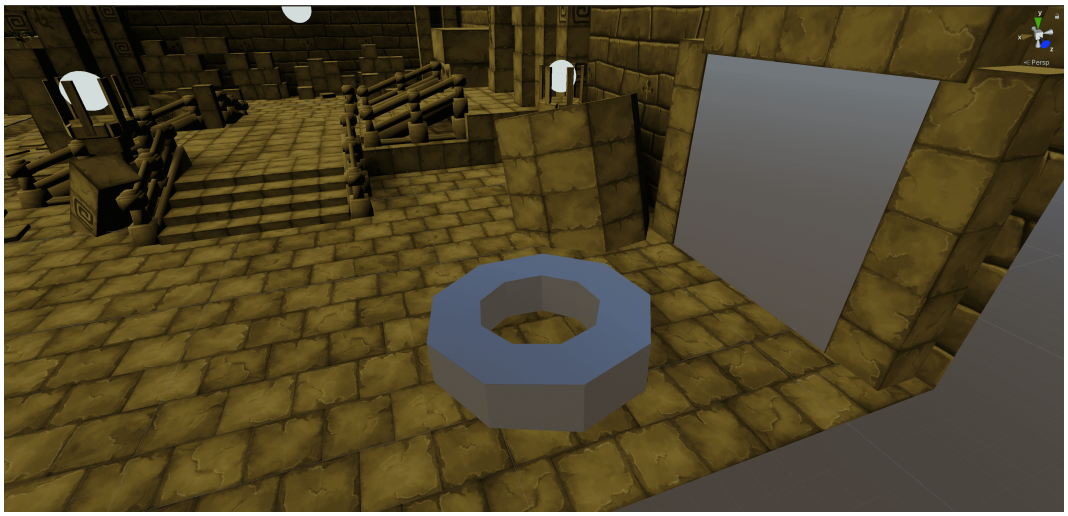
Obrázek 3.3. Čisté Ray-traced ambient occlusion

3.2.3 Odrazy

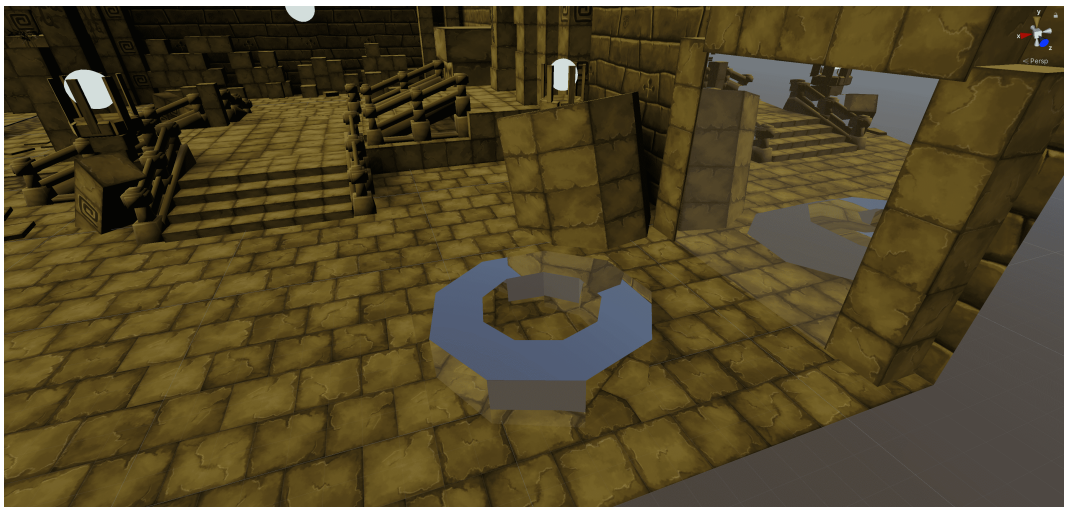
V Unity se používají reflection probes nebo screen space reflection pro simulaci odrazu od objektů s reflektivním materiálem. Obě metody mají své nedostatky. Reflection probes se používají pro odrazy v blízkém okolí a jsou primárně určeny pro statické objekty s potřebou je znovu `vypéct` (vypočítat) při změně okolí. Také je jich často potřeba více, je potřeba čas pro jejich nastavení a nejsou věrohodné pro objekty podobné zrcadlu (velké a velmi reflektivní). Screen space reflection je určené pro globální odrazy. Nedokáže zachytit odrazy objektů, které jsou mimo obrazovku a odrazy občas nemají vysokou věrohodnost (obrázek 3.5, je vidět, že objekty, které nejsou na obrazovce, nejsou ani v odrazu). Je to však relativně levná metoda při porovnání s ray-traced reflection. Je ideální pro simulaci například odrazů v kalužích a dalších podobných objektech, kde odraz neočekáváme příliš věrohodný.

Ray-traced reflection přidává odrazům věrohodnost a pro vykreslení odrazu není nutné, aby byl daný objekt v obrazu (obrázek 2.5, odraz je plný, i když některé části objektů nejsou v obraze). Unity přidává množství nastavení pro ray-traced reflection. [15]

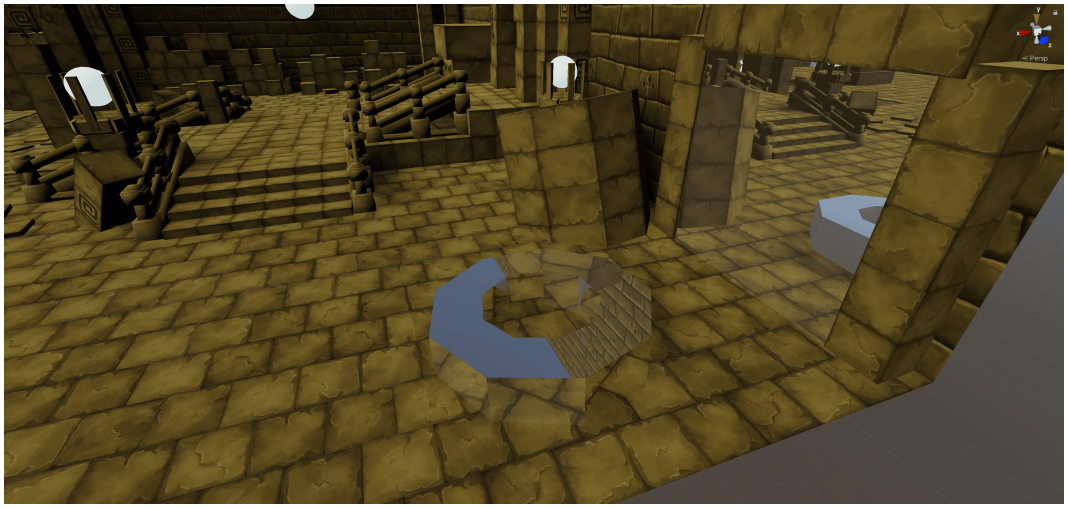
Na obrázcích níže jsou zobrazeny jednotlivé efekty vyjmenované výše. Z obrázku 3.5 je vidět, že screen space reflection nedokáže zobrazit prostor zastíněný objektem tvaru O v zrcadle napravo, a namísto toho zobrazuje oblohu. Obrázky 3.6 a 3.7 se liší pouze nastavením maximálního počtu odrazů paprsku. Při nastavení dvou odrazů má objekt tvaru O svůj vlastní odraz i v zrcadle (3.7).



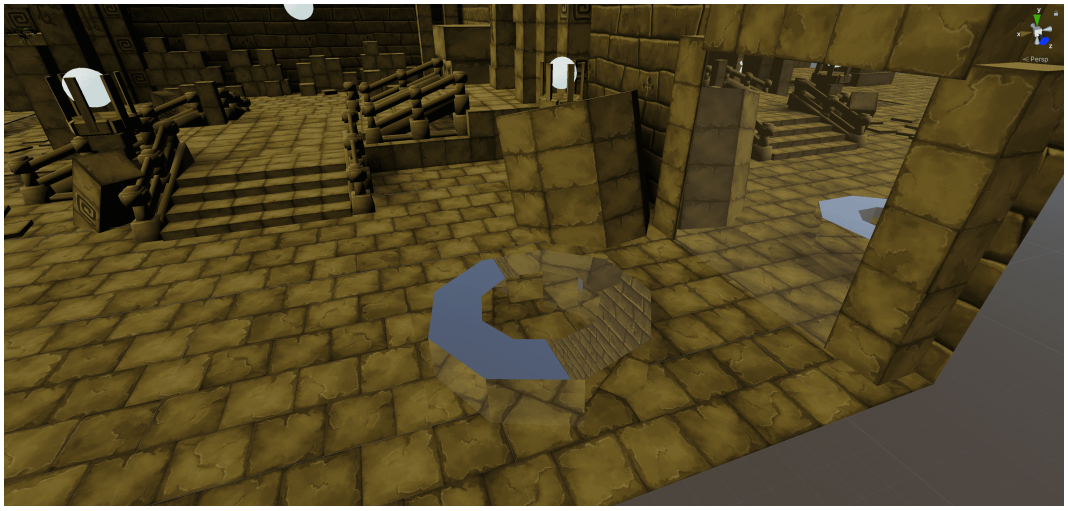
Obrázek 3.4. Pohled do scény pro demonstraci odrazů bez efektů



Obrázek 3.5. Screen space reflection



Obrázek 3.6. Ray-traced reflection s jedním odrazem

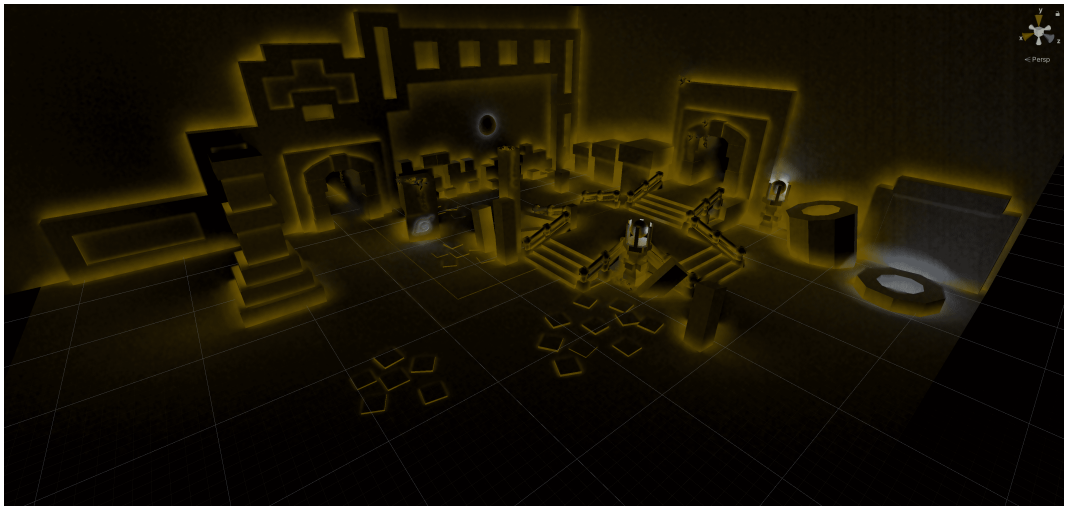


Obrázek 3.7. Ray-traced reflection s oběma odrazy

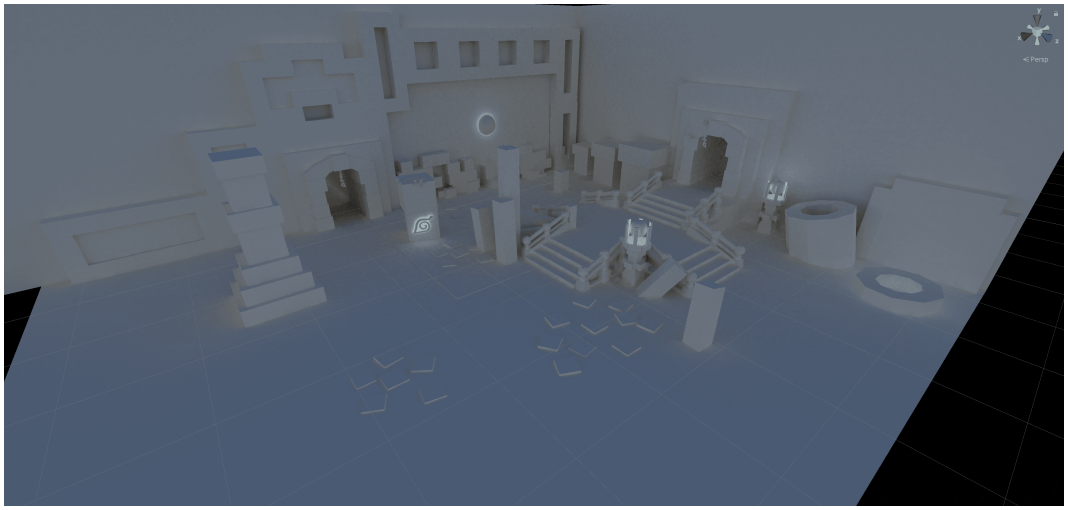
3.2.4 Globální iluminace

Global illumination se stará o nepřímé osvětlení ve scéně. Z osvětlených míst vyšle slabé paprsky do přilehlých míst. Toto světlo s sebou nese i barvu objektu, od kterého se odráží (osvícený červený objekt na svoje okolí přenáší načervenalé světlo). Global illumination je velice výpočetně náročná. V Unity se nejčastěji používá předem vypočítaná (takzvaně vypečená) lightmapa pro statické objekty. Pro dynamické objekty je možnost použít real-time global illumination, ale náročnost výpočtů je vysoká, často tedy na dynamické objekty v real-time aplikacích global illumination neaplikujeme a důsledkem může být, že tyto objekty pak do scény nezapadají.

Ray-traced global illumination je velice náročné na výpočet a pro statické objekty je věrohodnější a výhodnější předem vypočítat lightmapu. Ray-traced global illumination však může být výhodné pro místo s velkým množstvím dynamických objektů. Pomocí lokálního volume profilu lze například nastavit, aby se ray-traced global illumination zapnulo jen v místě, kde je potřeba, a lze tak ušetřit výkon. [16]



Obrázek 3.8. Screen space global illumination



Obrázek 3.9. Ray-traced global illumination

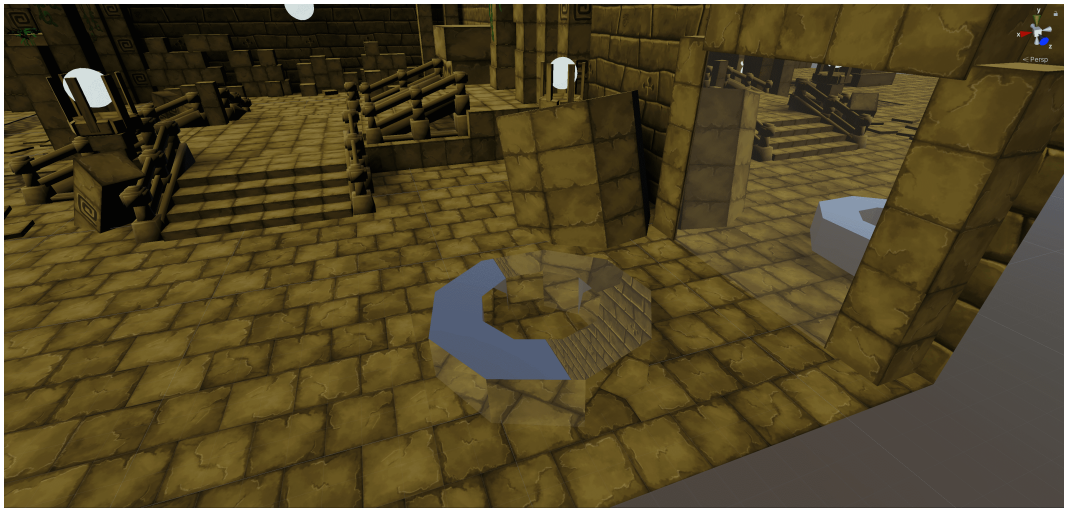
3.2.5 Stíny

Každý zdroj světla může vrhat stíny jednoho objektu na druhý. Přidávají tak scéně hloubku. K tomu se používají Stínové mapy (Shadow map). Stínová mapa je vlastně textura toho, co vidí světlo. Zdroj světla může mít více stínových map, například bodový zdroj světla potřebuje šest map (celou cube mapu).

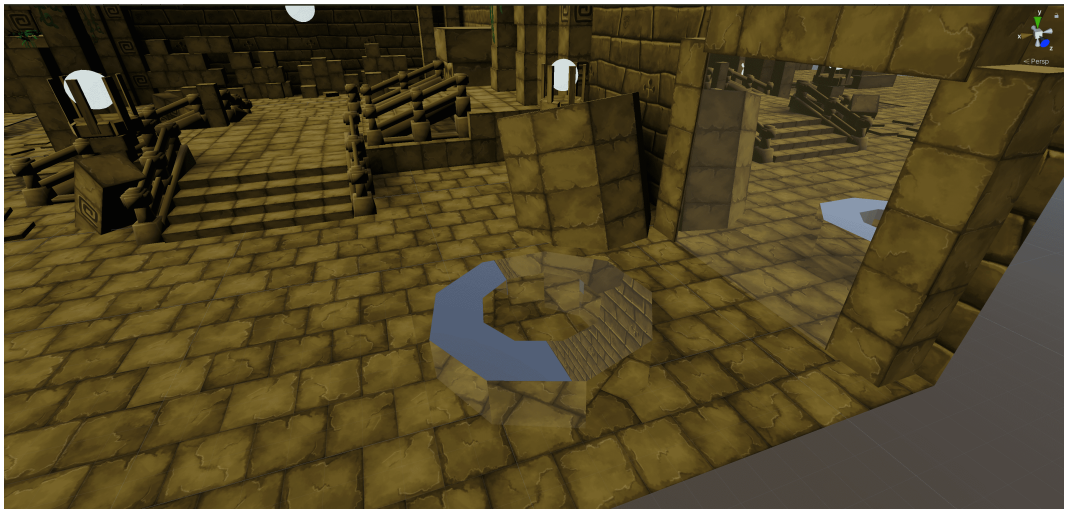
Při aktivaci ray-traced stínů se přestanou používat stínové mapy. Ray-traced stíny umí pracovat i s průhlednými objekty a vrhat věrohodnou intenzitu stínu, na rozdíl od jiných metod, které vrhají stín se stejnou intenzitou pro neprůhledné i průhledné objekty. Unity dokonce poskytuje možnost vrhat barevné stíny, pokud světlo prochází skrz barevný průhledný objekt. Zároveň dokáže dobře simulovat velikost zdrojů světla, díky tomu zanechává hrany stínů méně ostré stejně, jako je to ve skutečnosti. [17] Kontaktní stíny (Contact shadows) se používají pro vykreslování malých detailů, které normální stínové mapy nedokáží zobrazit kvůli omezenému rozlišení. Unity poskytuje možnost pro RT kontaktní stíny, které mají vyšší věrohodnost.

3.2.6 Recursive rendering

Recursive rendering je alternativa k vykreslování objektů v HDRP. Používá se pro reflektivní a průhledné materiály. Objekty s materiálem, který tento vykreslovací způsob používá, vysílají paprsky na okolní objekty a dostávají tak svůj vzhled. Recursive rendering je užitečný pro vykreslování průhledných objektů s několika vrstvami a je to jednoduchý způsob, jak dát odraz průhledným materiálům. Je možné jej použít i pro jednoduchá zrcadla, ale z výkonnostního hlediska je na zrcadlo lepší použít Ray-traced reflection.[18]



Obrázek 3.10. Recursive rendering s maximální hloubkou 2



Obrázek 3.11. Recursive rendering s maximální hloubkou 3

3.2.7 Omezení a požadavky

Hardware

Unity v současné době podporuje ray-tracing na následujících grafických kartách

- NVIDIA GeForce 20 Series
- NVIDIA GeForce 30 Series
- NVIDIA Quadro (3000 a vyšší)

Setup

Pro nastavení podpory ray-tracingu v projektu lze použít HDRP Wizard popsaný níže v sekci 5.2. Je nutné upgradovat projekt na DirectX 12 API z DirectX 11, žádný jiný API zatím ray-tracing nepodporuje.

Omezení

HDRP Ray-tracing render pipeline nepodporuje některé efekty, které HDRP poskytuje v rasterization render pipeline. Patří mezi ně například tesselace, decals a vertex animace. Jejich celý seznam lze nalézt zde [19](HDRP dokumentace).

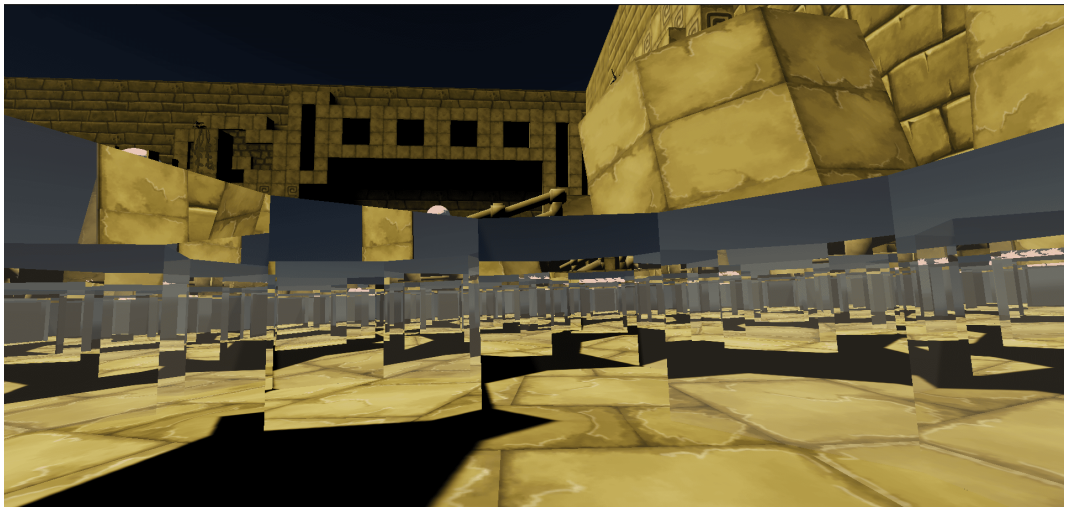
Kapitola 4
Výkonnostní testování - Ray-tracing

4.1 Specifikace testování

V předchozí kapitole jsem popsal jednotlivé efekty ray-tracingu a demonstroval, jak tyto efekty vypadají (často v porovnání s jejich alternativami). V této kapitole se zaměřím na ray-tracing z hlediska výkonu. Provedeno bylo několik testů pro každý z výše popsaných efektů, které se lišily různým počtem vzorků a odrazů, maximální délkou paprsků, množstvím vykreslované geometrie a pozicí pohledu do scény.

Testování bylo prováděno přímo v Unity editoru. Editor se vždy snaží vykreslit co nejvíce snímků za sekundu, a tak více vytěžuje grafickou kartu, na rozdíl od samostatné aplikace, kde je nastaven limit pro maximální počet snímků za sekundu a výsledky by tak byly zkreslené. Dalším důvodem je jednoduchost přepínání jednotlivých efektů a parametrů oproti samostatné aplikaci.

Obrázek 4.1. Kamera 1



Obrázek 4.2. Kamera 6 - reflektivní ray-tracing

Testy byly prováděny na zařízení s těmito specifikacemi:

- CPU Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz
- GPU NVIDIA GeForce RTX 2080 Ti
- 128 GB RAM

4.2 Vyhodnocení testování

V přiložené tabulce 4.1 jsou pouze dvě kamery z šesti testovaných. První čtyři kamery zahrnovaly celou scénu, dvě poslední jsou určeny pro testování reflektivních efektů, jelikož z hlediska výkonu měly reflektivní efekty minimální zatížení na první čtyři kamery. Ve scéně se také nachází několik modelů objektu zvaného Stanford dragon[1] za účelem vyšší náročnosti na vykreslení scény. Délka paprsku byla nastavena na hodnotu, kterou by uživatel zhruba použil ve svém projektu pro daný efekt. Měřeny pak byly údaje zatížení grafické karty a počet snímků za sekundu pro různé efekty s různým počtem vzorků a odrazů paprsku.

Porovnání náročnosti

Pro všechny testy, kde bylo možné toto měřit, platí, že přidávání počtu vzorků má na náročnosti vliv menší, než počet odrazů. Také platí, že navýšení počtu odrazů zvyšuje náročnost v závislosti na počet vzorků. To vyplývá z informací v sekci 3.1.

Testováno bylo pět efektů ray-tracingu. Dle výsledků je nejnáročnějším efektem ray-traced global illumination, který už s jedním vzorkem a jedním odrazem klesl na 110 FPS při téměř maximálním zatížení grafické karty.

Ray-traced ambient occlusion a ray-traced shadows mají velice podobnou náročnost na testované scéně.

Ray-traced reflections a Recursive rendering jsem porovnával s jiným umístěním kamer ve scéně, než ostatní tři efekty, neboť jejich zatížení je přímo závislé na množství reflektivních ploch v obrazu kamery. Z výsledků je zřejmé, že použití RTR je lepší pro jednoduché zrcadlo, ale pokud je potřeba větší množství odrazů, RR postupně překoná RTR.

Zde jsou některé vybrané záznamy z testování:

[1] Stanford dragon: `http://graphics.stanford.edu/pub/3Dscanrep/dragon/dragon_recon.tar.gz`

Feature	Samples	Bounces	Max ray length	Camera	FPS	GPU
default	-	-	-	1	180	98
RTAO	1	-	0.5	1	150	96
RTAO	2	-	3	1	150	96
RTAO	8	-	20	1	125	98
RTGI	1	1	50	1	110	98
RTGI	4	1	50	1	67	99
RTGI	2	2	50	1	53	99
RTGI	4	2	50	1	21	100
RTR	1	1	50	6	150	47
RTR	1	4	50	6	114	98
RR	-	1	100	6	160	58
RR	-	4	100	6	160	76
RTS	1	-	-	1	160	87
RTS	2	-	-	1	150	90
RTS	8	-	-	1	120	96

Tabulka 4.1. Vybrané záznamy testů výkonu ray-tracingu

Kapitola 5
Převod z URP do HDRP - Analýza

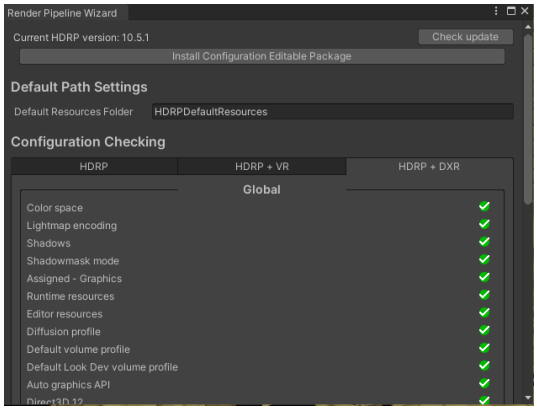
Jedním z cílů této práce je navrhnout proces konverze projektu ve vývoji z URP do HDRP a zjednodušit ho. Projekty se mohou nacházet v různých fázích vývoje. Obecně jsou komplikace v částech, které tyto dvě pipeliny poskytují každá trochu jinak. Problém lze rozdělit na několik částí. Největším problémem je převod materiálů mezi shadery. Další jsou například rozdílná reprezentace světel a více funkcí post-processingu.

5.1 Package manager

Pro převod projektu budeme potřebovat nejdříve importovat HDRP, projekt převést a pak odebrat URP. K tomu poslouží Unity package manager. Ten obecně slouží ke stahování balíčků, které unity poskytuje nebo nabízí ve svém asset store do projektu. Najdeme ho v Unity editoru v záložce Window→Package Manager.

5.2 HDRP Wizard

HDRP Wizard je `podprogram` přidaný HDRP pipeline, který dokáže automaticky konfigurovat nový projekt, aby byla HDRP plně funkční. Ve třech záložkách může uživatel pomocí tlačítek fix/fix all zprovoznit základní HDRP, HDRP s podporou VR a HDRP s podporou ray-tracingu (obrázek 5.1). Je proto užitečný jak při zakládání nového HDRP projektu, tak pro konverzi z jiné pipeline. Mohu ho tedy využít pro převedení projektu z URP do HDRP a odpadnou tak zdlouhavá nastavování profilů, kódování, stínů, atd.



Obrázek 5.1. High definition render pipeline Wizard

5.3 Materiály a shadery

Materiály v Unity definují, jak je objekt, na který je materiál připojen, ovlivňován světlem. Pro jeho definování používáme shadery. URP i HDRP poskytují každá svou vlastní

implementaci těchto shaderů (a to více než jednu). Obecně jsou materiály na uživatelské úrovni reprezentovány pomocí takzvaných properties, které uchovávají informace o jejich nastavení (barvy, textury, atd.). K těm uživatel přistupuje skrze Unity editor či v kódu přes skripty.

Pro převod projektu je tedy potřeba převést všechny materiály z URP shaderů do jejich HDRP protějšku. Počet materiálů je závislý na pokročilosti projektu a může dosahovat i několika stovek. Unity v současné době poskytuje převod materiálů pouze z Build-in RP do URP i HDRP. Neposkytuje převod přímo mezi URP a HDRP a ani zpětný převod z URP/HDRP do Build-in RP.

5.3.1 Shader graph

Shader graph nabízí možnost udělat si vlastní shader vizuálně přímo v Unity editoru. Místo psaní kódu je možnost vytvořit a propojovat uzly do grafového rámce. Od Unity verzí 2021 byl nahrazen jádrový uzel novějším, který je schopnen tvořit shader graphy pro URP a HDRP zároveň. Proto v této práci nebudu řešit konverzi shader graphů.

5.4 Světla

V počítačové grafice používáme několik typů světla. Jsou jimi directional light (většinou reprezentuje slunce), point light(žárovka), spot light (baterka) a area light. V unity mají všechny tyto typy podobnou reprezentaci. Reprezentace se však liší mezi URP a HDRP, protože HDRP používá svůj vlastní způsob. Přidává teplotu světla, mění jednotku intenzity a několik dalších menších změn.

HDRP také přidává intenzitu světla k zářivým (emission) materiálům.

5.5 Volume profile

Volume profily používají stejný základ. To znamená, že nastavení volume profilů a volume overrides projektu v URP zůstane i po převodu, ale HDRP přidává několik dalších volume overrides, které mohou být použity. V rámci převodu projektu by tedy neměly být volume profily problém

5.6 Podobné projekty

Nenašel jsem žádnou oficiální práci, která by se tímto tématem zabývala. Uživatelé Unity však tento problém několikrát řešili na různých internetových fórech. Většinou zde není problém vyřešen a odpovědí je, že se má uživatel rozhodnout pro jednu z render pipeline už při vytváření projektu. Dále pak, že pro převedení je potřeba manuálně převést všechny materiály.

Nejobsáhlejší diskuze na toto téma, kterou jsem našel, je na Unity fóru pod otázkou uživatele jménem Hannibal Leo[1], kde je popsaný problém uživateli, kteří tento problém řešili. Je zde přidaný i Unity skript pro konvertování materiálu, kterým byla inspirována sekce v následující kapitole.

[1] `https://forum.unity.com/threads/is-switching-from-urp-to-hdrp-possible.902099/`

Kapitola 6
Proces konverze - Řešení

Proces konverze lze zjednodušit několika způsoby. Za nejdůležitější pokládám zjednodušení převodu materiálů. Dále popíšu problémy s převodem projektu, na které jsem narazil a jak se jim vyvarovat.

6.0.1 Konverze

Na začátek jsem si do URP projektu importoval HDRP pomocí package manageru. Po importu se otevřel HDRP Wizard. Důležité je dát fix all pouze v záložce HDRP, pokud kliknu na fix all v záložce HDRP+DXR, Unity vyhodí error a spadne, projekt bude koruptován a nebude možné znovu projekt spustit. HDRP Wizard vytvořil defaultní HDRP asset a několik dalších souborů a uložil je do složky HDRPDefaulResources. Nedokázal však projekt nastavit tak, aby default HDRP asset používal, protože je již přítomný URP asset. Pro nastavení HDRP asset je třeba ho navolit v Edit→Project Settings→Graphics→ScriptableRenderPipelineSettings a v Edit→Project Settings→Quality→Rendering pro všechny úrovně kvality.

Nyní zbývají světla a materiály.

6.1 Material Convertor

Převod materiálů lze implementovat přímo v Unity editoru jako editor skript, který by dokázal:

- Převést materiál s URP shaderem na materiál s HDRP shaderem se stejnými texturami, barvami a nastavením.
- Převést všechny materiály v projektu najednou.

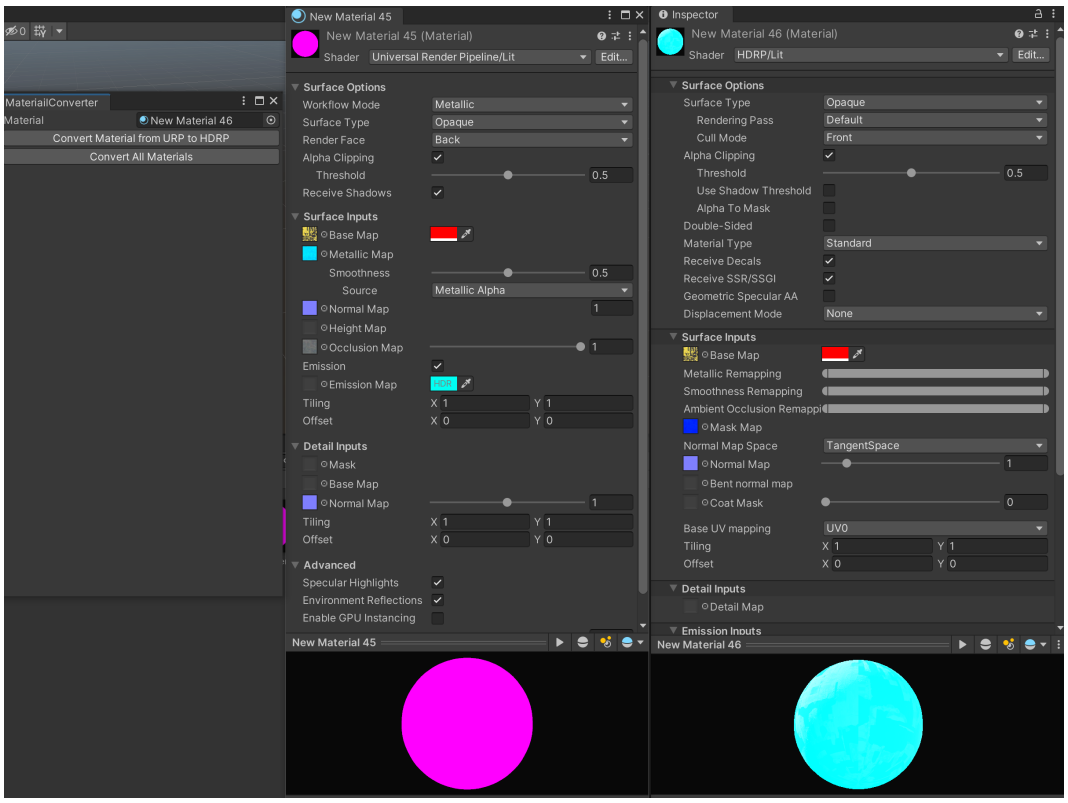
URP a HDRP implementují více shaderů a ne všechny z implementací mají svůj protějšek v druhé render pipeline. Pomocí skriptu budu převádět jen ty materiály, které mají svůj protějšek. Po prozkoumání jednotlivých shaderů a jejich vlastností jsem vybral tyto:

- URP Lit → HDRP Lit
- URP Complex Lit → HDRP Lit
- URP Simple Lit → HDRP Lit
- URP Unlit → HDRP Unlit

Vytvořil jsem tedy skript MaterialConverter (dále MC). Skript je možné otevřít přímo v Unity editoru v horním okně záložek. MC má jednoduché GUI s dvěma tlačítky a oknem pro připojení materiálu. Jedno z tlačítek převede připojený materiál, druhé všechny materiály v projektu.

Skript nejdříve načte vlastnosti (properties) materiálu v URP, pokud materiál používá jeden z shaderů vyjmenovaných výše. Změní shader materiálu na jeho protějšek a vloží načtené vlastnosti do materiálu v HDRP. V HDRP mají shadery více možností nastavení. Vlastnosti, které nejsou v URP shaderu, mají ponechanou základní hodnotu.

Pro převedení všech materiálů najednou skript načte všechny materiály v AssetDatabase a v cyklu převede jeden po druhém.



Obrázek 6.1. Material Convertor UI a materiál před konverzí a po konverzi

6.2 Mask Map

Během implementace jsem narazil na problém s texturami. HDRP používá texturu Mask Map místo čtyř různých textur. Mask Mapa je složení čtyř různých textur pomocí barevných kanálů RGBA. [20]

- R - Metallic
- G - Occlusion
- B - Detail mask
- A - Smoothness

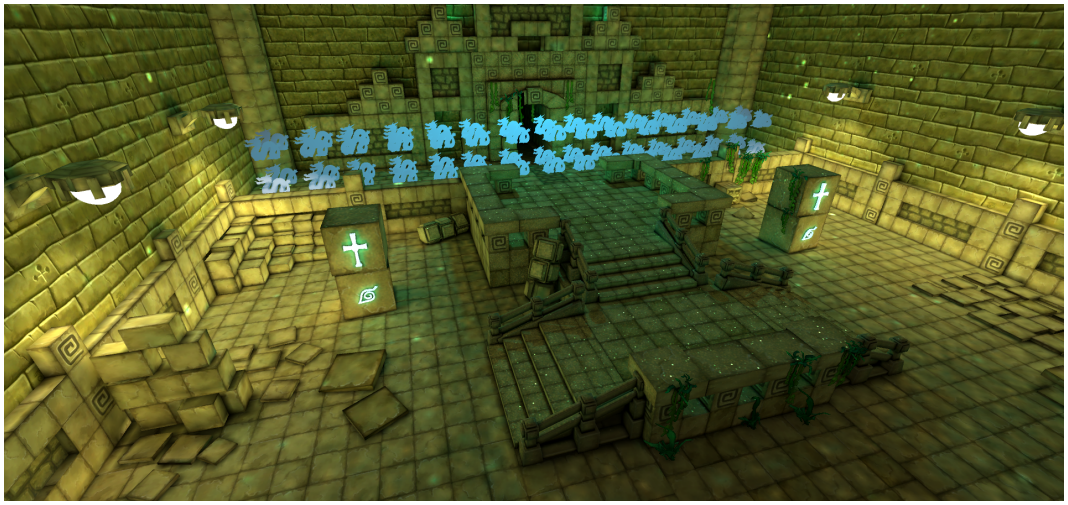
Jako řešení jsem v MC přidal funkci na vytvoření Mask mapy, pokud je přítomna alespoň jedna z těchto čtyř textur. Limitace této funkce je, že je schopná spojit textury pouze pokud mají stejný rozměr v pixelech. V opačném případě se Mask mapa nevytvoří, v konzoli editoru se vypíše chybová hláška a program pokračuje bez připojení Mask mapy na HDRP shader. Vytvořené Mask mapy se ukládají do stejné složky jako materiály, ke kterým jsou přiřazeny. Jejich název začíná názvem materiálu a končí *MaskMap*.

6.3 Světla

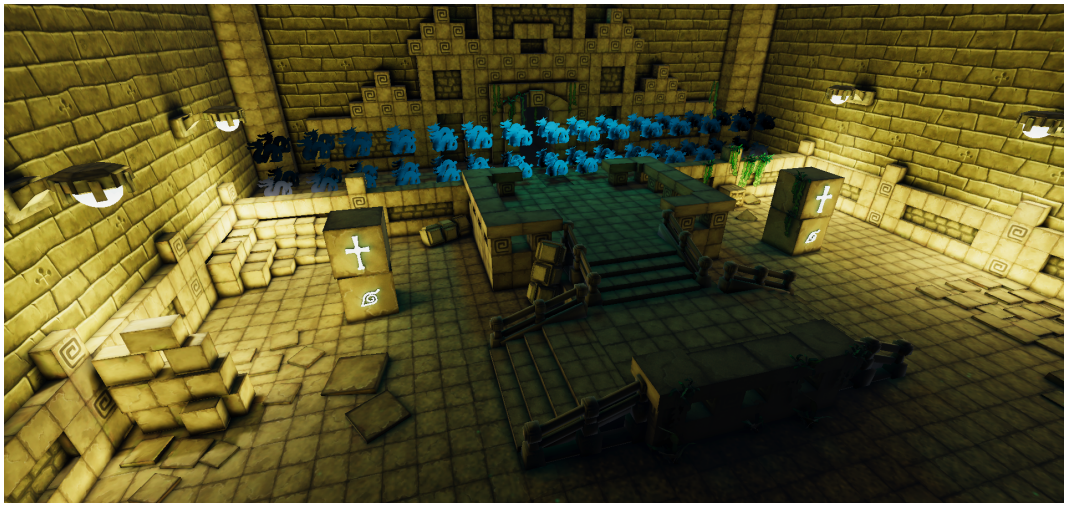
Světla ve scéně si při převodu uchovají většinu údajů. Jak ale bylo zmíněno v sekci 5.3, neuchovají si svou intenzitu a přidají se další nastavitelné parametry. Po několika

pokusech konverze jsem usoudil, že není ideálním řešením slepé přepočítání intenzity do jednotek používaných v HDRP a aplikování na jednotlivá světla, jelikož se v průběhu většinou rozhodí osvětlení celé scény a pouhé nastavení intenzity světel nestačí k tomu, aby scéna dostala svůj původní vzhled. Společně s nastavením světel jsem nastavoval některé volume overrides v defaultním volume profilu a vypočítával jsem novou lightmapu pro scénu. Na emissivních materiálech je potřeba zkontrolovat intenzitu záření. MC automaticky nastaví hodnotu 1000 Nits, která se zdá být vizuálně podobná základní hodnotě intenzity v URP. Pokud však používáte emission jako důležitou součást světelných zdrojů ve scéně, budete nejspíše muset hodnotu zvýšit.

Testovací scéna z další kapitoly před a po konverzi



Obrázek 6.2. Základní scéna - kamera 1, URP



Obrázek 6.3. Základní scéna - kamera 1, HDRP

6.4 Možné problémy konverze

V nižších verzích Unity se mi při prvních testech konverze stalo, že se projekt převedl s kompilačními chybami. Stejnou zkušenost ohlásili i další uživatelé ve fórech prozkoumaných v předchozí kapitole. V Unity verzi 2020.3.8f1 jsem na takový problém již nenarazil.

6.4.1 Material Convertor

Pokud se v projektu nachází materiály s jinými shadery, než které je MC schopný převést, materiály zůstanou nepřevedené a budou se zobrazovat s růžovou texturou, která obecně značí chybějící texturu. V takových případech nemá materiál svůj protějšek v HDRP. Další možnosti jsou převést zbytek materiálů manuálně, zkusit automatický převod z Build-in RP do HDRP nebo přepsat skript MC, aby převáděl všechny materiály do HDRP Lit. Poslední z možností nebyla v této práci vyzkoušena.

Kapitola 7
Výkonnostní testování

V této kapitole testuji scény na různém hardware a různých headsetech pro virtuální realitu z výkonnostního hlediska. Cílem tohoto testování je porovnat náročnost URP a HDRP ve stejných podmínkách, tedy v URP před konverzí projektu a v HDRP po konverzi projektu, v co možná nejpodobnějších podmínkách.

7.1 Specifikace testování

Testované byly dvě scény. Nazval jsem je Základní a Pokročilá. Základní scéna byla vytvořena v novém projektu přidáváním statických objektů s velkým množstvím geometrie, aby se zvýšila náročnost na vykreslení scény. Pokročilá scéna je v projektu v pokročilé fázi vývoje, kde velkou část výpočetní náročnosti tvoří i logika aplikace.

Stejně jako ve čtvrté kapitole byly i zde zaznamenávány snímky za sekundu a zatížení grafické karty. Navíc zde bylo zaznamenáno i zatížení procesoru, jelikož by mohlo mít dopad u Pokročilé scény, kde se může vyskytovat náročná herní logika, která je závislá na procesoru a ne na grafické kartě.

Testované PC

- PC1:
  - CPU - Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz
  - GPU - 2080
  - RAM - 128 GB
- PC2:
  - CPU - Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz
  - GPU - NVIDIA Quadro RTX 5000
  - RAM - 64.0 GB
- PC3:
  - CPU - Intel(R) Xeon(R) W-2125 CPU @ 4.00GHz
  - GPU - NVIDIATAN Xp
  - RAM - 64.0 GB

Testované VR headsety

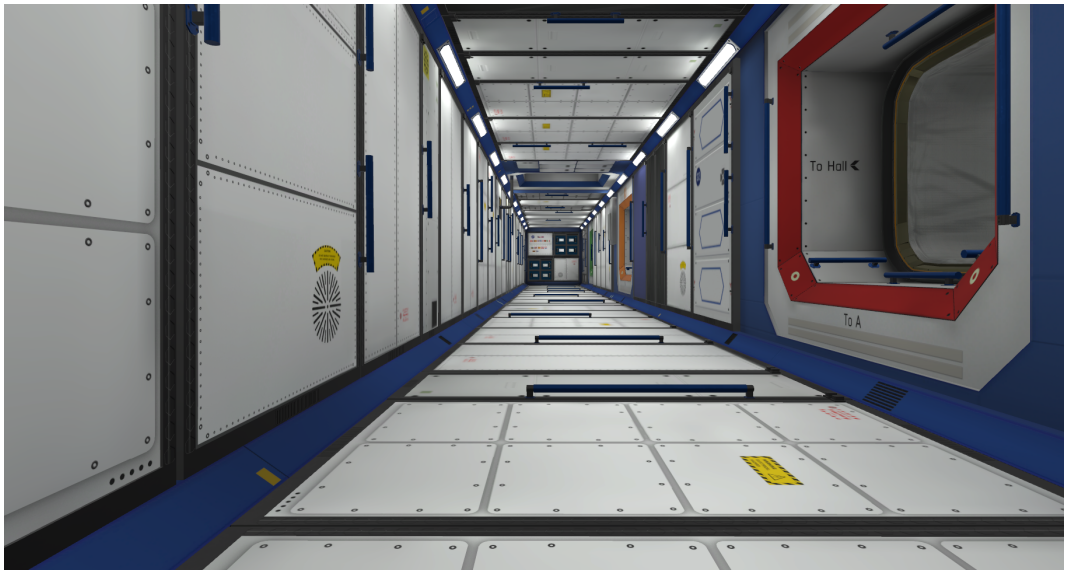
- HTC VIVE
- HTC VIVE PRO
- XTAL 8k
- Oculus Quest 2

Kamery a množství geometrie scén

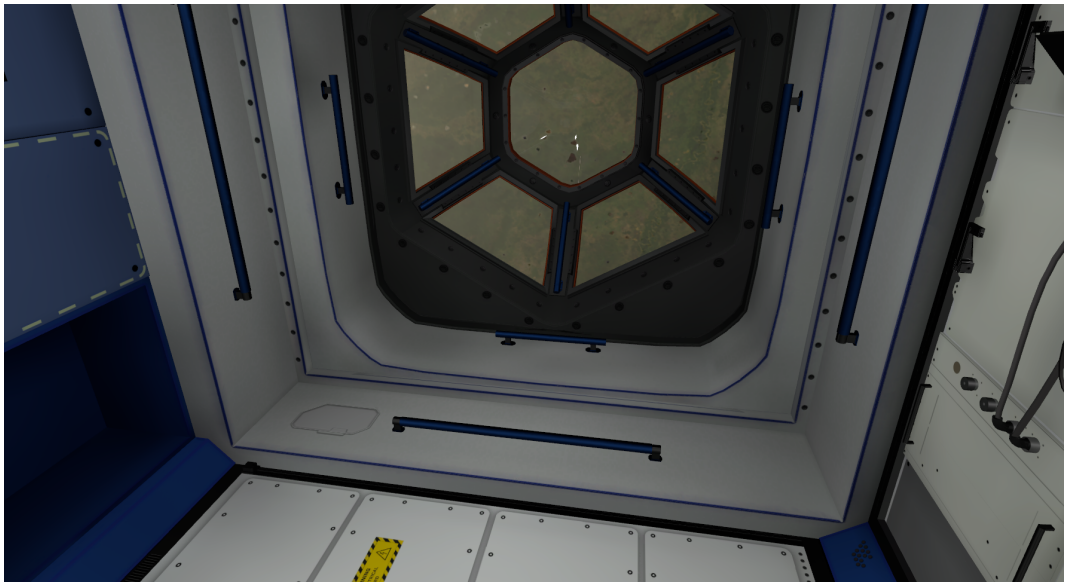
V Základní scéně je dohromady 52,3 miliónů trojúhelníků (tris) a 26,2 miliónů bodů (verts). V pokročilé scéně je 2,45 miliónů trojúhelníků a 2,34 miliónů bodů. Ve každé

scéně je pět kamer. Pro každou kameru je v záznamu testování napsán počet trojúhelníků, které musela pro zobrazení vykreslit. Níže jsou obrázky pohledů vybraných kamer do scény. Množství geometrie vykreslované kamerou může být několikanásobně větší, než celkové množství geometrie ve scéně v závislosti na efektech přidaných renderovací pipeline. Takové efekty mohou být například stíny.

Také se může možství geometrie lišit v závislosti na použitém headsetu, jelikož má každý jiné zorné pole. Každý tak svým zorným polem zabírá menší či větší část scény.



Obrázek 7.1. Pokročilá scéna - kamera 1



Obrázek 7.2. Pokročilá scéna - kamera 2

7.2 Výsledky testování

Při testování se ukázako, že pokud spustím aplikaci na headsetu, tak se snímky za sekundu automaticky snaží přiblížit k určitým mezím. Často se ukazovaly hodnoty 90, 45, 72, 37,5, které jsou maximalním množstvím snímků za sekundu nebo polovina

maximalních snímků za sekundu na daném headsetu. U těchto výsledků je pak tedy rozhodující zatížení grafické karty pro rozdělení náročnosti u jednotlivých testů. Zatížení procesoru se dle předpokladu ukázalo být užitečné jen v pokročilé scéně.

Porovnání URP a HDRP

Pro základní scénu byly obě pipeline ponechány v základním (default) nastavení. HDRP mělo znatelně vyšší náročnost na každý snímek, nejspíše proto, že v základním nastavení HDRP je zapnuto více efektů.

Porovnání PC

Z testů se ukazuje, že PC1 a PC2 mají podobné výsledky. Se stejným procesorem a dostatkem paměti bychom hledali rozdíl pouze u grafické karty. Ty mají dle testů podobný výkon s malým přikloněním k PC1, které občas přesahuje PC2 o jednotky procent zatížení GPU. PC3 bylo v průběhu testů nejzatíženější, často s velkým rozdílem snímků za sekundu i zatížení GPU.

Porovnání VR headsetů

V rámci testování headsetů se mimo XTAL všechny headsety shodují v náročnosti při vykreslování v URP. V HDRP jsou nejrychlejší Quest 2 a HTC VIVE. HTC VIVE PRO je pomalejší. XTAL je obecně nejnáročnější na vykreslování. Tyto výsledky jsou částečně očekávané, jelikož náročnost závisí na rozlišení každého headsetu.

Headset	Pipeline	Tris	Verts	FPS	CPU(%)	GPU(%)
-	URP	114.2M	57.1M	105	8	98
-	HDRP	149.9M	75M	55	8	98
VIVE	URP	173M	86M	44.8	8	85
VIVE	HDRP	278.1M	139.2M	20	8	99
VIVE PRO	URP	173M	86M	45	8	81
VIVE PRO	HDRP	278.1M	139.2M	19	8	99
XTAL	URP	173M	86M	37.5	8	86
XTAL	HDRP	278.1M	139.2M	16	8	99
QUEST 2	URP	173.4M	86.8M	52	8	97
QUEST 2	HDRP	278.1M	139.2M	20	8	99

Tabulka 7.1. Vybrané záznamy testů výkonu scén z konverze - PC1, Základní scéna

Headset	Pipeline	Tris	Verts	FPS	CPU(%)	GPU(%)
-	URP	10.2M	8.6M	85	28	15
-	HDRP	5M	3.9M	50	20	19
VIVE	URP	8.2M	7M	45	27	34
VIVE	HDRP	8M	6.8M	45	18	69
VIVE PRO	URP	8.2M	7M	45	26	38
VIVE PRO	HDRP	8M	6.8M	45	15	71
XTAL	URP	8.8M	7.7M	37.5	32	51
XTAL	HDRP	8.6M	7.6M	37.5	22	92
QUEST 2	URP	8.2M	7.0M	45	35	35
QUEST 2	HDRP	8.0M	6.8M	30	30	58

Tabulka 7.2. Vybrané záznamy testů výkonu scén z konverze - PC1, Pokročilá scéna

Kapitola 8
Závěr

Prvním z cílů bakalářské práce bylo popsat a otestovat podporu ray-tracingu v Unity. Byly popsány jednotlivé efekty a zároveň byl popsán ray-tracing jako takový. V testování byla měřena zátěž na stroj a porovnávána náročnost jednotlivých efektů. Při testování jsem musel nakonec testovat reflektivní efekty zvlášť, jelikož bylo potřeba množství reflektivních ploch v okolí kamery, aby byly tyto efekty zatížené.

Druhým cílem bylo navrhnutí procesu konverze a výkonnostní porovnání projektu před a po konverzi, a to i na různých počítačích a různých headsetech pro virtuální realitu. Proces konverze byl popsán společně s množstvím možných problémů, které mohou vzniknout. Byl zároveň zjednodušen poskytnutím skriptu MaterialConvertor. V následném testování bylo zaznamenáno 280 záznamů z testování na třech počítačích a čtyřech headsetech pro virtuální realitu.

V dalším vývoji je hlavním bodem rozšíření skriptu MaterialConvertor tak, aby fungoval na více variant shaderů. To by znamenalo, že by se každý materiál nemusel měnit přímo na svůj protějšek v druhé pipeline.

Literatura

[1] Unity Technologies. *Render pipelines*. 2021. https://docs.unity3d.com/Manual/render-pipelines.html. accessed: 02.01.2022.

[2] Jiří Sochor a Petr Ferkel Jiří Žára, Bedřich Beneš. *Moderní počítačová grafika*. Computer Press, 2004.

[3] *Culling Explained*. 2021. https://docs.cryengine.com/display/SDKDOC4/Culling+Explained. accessed: 02.01.2022.

[4] Unity Technologies. *Video post-processing*. 2021. https://en.wikipedia.org/wiki/Video_post-processing . accessed: 02.01.2022.

[5] *Single Pass Stereo rendering (Double-Wide rendering)*. 2021. https://docs.unity.cn/2020.3/Documentation/Manual/SinglePassStereoRendering.html. accessed: 02.01.2022.

[6] *Single Pass vs Single Pass Instanced rendering*. 2021. https://forum.unity.com/threads/single-pass-vs-single-pass-instanced-rendering.824694/. accessed: 02.01.2022.

[7] *Forward Rendering vs. Deferred Rendering*. 2021. https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342. accessed: 02.01.2022.

[8] Unity Technologies. *Rendering paths in the Built-in Render Pipeline*. 2021. https://docs.unity3d.com/Manual/RenderingPaths.html. accessed: 02.01.2022.

[9] Unity Technologies. *Universal Render Pipeline overview: Universal RP: 13.1.4*. 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@13.1/manual/index.html. accessed: 02.01.2022.

[10] Unity Technologies. *High Definition Render Pipeline overview: High Definition RP: 13.1.0*. 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@13.1/manual/. accessed: 02.01.2022.

[11] Wikipedia Contributors. *Ray tracing (graphics)*. 2021. https://en.wikipedia.org/wiki/Ray_tracing_(graphics). accessed: 02.01.2022.

[12] *Ray Tracing Gems*. Apress, 2019. http://raytracinggems.com.

[13] Wikipedia Contributors. *Ambient occlusion*. 2021. https://cs.wikipedia.org/wiki/Ambient_occlusion. accessed: 02.01.2022.

[14] Unity Technologies. *Ray-Traced Ambient Occlusion: High Definition RP: 13.1.0*. 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@13.1/manual/Ray-Traced-Ambient-Occlusion.html. accessed: 02.01.2022.

[15] Unity Technologies. *Ray-Traced Reflections: High Definition RP: 13.1.0*. 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@12.0/manual/Ray-Traced-Reflections.html. accessed: 02.01.2022.

[16] Unity Technologies. *title=Ray-Traced Global Illumination: High Definition RP: 13.1.0*. 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@12.0/manual/Ray-Traced-Global-Illumination.html. accessed: 02.01.2022.

[17] Unity Technologies. *Ray-traced shadows: High Definition RP: 13.1.0*. 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@12.0/manual/Ray-Traced-Shadows.html. accessed: 02.01.2022.

[18] *Recursive rendering: High definition RP: 13.1.0.* 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@13.1/manual/Ray-Tracing-Recursive-Rendering.html. accessed: 02.01.2022.

[19] *HDRP Ray-tracing documentation*. 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@13.1/manual/Ray-Tracing-Getting-Started.html. accessed: 02.01.2022.

[20] *Mask and detail maps: High definition RP: 13.1.0. High Definition RP — 13.1.0.* 2021. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@13.1/manual/Mask-Map-and-Detail-Map.html. accessed: 02.01.2022.

Příloha A
Struktura přiložených souborů

- Tabulka se záznamem testů
  - *TestovaciTabulka.zip*
- Ilustrační materiály
  - *imgs.zip*
- Material Convertor skript
  - *MaterialConvertor.zip*