VYSOKÁ ŠKOLA

KREATIVNÍ KOMUNIKACE

Katedra vizuální tvorby

DIPLOMOVÁ PRÁCE

Unreal Engine a integrace do existujících pracovních postupů

2024 **BcA. Šimon Kryštof Honal**

VYSOKÁ ŠKOLA KREATIVNÍ KOMUNIKACE

Katedra vizuální tvorby

Vizuální a literární umění

Animace a vizuální efekty

Unreal Engine a integrace do existujících pracovních postupů

Teoretická část: Unreal Engine a integrace do existujících pracovních postupů

Praktická část: Animovaný 3D film

Autor: BcA. Šimon Kryštof Honal

Vedoucí práce: MgA. Martin Hovorka

2024

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal. Souhlasím s tím, aby práce byla zpřístupněna veřejnosti pro účely studia a výzkumu.

V Bdeněvsi dne…… Podpis autora:

PODĚKOVÁNÍ

Rád bych touto formou poděkoval vedoucímu diplomové práce, MgA. Martinu Hovorkovi za jeho trpělivost, vstřícnost a ochotu.

Dále chci poděkovat svým rodičům za možnost studovat mé vysněné řemeslo, a za jejich láskyplnou podporu.

Děkuji zároveň i profesorskému sboru, a taktéž několika málo vyvoleným, které mám tu čest nazývat mými přáteli.

Děkuji.

ABSTRAKT

Diplomová práce volně navazuje na bakalářskou práci jménem *Unreal Engine a Průlomy v real-time renderingu*. Specificky se zajímá o integraci programu Unreal Engine do pracovního postupu (tzn. pipeline) existujícího studia.

Vývoj animovaného filmu vyžaduje propojení a spolupráci operátorů mnohých počítačových programů, jejichž individuální výstupy se kombinují pro finální output dat. Jak je popsáno v jmenované bakalářské práci, Unreal Engine je programem původně vyráběným pro vývoj a podporu počítačových her, a i přes posun ve filosofii designu směrem k filmové produkci, některá specifika zůstávají.

Tato technologická specifika s sebou přinášejí rozdíly, změny a často, pro filmařinu, netradiční pracovní postupy. Jeho integrace do již funkčního a zavedeného pracovního postupu tedy představuje nové výzvy nejen pro individuální studia, ale i celý audiovizuální průmysl.

Hlavní část diplomové práce je věnována popisu specifických problémů, které zpravidla vznikají v komunikaci mezi tradičními programy animované produkce a programem Unreal Engine. Práce tyto problémy analyzuje, vysvětluje jejich příčinu a důsledně slouží jako manuál, jak tyto problémy řešit.

Hlavním cílem práce je tvorba struktury pro bezproblémovou integraci programu Unreal Engine do studií, které nejsou zvyklé na pracovní postupy s herními enginy. Dále tuto strukturu rozšiřuje o návody a tipy, jak využívat funkce specifické pro Unreal Engine pro zlepšení, zrychlení a zpříjemnění animované produkce.

Svou poslední část diplomová práce věnuje popisu vývoje krátkého animovaného filmu *„Darrowshire“*, který byl vyroben za využitím Unreal Engine společně s jinými animačními programy.

KLÍČOVÁ SLOVA

Pipeline, grafika, počítačová grafika, přípona souboru, geometrie, workflow, engine, rendering, Unreal Engine, animace, vizuální efekty, virtuální produkce, filmový průmysl, videoherní průmysl, ray-tracing, path-tracing, blueprint.

ABSTRACT

This master thesis loosely follows on from the bachelor thesis by the name *Unreal Engine and Breakthroughs in real-time rendering*. It's specifically interested in the integration of Unreal Engine into an existing studio pipeline.

The development of an animated film requires the joint effort and cooperation of operators of many varying computer programs, the outputs of which later combine into the final product. As described in the bachelor thesis, Unreal Engine was originally developed for the purposes of video game development and support, and despite its shift in design philosophy more toward film production, certain differences remain.

These technicalities usually represent quite unorthodox workflow changes as compared to typical, traditional VFX workflows. Unreal Engine's integration into an already working and stable workflow represents new challenges not just for individual studios but for the audiovisual industry as a whole.

The bulk of the thesis is dedicated to describing specific issues which generally arise in the processes of communication between traditional computer animation programs and the Unreal Engine. This thesis analyzes these issues, explains their root causes and consequently serves as a manual, as to how to solve and avoid these issues in the first place.

The goal of this thesis is the formation of a clear structure for problem-free integration of the Unreal Engine program for studios not used to game engine workflows. Furthermore, this structure is expanded on with guides and tips on features specific to the Unreal Engine, used for expediting and improving production.

The final part of the thesis describes the creative process of using the Unreal Engine in tandem with other production packages on the short film *„Darrowshire“*.

KEYWORDS

Pipeline, graphics, computer graphics, file extension, geometry, rendering, Unreal Engine, animation, visual effects, virtual production, film industry, videogame industry, ray-tracing, path-tracing, blueprint.

OBSAH

1. ÚVOD

Před samotným úvodem mi dovolte krátkou předmluvu. Tato diplomová práce rozšiřuje a navazuje na bakalářskou práci *Unreal Engine a Průlomy v real-time renderingu.* Jako taková se tedy na danou práci často odkazuje a čerpá z ní informace nezbytně důležité pro uchopení témat prozkoumaných v práci diplomové. Jde o hierarchicky vertikální větvení, kdy poznatky jedné rozšiřují poznatky druhé. Ačkoliv nejde o naprostou nutnost, i tak naléhám na čtenáře: Nejlepší pochopení tkví v prvotním pročtení práce bakalářské, která tvoří teoretické pozadí této diplomové práce. I tak je samozřejmostí, že každá publikace musí stát sama o sobě, a tak je tomu i zde. Z tohoto důvodu, a kvůli specificitě tématu, jsou části bakalářské práce v této práci přímo citovány, a to především v úvodních kapitolách. Tyto kapitoly shrnují nejdůležitější závěry bakalářské práce, se kterými diplomová práce nadále pracuje.

Akademický čtenář tedy s dovolením omluví přílišné citace práce bakalářské.

Dále, následující kapitoly čerpají především z osobní zkušenosti autora při práci s, a adaptací relativně nového, málo zdokumentovaného programu. Chápejme tedy, že pokud chybí citace u pasáží, jako „musí se tedy…“, jde o syntézu vlastní profesní zkušenosti, nikoliv o akademicky dohledatelnou informaci. Cílem této práce je právě transformace takové zkušenosti do akademicky dohledatelné informace.

Děkuji.

Produkce animovaných médií za využití počítačů, tedy počítačové animace, efektů, prostředí a jiných aspektů, již v dnešní době není nic nového. Ba naopak, od dob prvních filmů Dream Works, prequel trilogii Star Wars, až po Marvel filmy dneška, počítačová animace měla dostatek času na ukotvení a upevnění některých základních pracovních postupů. Tyto postupy jsou zpravidla spojeny s danými programy, které nejlépe vyhovují jejich vývoji. Podle aspektů jako je rychlost, uživatelská přístupnost či složitost užití se postupem času etabovaly různé programy v různých oblastech produkce.

Příklady zahrnují Autodesk Maya pro animaci, díky skvělému systému virutálních loutek (rigging), či možnosti vytváření realistických kožešin pro roztomilé zvířecí hrdiny (Yetifurs), Blender pro modelování strojních a pevných (tzn. „hard-surface“) modelů, Houdini pro efekty a simulace jako jsou exploze, tekutiny, mraky nebo Nuke pro finální kompoziting, postprodukci a sestavení samotného výsledného obrazu.

Všechny tyto programy mají svá specifika, vlastní způsoby práce a výroby. Obecná pravidla, jako jsou kartézské souřadnice, či matematické principy geometrie zůstávají stejná, ale existují technikality, které, ačkoliv se na první pohled mohou zdát malé, v důsledku malé nejsou a při výrobě se s nimi musí počítat. Je tedy potřeba pracovní postup, kdy na práci vykonanou v programu X následně navazuje práce v programu Y a jejich kompatibilita a návaznost musí být zaručena.

Tuto návaznost zpravidla zajišťuje tzv. TA – Tech Artist. Jde o pozici umělce hybrida, který nejen, že rozumí uměleckému obsahu daného díla, také musí hlouběji rozumět technikalitám daných programů, především jejich interkompatibilitě. Krom něj ale samozřejmě daný workflow zajišťují i samotní umělci operátoři, kteří, dle daných parametrů, nálsedují pravidla práce tak, aby byl zachován hladký pracovní postup a samotný průběh práce. Taková pravidla zahrnují například počet polygonů v modelu, zda je model vodotěsný, nebo zda má čistou čvercovou topologii (=rozvržení geometrie) a tato pravidla umělcům určuje samotný Tech Artist.

Během posledních dvou desetiletí, tzn. od roku 2000, se především díky zkušenosti a dané technologii tato pravidla do určité fáze ustanovila.

1.1 Tradiční pravidla 3D animované produkce

Obr. 1 – Tradiční pipeline animované produkce

Na rozdíl od videoher, pracovní postup u animované produkce, či jiné filmové produkce za využití VFX technik, je lineární proces. Tento proces nabízí plnou kontrolu nad všemi aspekty finálního obrazu, převážně protože využití offline renderingu zajišťuje přítomnost technik jako jsou cryptomasky, či správné render passes, se kterými se dále pracuje ve fázi kompozitingu, během postprodukce.

Během preprodukce, tzn. během doby vývoje, kdy se připravujeme na samotnou produkci, vytváříme kinematografický režijní dokument, či pořádáme casting pro dabéry, také rozmýšlíme výběr softwaru a celkový postup produkce, potažmo zvolení pracovních postupů vhodných pro danou produkci. Pokud zvolíme tradiční pipeline 3D animované produkce, pak je středem pozornosti především software Maya a kouzelníci, záchranáři (myšleno upřímně) z departmentu postprodukce. Hlavním důvodem je rendering.

Rendering, tj. vykreslování, se vždy řešil pomocí tzv. offline renderingu.

„Rendering (česky renderování, či vykreslování) je proces, při kterém jsou pomocí počítačové grafiky vizualizována data ve formě obrazu. Vždy jde o poslední fázi produkce ať už animovaných, či jinak digitálně upravovaných filmů (např. filmů, které uživají digitální vizuální efekty).“[1] (Honal, 2022)

Při offline renderingu se data využívaná pro následné sestavení obrazu, tj. hotová prostředí, postavy, či animace, posílají na výpočetní farmu, kde se takzvaně „vyrenderují“.

[1] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]

Jde tedy o výpočetně náročný a pomalý proces, ale také jde o proces – kolegové z VFX prominou - kdy tolik nezáleží na optimalizaci daných dat, na rozdíl od online renderu.

„Jako takový, rendering dělíme do dvou hlavních skupin: Offline (pre-calculated) a online (real-time) rendering. Offline rendering se používá pro média, která nejsou interaktivní. Filmy, reklamy, televizní produkce a podobně. Offline rendering si tedy může dovolit dlouhou časovou prodlevu, během které je produkce odeslána do tzv. render farem, na kterých se následně renderuje třeba celé týdny. Render farmy jsou velké střediska počítačů určená výhradně renderingu nebo například propočítávání fyzikálních simulací. Naproti tomu, online, tedy real-time rendering je forma vykreslování, která musí být co možná nejrychlejší a nejvíce optimalizovaná. Donedávna se užívala skoro exkluzivně ve sféře počítačových her. Oba druhy renderingu také vyžadují vlastní přístup umělců i techniků. Pokud je offline rendering pomalý, ale zato detailní, můžeme si při práci s ním dovolit užívat velice detailní modely, ve kterých jsou všechny detaily přímo vymodelované v do sítě polygonů model tvořících. Assety tak není nutné optimalizovat. Také se nemusíme starat o jejich strategické rozložení v kompozici, na rozdíl od rozložení assetů v počítačových hrách, kde potřebujeme, aby byly všechny objekty optimálně vykresleny s ohledem na výkon počítače. To samozřejmě neznamená, že v offline renderingu si můžeme dovolit plýtvat výkonem, nicméně přístup k němu je mnohem volnější a dostupnější umělcům, kteří chtějí pracovat s maximálním detailem.“ [2] (Honal, 2022)

Především ve světě offline renderingu, se kolem samotného renderu, a kolem následné postprodukce, točí prakticky všechna pravidla animované produkce. Příliš detailní, či komplexní, neoptimalizovaný model? Čas renderu neúměrně roste. Nestihne se render? Oddělení postprodukce stojí. Jenže postprodukci se datumy odevzdání neprodlužují, tudíž následně dělají tu samou práci v o to kratší době. Daná pravidla je tedy nutno dodržovat.

Výpočetní technika se neustále vyvíjí a zdokonaluje. Offline render jako takový se využívá především z nutnosti. Kdyby existovala taková technologie, kdy se Shrek nebo podobný film renderoval okamžitě před očima, byla by pro tento úkon využita. Samozřejmě, toto není důvodem jediným a osamoceným, ve hře jsou i velice důležité aspekty precizní kontroly obrazu, render layers, kryptomasky apod., tyto aspekty prozkoumáme později. Shrek se tedy renderoval 5 000 000 CPU render-hodin, kdy byl

[2] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]

renderován nejen offline, ale skrze softwarový translation layer, tj., bez nativního využití dedikovaných grafických karet (GPU).

Současně, v trošku jiném, ale zároveň úzce souvisejícím průmyslu, průmyslu videoher, se začínala osvědčovat 3D grafika renderovaná v reálném čase. Samozřejmě, v oné době, bavíme se o roku 2001, se zatím nemohla Shrekovi ani jakékoliv jiné čistě animační tvorbě vyrovnat. To mělo trvat ještě více než desetiletí. I tak, počítačové hry, jejich vykreslování a především zájem jak ze strany hráčů, tak ze strany vývojářů, dával růstu čím dál tím sofistikovanějším grafickým kartám. Síla vykreslování počítačových her také tkví v nativním využití zmiňovaných GPUs, grafických karet, či, chcete-li, grafických procesorů. Bez zacházení do počítačové vědy, GPU nepracují v lineární posloupnosti tak, jako CPUs, centrální procesory. Reprezentaci virtuálních světů vnímají maticemi (matrices), a tím pádem jsou mnohem efektivnější při vykreslování trojrozměrného prostoru. Nicméně, je nutno podotknout, že GPUs ke své práci stále potřebují CPUs, centrální procesory. Toto téma budeme dále zkoumat.

Výpočetní výkon směřovaný směrem rychle vykreslované grafiky se neustále zvyšoval. Nevyhnutelně tedy padla otázka: Co kdybychom tuto techniku využili pro vykreslování filmu, či efektů? S dostatečným výkonem by se render time značně redukoval, či, v ideálním případě, úplně zmizel.

1.1.1 Počítačové hry, rendering a pravidla obecně

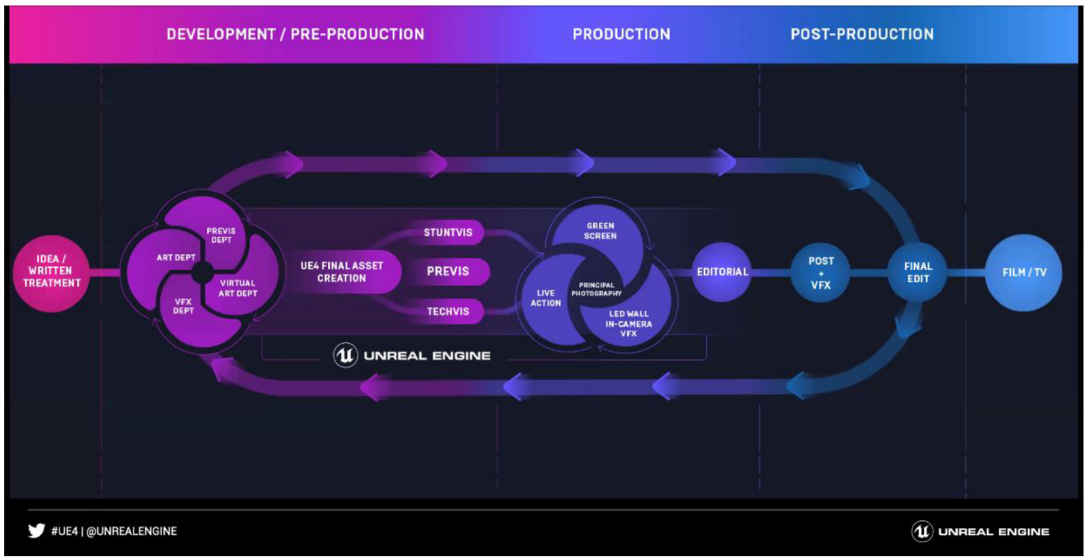
Unreal Engine je program, který spadá do kategorie tzv. herních enginů. Herní enginy jsou programy, které pohánějí a tvoří pomyslnou kostrč počítačových her. Mají za úkol zvládat programované i dynamické interakce v rámci hry, samotný rendering, fyziku, chování postav a celkově podporují celou hru. Jsou případy, kdy je engine doprovázen modulem, či kompletně separátním enginem, například pro zvládání fyzikálních interakcí. Příkladem je Grand Theft Auto IV a integrace prorieterního fyzikálního modelu, avšak toto není pravidlem. Pro naše účely se budeme soustředit na herní engine jakožto komplexní sadu nástrojů v rámci enginu, jako je tomu v případě Unreal Engine.

Tedy nejde pouze o render engine, nejde pouze o nástroj využívaný pro rychlé vykreslování scény. Tento fakt je naprosto zásadní pro správné a funkční uchopení enginu jako nástroje. Člověka to totiž k tomuto závěru navádí. Rychlý, či okamžitý render, komplexní prostředí, animační nástroje… „Tak to vyrobíme v Maye a pošleme to do Unrealu na render!“

Je to jednoduché a logické, nezvažujeme žádnou specifickou technikalitu. Taková specifická technikalita by zahrnovala například různé rozvržení virtuální loutky nebo nešťastné využití UDIMs při připravování modelů, či nepřipnutí simulovaného kabátu přímo na tělo dané loutky. Bavíme se o již skutečných specifikách softwarů, kdy, při zvolení pracovního postupu, se kterým daný software z technických důvodů jednoduše nesouhlasí, onen software prostě nahlásí chybu, a produkce stojí.

Tak je to i v případě Unreal Engine. Jakožto herní engine se řídí především pravidly ze světa her. Vodotěsné modely, modely tvořené plnými, souvislými a především spojenými geometrickými sítěmi, namísto rozvržených objektů, modely uzpůsobené pro lightmapping, s detaily zvládnutými pomocí optimalizovaných textur na rozdíl od skutečných vysoce detailních reprezentací. Toto je pouze obecný přehled pravidel ze světa produkce her. Nutno podotknout, že právě změny v těchto pravidlech jsou tím, co přineslo revoluci a širší adaptaci tohoto programu, a tyto změny budeme nadále popisovat a zkoumat. Ale i tak není pravdou, že by člověk mohl jednoduše převzít scénu z programu X a „převést ji“ do programu Y bez úprav či bez hlubším zamyšlením nad tím, jak má daná věc pracovat.

Samotný pracovní pipeline Unreal Engine vypadá méně jako lineární proces, kdy se v různých dobách produkce věnujeme převážně různým softwarům, ale spíše jako koncentrický kruh, v jehož centru je samotný Unreal Engine, kolem kterého orbitují ostatní specifické softwary.



Obr. 2 – Pipeline produkce (virtuální produkce) s UE4

1.1.2 Herní engine a real-time rendering

Při jakékoliv práci s renderingem, ať už jde o offline či online rendering, vždy pracujeme s určitým budgetem, rozpočtem výkonu počítače na kterém samotný render hodláme provádět. V případě offline renderu, jak jsme již zmiňovali, jde typicky o render farmu, v případě menších projektů o pracovní stanice. Přesuneme-li se do světa online renderingu, bavíme se typicky o domácích počítačích, či v lepším případě „herních“ počítačích, avšak najčastěji jde o herní konzole.

To má své implikace. Offline rendering, farmy, mají k disposizi nepoměrně větší výpočetní výkon než jakákoliv domácí počítačová sestava. Konec konců, velká výpočetní centra jsou k tomuto úkonu přímo navržena. Nejde tedy o interaktivní obsah. Umělci si mohou dovolit mnohem větší úroveň detailu a propracovanosti svých modelů, prostředí, postav a nasvícení, než u obsahu interaktivního, vykreslovaného na domácím stroji.
V offline produkci tedy není problém využívat modely s vysokou hustotou polygonů (high-poly modely) či rigy (virtuální loutky) s masivním počtem kostí.

Naproti tomu, umělec pracující ve sféře počítačových her, nebo například architekt, využívajíc herní engine k architektonické vizualizaci, který chtěl i tak klientovi ukázat propracovaný a rychlý render, byl nucen (a do určité „zdravé“ míry stále je) svůj obsah

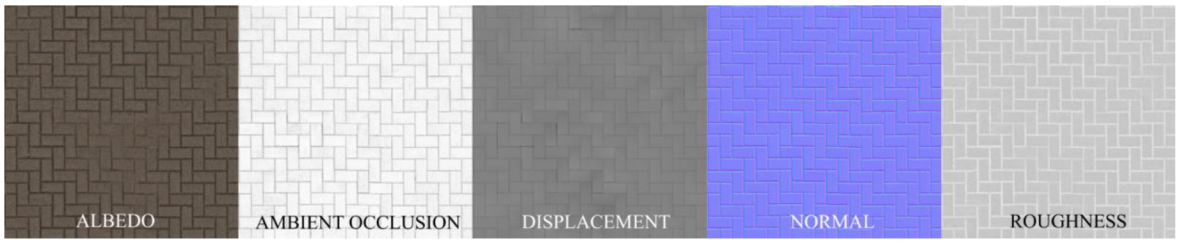
vyrábět s ohledy na limitace real-time rendereru a výpočetní techniku, se kterou pracuje. Tento přístup se nazývá optimalizace.

„Ale jak tedy v real-time renderingu zobrazujeme detailní modely, či modely s velkým počtem polygonů? Odpovědí je, že je nezobrazujeme. Pro real-time rendering totiž využíváme verze modelů, které mají menší počet trojúhelníků. Těmto modelům říkáme tzv. „low-poly modely“, zatímco modely s vysokým počtem polygonů nazýváme „high-poly modely.“ [3](Honal, 2022)

Detaily do low-poly modelů přinášíme pomocí procesu nazývaného „baking“. Tradiční pipeline výroby modelu určeného pro real-time rendering je následující. Nejdříve je vyroben high-poly model se všemi detaily, které umělec zamýšlí. Dále je tento model redukován a je provedena tzv. retopologie, tj. změna topologie, čištění geometrické sítě po redukci původní vysoce husté sítě na síť méně hustou a optimalizovanou. Následuje proces bakingu, při kterém se detaily z high-poly verze modelu zaznamenají do formy textur, které dále aplikujeme na low-poly model, čímž simulujeme 3D detail bez nutnosti jeho skutečného geometrického vykreslování.

„Baking je proces, při kterém se data z high-poly modelu zapisují do textur – nejčastěji normal mapy, či displacement mapy. V případě normal mapy jde o datovou mapu, která obsahuje informace o normálech vektorů vůči vektorům, které určují geometrii modelu. Jinými slovy se bavíme o kolmicích k existujícím povrchům modelu. Metoda normal mappingu nám umožňuje předstírat nerovnosti a promáčkliny pomocí materiálu, který dle normal mapy tyto detaily vykresluje.“[4] (Honal, 2022)

Tyto textury tvoří základ pro tzv. PBR pipeline, způsob shadingu, neboli stínování počítačově generovaných scén.



Obr. 3 – Příklad map užitých pro PBR materiál non-metalického povrchu chodníku.

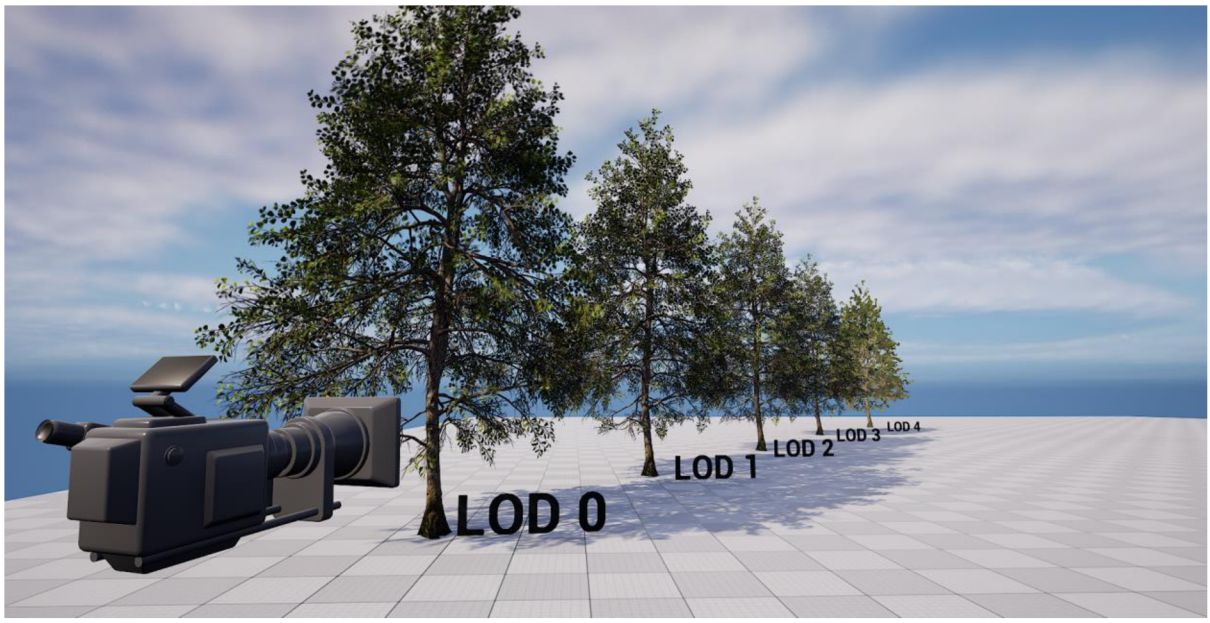
[3], [4] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]

PBR je užíváno jak v offline renderingu, tak v online renderingu, nicméně pro online rendering je velice užitečný především kvůli displacement a height mapám, které dovolují iluzi 3D geometrie, která ve skutečnosti neexistuje. Je nutno podotknout, že tyto mapy byly využívány již v starším SBR pipeline. Téma je do hloubky prozkoumáno v práci *Unreal Engine a Průlomy v real-time renderingu*.

Dalším způsobem optimalizace jsou tzv. LODs.

„LOD – Level of Detail modely jsou ještě více zjednodušené modely, které užíváme ve scéně dynamicky v závislosti na tzv. screen size. Screen size je hodnota, která určuje 28 velikost vykreslovaného objektu na obrazovce. Čím více místa na obrazovce model zabírá, tím detailnější potřebujeme, aby byl. Každý model ve scéně má určitý počet LOD modelů. Obvykle se pohybujeme kolem pěti až šesti LOD modelů na každý model, ačkoliv toto číslo záleží na využití daného modelu.“[5] (Honal, 2022)

Čím dál od kamery daný objekt je, tím méně detailní je. Pokud je objekt dostatečně daleko, nepotřebujeme ho vykreslovat v maximálním detailu. Naopak je prospěšné tento model co nejvíce zredukovat. V případě stromů je takový model často redukován až na dvojdimenzionální rovinu s pouhou obrázkovou reprezentací daného stromu. Takové reprezentaci říkáme tzv. billboard texture.



Obr. 4 – Příklad využití LOD modelů v Unreal Engine[6]

[5] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]
[6] Epic Games, Inc. *Visibility and Occlusion Culling | Unreal Engine Documentation.* 2024 [online] [Citace: 18. Květen 2024]

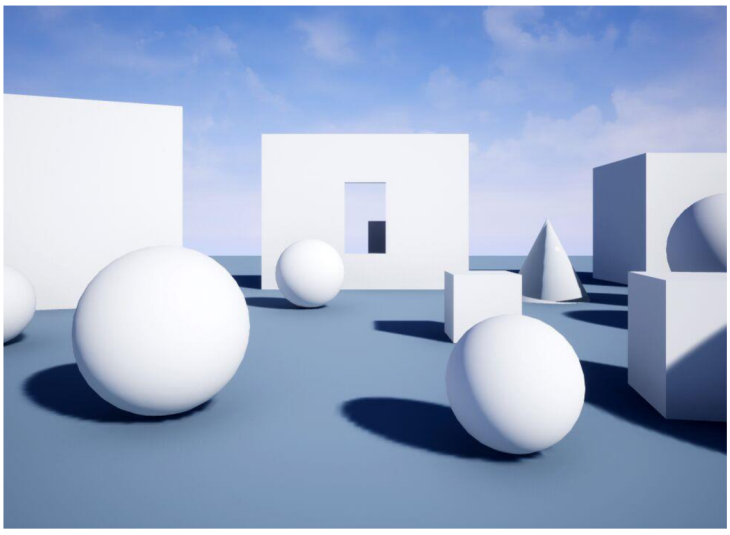
LODs můžeme buď vyrábět manuálně pro skutečně precizní kontrolu nad vykreslovaným obsahem, nicméně v dnešní době je již game enginy vyrábějí automaticky.

Další výsadou herních enginů je tzv. „culling“. Jde o „destrukci“, lépe řečeno „schovávání“ objektů, které na vykreslovaném obrazu nejsou vidět. Tato optimalizace je plně automatizovaná. Koncept za processem cullingu je jednoduchý. Co není vidět, nerenderujeme, a tím získáváme čas renderovat to, co vidět je.

„Obecná idea metod visibility a occlusion culling je zmenšení počtu objektů, které jsou v jakémkoliv daném okamžiku viditelné na obrazovce, s cílem získání výkonu.“[7] (Epic Games, Inc., 2022)

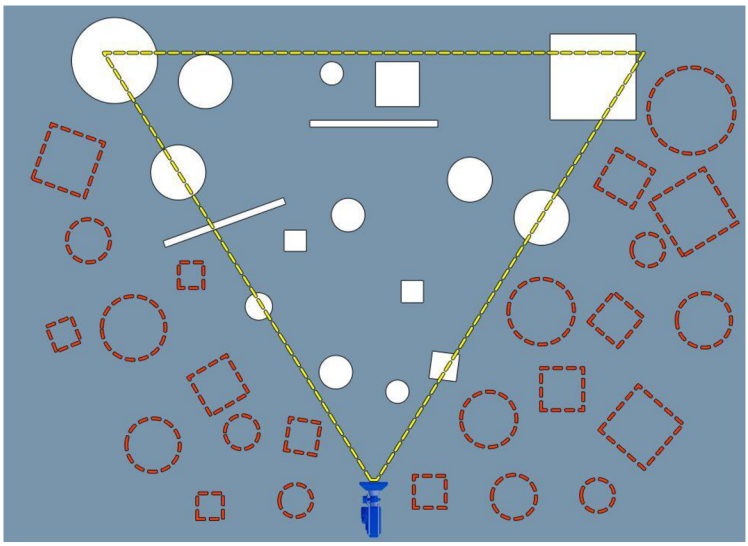
Zde je nutno podotknout, že herní enginy mohou s objekty pracovat různými způsoby. Nejpopulárnějším a nejčastějším způsobem je tzv. „per-object approach“, tzn. přístup od objektu k objektu. Tento přístup volí i Unreal Engine.

Znalost tohoto faktu je důležitá pro pochopení samotného visibility (frustum) či occlusion culling. Totiž, tradičně herní enginy volící tento přístup nemohou „schovat“ část objektu. Mohou buď vykreslit celý objekt, nebo žádný objekt, ale nikdy ho nemohly podrozdělit do menších částí, které by následně schovávaly.



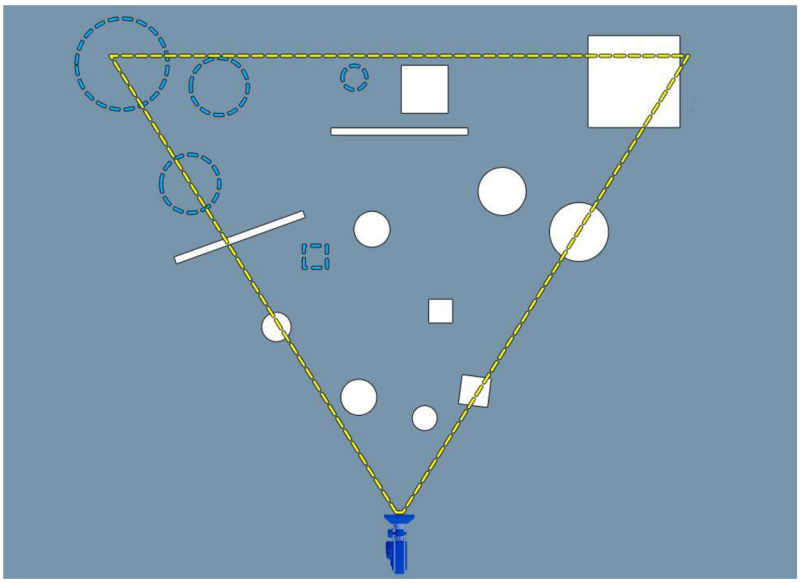
Obr. 5 - Kamerový pohled do scény[8]

[7], [8]Epic Games, Inc. *Visibility and Occlusion Culling | Unreal Engine Documentation.* 2024 [online] [Citace: 18. Květen 2024]



Obr. 6 – Náhled scény shora[9]

„Červeně vyznačené objekty díky Visibility culling nerenderujeme. Nicméně, pořád si můžeme všimnout, že dle obrázku X stále vykreslujeme některé objekty, které z pohledu kamery nejsou vidět. Jde o objekty, které jsou „schované“ za jinými objekty. Pokud chceme renderovat efektivně, potřebujeme se zbavit i těchto modelů.“[10] (Honal, 2022)



Obr. 7– Znázornění objektů ve scéně, které jsou zakryté jinými objekty[11]

„Díky occlusion culling zbytečně nerenderujeme objekty, které kamera neuvidí. Stojí za to zmíňit, že ale pořád renderujeme objekty, které vidíme byť jen z části. Jak jsme

---

[10] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]
[9],[11] Epic Games, Inc. *Visibility and Occlusion Culling | Unreal Engine Documentation.* 2024 [online] [Citace: 18. Květen 2024]

již zmiňovali, renderer pracuje na bázi objektů a tudíž pokud vidí byť jen část daného objektu, musí jej renderovat celý.“[12] (Honal, 2022)

Inovace v těchto optimalizačních procesech, jejich streamlining a v některých případech i totální eliminace jsou důvody úspěchu programu Unreal Engine v posledním desetiletí. I přes to je nezbytně důležité znát kontexty pracovních postupů, důvodů pro optimalizaci a důvody pro zrušení a streamlining zmiňovaných optimalizačních procesů. Pojďme si tedy tento postupný vývoj zrekapitulovat.

2. UNREAL JAKO HERNÍ ENGINE

2.1 Rekapitulace vývoje Unreal Engine

Unreal Engine byl původně vytvořen pro hru Unreal Tima Sweeneyho, který samotný engine také sám naprogramoval. Unreal byl vydán v roce 1998 a tím započalo dlouholeté rozšiřování nejen značky Unreal, ale i samotného herní enginu. I když byl původně zamýšlen jako engine čistě vyrobený pro tzv. „first person shooters“ (střílečky z vlastního pohledu), jeho flexibilita a píle vývojářů umožňovaly rozšiřování do různých dalších herních žánrů.

„Roky šly a Unreal Engine procházel různými verzemi a revizemi. Unreal Engine 2.0 byl vydán v roce 2001. Byl užit v kultovních klasikách jako jsou Unreal Tournament 2002 a Unreal Tournament 2004. Znovu se setkal s obrovským úspěchem. Jak Unreal Tournament 2002, tak i Unreal Tournament 2004 jsou obě hry multiplayerové – tzn. hry více hráčů. Jde o arénové střílečky (FPS – first person shooter), ve kterých soutěží hráči z celého světa.“[13] (Honal, 2022)

Významným milníkem bylo vydání Unreal Engine 3. Tato verze enginu byla poprvé široce licensována a využitá mnohými herními giganty jako například sérií Gears of War. Unreal Engine 4 byl poprvé představen v roce 2012, a v roce 2014 byl volně

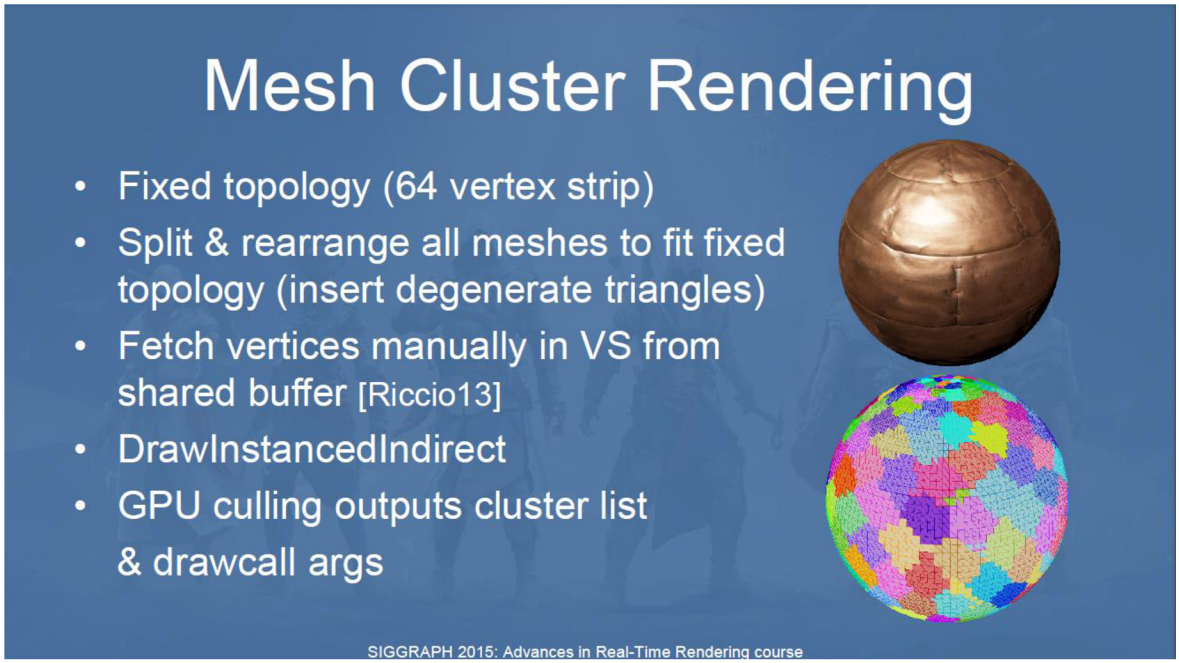
[12], [13] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]

vypuštěn mezi masy. Pro tento text je důležité představení Unreal Enginu 5, a to v roce 2020.

Unreal Engine 5 přinesl mnoho technologických inovací, některé převzal a zpopularizoval, jako například technologii virutalizované geometrie, Nanite, zatímco jiné sám přivedl na světlo světa, např. Lumen, systém globální iluminace v reálném čase. Tyto technologie jsou klíčové pro vysvětlení positioningu Unreal Engine jako herního enginu s přesahem do sféry filmu, animace a virtuální produkce.

2.1.1 Virtualizovaná geometrie, Nanite

Jednou z nejdůležitějších inovací Unreal Engine 5 je Nanite. Nanite je sourhnný název pro systém virtualizované geometrie, který dovoluje sítě modelů podrozdělovat do tzv. „clusters“, klusterů, tedy skupin trojúhelníků. Proto terminus technicus, „mesh cluster rendering“.



Obr. 8 – Slide z prezentace SIGGRAPH 2015: Advances in Real-Time Rendering[14]

Mesh Cluster Rendering byl původně vyvinut pro počítačovou hru Assassin’s Creed Unity. Tato hra se odehrává v ulicích revoluční Paříže, a studio tak potřebovalo přijít s způsobem, jak město efektivně renderovat. Tradiční per-object approach, který jsme si již vysvětlovali, by v takovém případě nefungoval dobře.

Ulice a budovy Paříže měly být vyrobeny z modulárních bloků. Typicky by tento přístup znamenal, krom neuvěřitelného množství trojúhelníků, především nepředstavitelné množství draw calls. Jak jsme si již říkali, zjednodušeně, každý objekt je vyrenderován tolikrát, kolik má materiálů. Další překážkou bylo zpracování samotných interiérů budov, které měly obsahovat obrovské množství rekvizit.[15] (Haar, a další, 2015)

[14], [15] Haar, Ulrich a Aaltonen, Sebastian. *SIGGRAPH 2015: Advances in Real-Time Rendering in Games.* 2015 [Parafráze: 19. Květen 2024]

Mimoto, v prostředí velkého města musíme řešit již zmiňované visibility a occlussion culling. Ty sice real-time rendering extrémně zrychlují, ale dokud pracujeme s paradigmatem „per-object“ – dokud pracujeme s celými objekty, nemůžeme si dovolit například nevykreslit polovinu domu, ačkoliv ji nevidíme. Toto je velký problém ve scéně, kde hledíme na mnoho velkých, někdy se částečně překrývajících budov, protože v takové situaci musíme vykreslovat i velkou část geometrie, kterou vůbec nevidíme… …Technologie Nanite naprosto mění paradigma real-time renderingu právě tím, že již nezachází s jednotlivými objekty jako s celky. Místo toho, díky tzv. mesh cluster renderingu, rozděluje modely na clustery - skupiny trojúhelníků, se kterými dál pracuje. Tento přístup má za následek například kompletní deprekaci a konec LOD modelů, které již nejsou potřeba. Navíc, díky „podrozdělení“ modelů do clusterů nyní můžeme vykreslovat pouze některé části modelu, v závislosti na zorném poli kamery. Místo low poly modelů nyní můžeme používat high-poly modely bez optimalizací. A třešničkou na dortu je sdílení vlastností clusterů mezi více clusterů na úrovni celé scény, což znamená extrémní redukci draw calls.[16] (Honal, 2022)

Tím se tedy zbavujeme nutnosti vyrábět low-poly modely. Můžeme vykreslovat high-poly zdrojovou geometrii, například rovnou exportovanou z programu Zbrush.

„Co [Nanite] tedy dělá, když víme, že se nemusíme starat o vytváření LOD modelů a retopologizaci modelů, co Nanite tedy dělá? Dělá to [LODs a retopologii] za nás? Nebo provádí nějakou zvláštní konverzi?... …Ve skutečnosti [Nanite] dělá to, že analyzuje trojúhelníkovou síť modelu, když je importována do Unreal Enginu. Potom tyto trojúhelníky rozdělí do skupin a tyto skupiny se nazývají clusters… … proč to tedy dělá, proč vytváří skupiny (clustery)? No, vytvořením skupin v podstatě síť rozbíjí. Rozděluje trojúhelníky do skupin, které může dále ovládat. A jakmile to udělá, je schopen tyto skupiny trojúhelníků dynamicky upravovat a vyměňovat za různé úrovně detailů, řekněme na základě pohledu kamery.”[17] (Conditional, 2021)

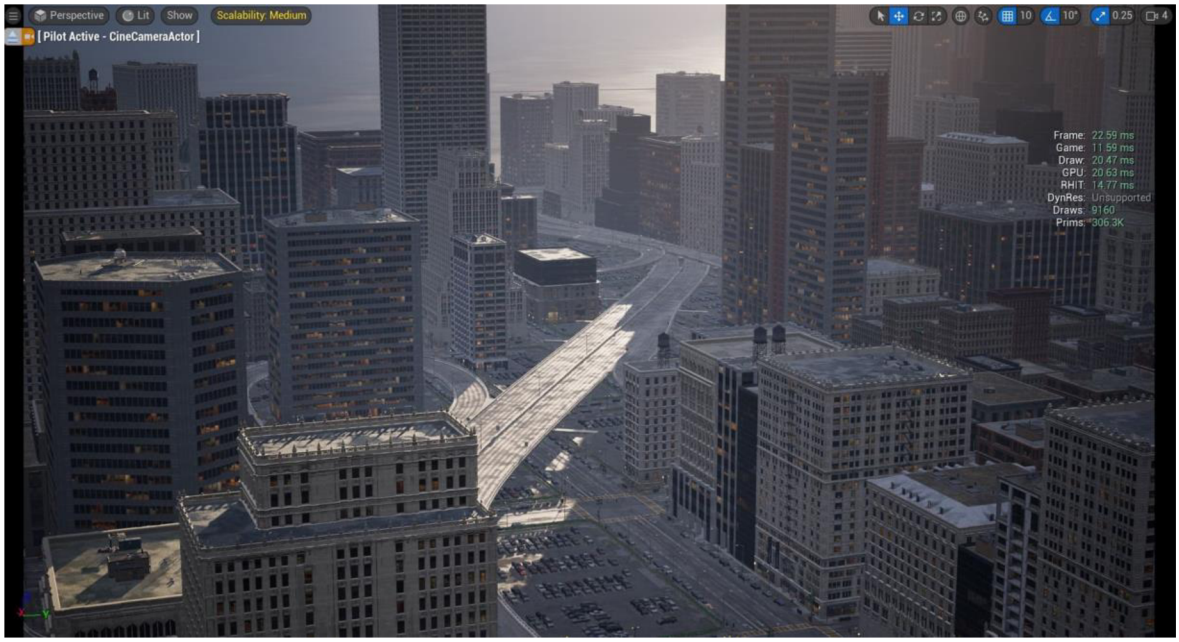
Nanite preferuje high-poly modely s hustou topologií.

„Nanite má ale i jeden velký háček. Pro to, aby Nanite mohl model rozdělit na skupiny trojúhelníků (clusters), model potřebuje mít hustou topologii. Při renderingu low

[16] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]
[17] Zero Conditional. *Absolute Beginner's Guide to Unreal Engine 5 Nanite.* 2021 [YouTube video] [Citace: 19. Květen 2024]

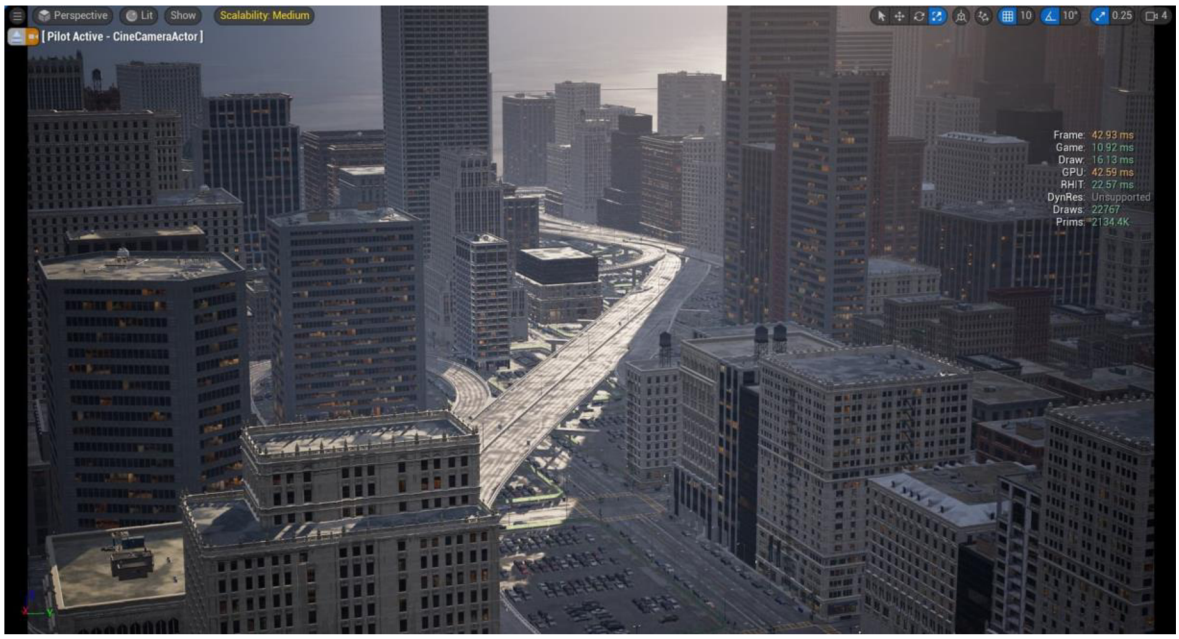
poly modelů tudíž výhody Nanite neužijeme, protože ty nejmenší trojúhelníky, zaznamenané ve vstupních datech modelu importovaného do enginu, jsou již při importu velké – optimalizované.“[18] (Honal, 2022)

Tedy, zdá se, že s Nanite se Unreal Engine přiblížil světu offline renderingu, kdy umělci již tráví méně času optimalizací high-poly modelů do low-poly sítí. Je nutno podotknout, že klasické real-time vykreslování per-object stále má své využití, a to především kvůli zmiňované hustotě sítě. Podrozdělování geometrických sítí modelů za využitím mesh cluster renderingu samo o sobě představuje cenu na budgetu výkonu. Tedy, mesh cluster rendering dává smysl především ve scénách, kdy jsou objekty renderovány přes sebe, a kdy využijeme částečné renderování daných objektů, jako je tomu například v městských scénách. Toto je jasné ze statistik (vpravo) na následujících obrázcích.



Obrázek 9., Screenshot z City Sample Demo v UE5.1 za použitím mesh cluster renderingu

[18] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]



Obrázek 10., Screenshot z City Sample Demo v UE5.1 bez použití mesh cluster renderingu

Zde se setkáváme s již zmiňovanými technikalitami. Pokud bychom v produkci chtěli využívat Unreal Engine a daná produkce by počítala s low-poly, stylizovanými postavami a prostředím, museli bychom se rozmyslet, zda pro produkci dává mesh cluster rendering smysl. Pokud by šlo o mnoho lesníc scén, kdy se objekty překrývají, zabírají mnoho místa na obrazovce a vidíme pouze jejich části, volili bychom mesh cluster rendering, tedy Nanite v Unreal Engine. Pokud ne, spíše než rendering by nás následně ovlivnily jiné aspekty Unreal Enginu, jako je například Marketplace či vyžadovaný workflow. Pokud bychom ani jeden z těchto vlastností enginu nechtěli využít, pak bychom se vrátili k tradičnímu offline přístupu.

Pokud bychom ale potřebovali velice rychle zpracovat realistickou městskou scénu, pak máme jasno. Krom virtualizované geometrie nám v tomto případě bude pomáhat i Lumen, systém dynamické globální iluminace vykreslované v reálném čase v rámci Unreal Engine.

2.1.2 Lumen, real-time global illumination

„Lumen je plně dynamický systém globálního osvětlení a odrazů v Unreal Engine 5, který je navržen pro konzole nové generace a je také výchozím systémem globálního osvětlení a odrazů. Lumen poskytuje rozptýlené vzájemné odrazy s nekonečnými odrazy a nepřímými zrcadlovými odrazy ve velkých, detailních prostředích v měřítku od milimetrů po kilometry.” [19] (Epic Games, Inc., 2024)

I Lumen přiblížil herní engine Unreal blíže světu offline filmové produkce. Toto téma do hloubky popisuji v práci *Unreal Engine a Průlomy v real-time renderingu*, pro potřeby tohoto textu následuje krátká rekapitulace.

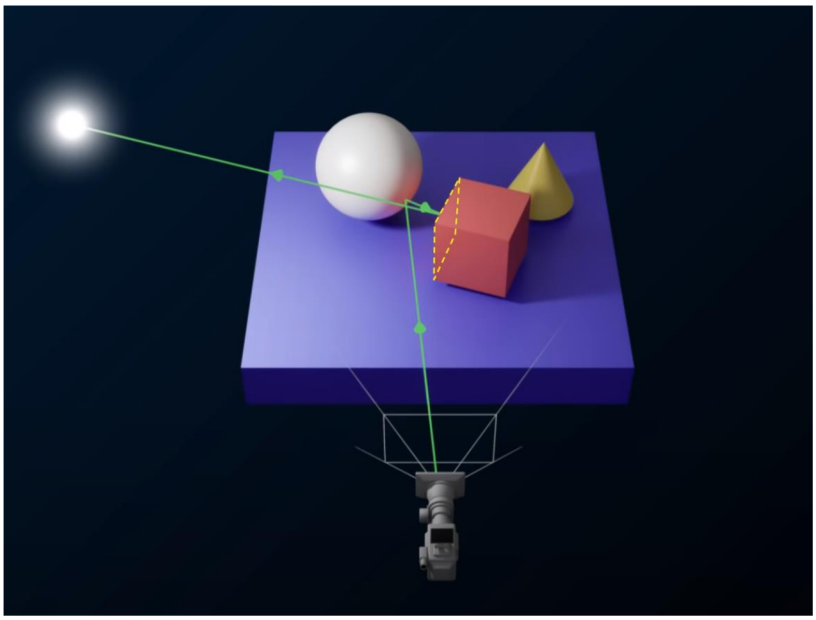
Online rendering pro nasvěcování scén využívá procesu známého pod termínem raytracing. Jde o princip analogicky opačný vůči interakcím světla, jak je známe ve světě skutečném. Z každého pixelu obrazovky přímo „vystřelíme“ simulovaný paprsek, který sledujeme dokud neprotne jakýkoliv objekt naší scény. Dle vlastností povrchu, který trefil, vytváříme paprsky nové, jde o fenomén zvaný ray-bouncing. Vlastnosti odrazu, jako je jeho úhel, tzv. scattering či absorbce jsou definovány vlastnostmi trefeného povrchu, tyto vlastnosti nazýváme shadery. Paprsek se po určitém počtu odrazů, který definuje operátor, odráží směrem nejbližšího zdroje světla. Tímto zdrojem může být například žárovka lampy či samotné Slunce. Každý světelný zdroj má přiřazený vlastní počet onech odražených paprsků. Pokud paprsek na své cestě za tímto zdrojem narazí na další objekt, zpětně pak definuje stín daného objektu.[20] (parafráze, Honal, 2022)

Paprsky jsou vrhány z kamery, aby se šetřil výkon počítače, protože každý paprsek je výpočetně náročný. Pokud bychom paprsky vysílali ze světelných zdrojů, museli bychom počítat interakce světla v celé scéně, včetně částí, které kamera nevidí. Posíláním paprsků z kamery renderujeme jen to, co je v jejím zorném poli, čímž šetříme čas.

„Screen-space paprsky mohou interagovat pouze s objekty v zorném poli kamery. Ale ve většině případů existují části obrazovky, ke kterým se screen-space traces nemohou

[19] Unreal Engine. *Lumen in UE: Let there be light!* | *Unreal Engine.* 2021 [YouTube video] [Citace: 19. Květen 2024]
[20] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]

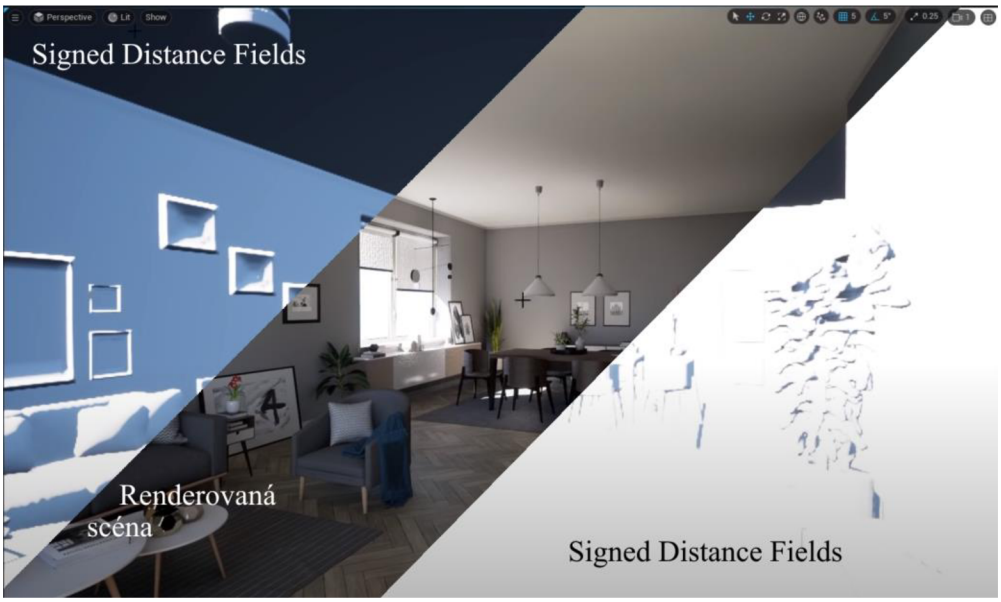
dostat. A pro správné osvětlení a odrazy musíme neustále brát v úvahu objekty i mimo obrazovku."[21] (Unreal Engine, 2021)



Obr. 11 – Příklad roviny (strany krychle), ke které se při užití screen-space ray-tracingu nemůže paprsek dostat.

Tento nedostatek raytracingu byl pravděpodobně hlavní hnací silou při vytváření systému Lumen. Lumen řeší tento problém za pomoci tzv. Signed Distance Fields (SDF). Každý objekt ve scéně, který má určitý objem, jinými slovy, každý model, má přiřazený SDF, jakožto objemovou reprezentaci daného modelu. Jde o zjednodušenou verzi daného modelu. Za pomoci tzv. Cards, které můžeme přirovnat k "opačným cubemapám", zachycujeme odlesky, barvy a ostatně různé vlastnosti okolní scény, které nadále využíváme pro render scény.

[21] Unreal Engine. *Lumen in UE: Let there be light!* | *Unreal Engine.* 2021 [YouTube video] [Citace: 19. Květen 2024]

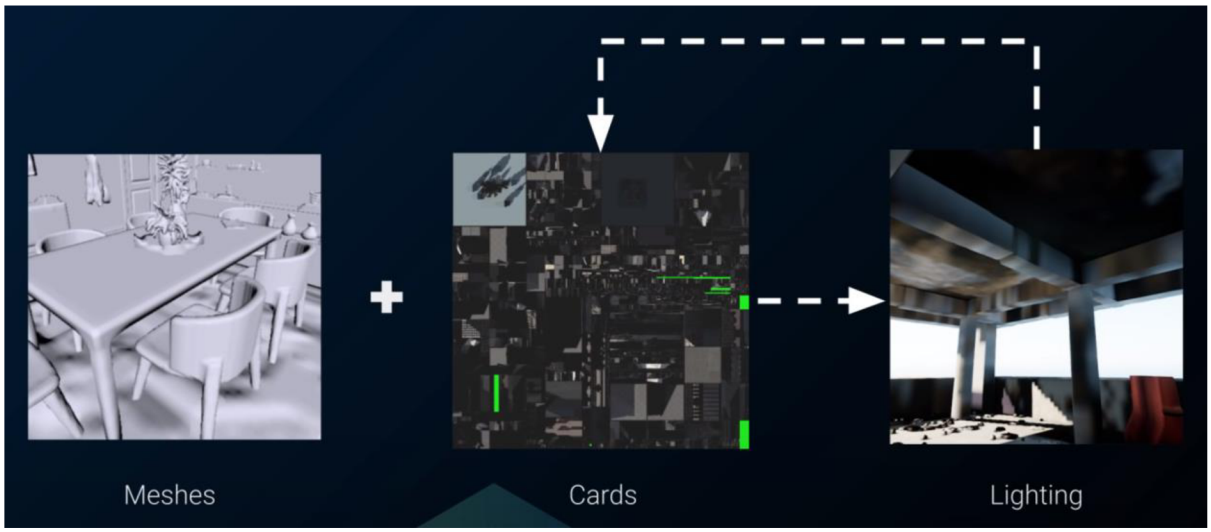


Obr. 12 – Vizualizace mesh SDFs v Unreal Engine.

„Vygenerovali jsme parametrizaci povrchů blízkých objektů, kterou nazýváme „povrchová mezipaměť“. Tato mezipaměť se používá k vyhledání informací o povrchu a osvětlení, když paprsek zasáhne daný (tento) bod. Berte to jako způsob, jak uložit do mezipaměti informace o povrchu a osvětlení v GPU, abyste je mohli později použít.“ (Epic Games, 2021)[22]

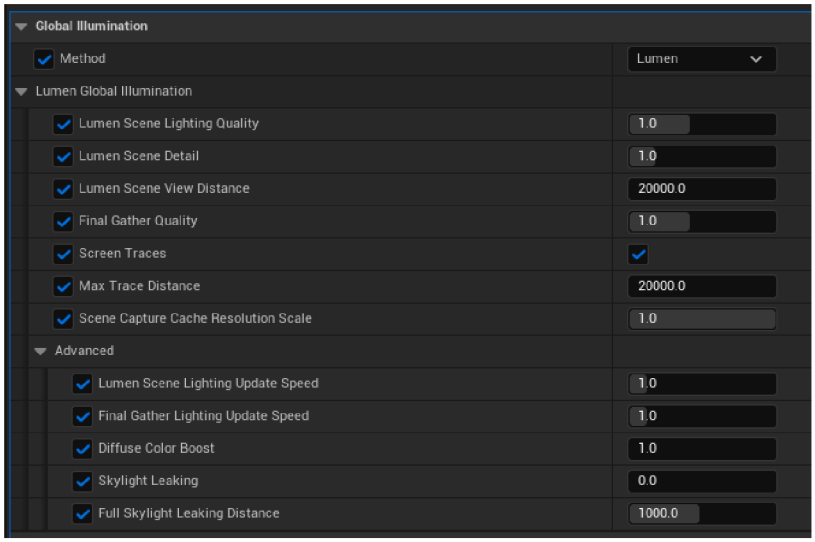
“Abychom získali vlastnosti povrchu blízkých objektů, využíváme tzv. cards. Cards bychom mohli přirovnat k „opačným cubemapám“. Zatímco tradiční cubemap nám popisuje offline-vygenerované odlesky okolního prostředí, cards obdobná data zachycují. Tato data poté Lumen používá vyrenderování všech vlastností povrchů objektů ve scéně, z několika úhlů. Například, pokud chceme vykreslit krychli na chodníku, projekce cards vyrenderuje 5 stran naší krychle a tato data uloží do povrchové mezipaměti. Z těchto dat nyní Lumen může spočítat jak přímé, tak nepřímé osvětlení. Výstup osvětlení se poté užívá ve smyčce, přičemž output osvětlení je zachycen cards, které tato data dále propagují v dalších snímcích.“[23] (Honal, 2022)

[22] Unreal Engine. *Lumen in UE: Let there be light!* | *Unreal Engine.* 2021 [YouTube video] [Citace: 19. Květen 2024]
[23] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]



Obr. 13 – Pipeline pro vyrenderování osvětlení pomocí Lumen.

Z předchozí citace je značně důležitá poslední věta. Výstup se užívá ve smyčce a data se propagují v dalších snímcích. Jinými slovy, jde o temporální přístup vykreslování. Podobně, jako u samotného raytracingu, se nyní bavíme o procesu závislém na zorném poli obrazovky a obsahu na ni přítomného. Jde o takzvaný „screen space“. Do pipeline renderu jsme ale přidali již zmiňovanou mezipaměť. Ona mezipaměť je z definice závislá na obsahu obrazovky, který se mění každý snímek, avšak ona mezipaměť existuje právě pro optimalizaci rychlosti vykreslování obrazu. Frekvence její proměny a zaznamenávání změn tedy nemůže být okamžitá, alespoň ne natolik, jako je tomu u raytracingu. Nunto podotknout, že ona frekvence je parametrem, který může operátor do značné míry upravovat.



Obr. 14 – Parametry Lumen v UE5

Tedy existuje určitá časová prodleva mezi změnou nasvícení scény pomocí raytracingu a zaznamenání oněch změn do mezipaměti Lumen. V určitých „edge-cases“ má tento fakt negativní konsekvence.

Kupříkladu, pracujeme-li s absolutně temnou scénou, do které postupně vpouštíme světlo, ona prodleva v mezipaměti způsobí, že propočítání globální iluminace bude vůči raytracingovým interakcím opožděno. Tedy, bude vznikat nepříjemný efekt, kdy stíny, odlesky, spekularita a jiné vlastnosti povrchů budou doslova opožděny. Pro ilustraci, ve scéně může být naprosto jasný zdroj světla, který již vidíme, a přeci ono světlo několik snímků zdánlivě neinteraguje s okolním prostředím. S touto limitací jsem se profesně osobně setkal na projektu Morricone, který jsem společně s režisérem Martinem Pokorným měl na starost.

Díky dostatečnému času na Research a Development projektu, testování a trpělivosti všech přítomných jsem byl schopen s touto limitací pracovat a následně ji i obejít pomocí tzv. Light Functions přítomných v toolsetu Unreal Engine. Vyjadřuji svůj vděk kolegům z offline renderu a VFX, kteří se posléze na projektu podíleli a mnohé snímky pro projekt vytvořili tradičními offline workflows, což nahradilo čas strávený nad RaD enginu.

Obr. 15 – Problematické snímky z Morricone, kdy latence Lumen zaostává za zdrojem světla

Obr. 16 – Lumen ve spojení s raytracing v projektu Morricone

Tyto, a podobné screenspace efekty, jako jsou například, SSGI (screen-space global illumination, předchůdce Lumen), SSFS (fog scattering), SSR (reflections), SSAO (ambient occlusion) existují především jako způsob jak z dat, které již máme k dispozici – už se vyrenderovala – extrapolovat a odvozovat co možná nejvíce informací o scéně, kterou se snažíme vykreslit za cílem maximální optimalizace a minimální prodlevy v doručení snímku na obrazovku.

Ačkoliv je tedy Lumen velice mocným nástrojem, a skutečně poprvé se bavíme o real-time global illumination, je pořád nutno si uvědomit, že nejde o náhradu offline renderingu a plně fyzikálně korektního propočítání světelných interakcí scény. Se specifiky těchto technik se musí plánovat a počítat, a workflow daného projektu se jim musí podřídit.

Vývoj Unreal Enginu pochopitelně provázely i jiné skokové technologie, které zde nebyly zmíněny. Některým se ještě budeme věnovat, ale Lumen a Nanite se považují za ony přelomové vlastnosti Unreal Engine 5, které provázejí a představují posun paradigmatu Epic Games od videoher k virtuální produkci a světě VFX a filmařině.

2.2 *Unreal Engine jako sada nástrojů*

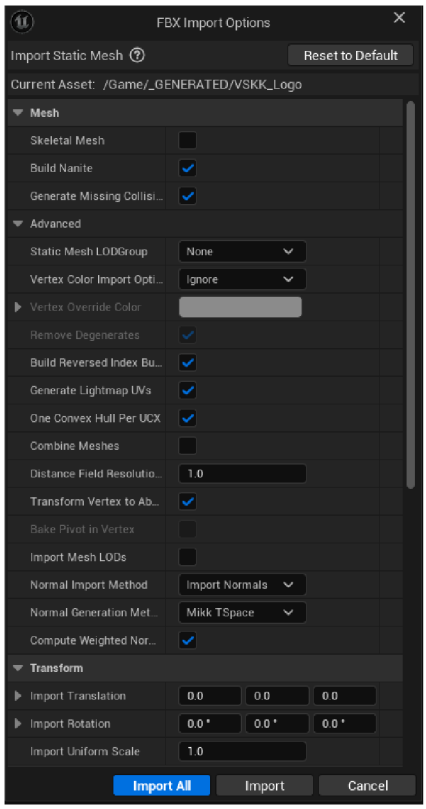
Unreal Engine je třeba vnímat jako soudržnou sadu nástrojů, ne pouze jako render engine. Pokud tak neučiníme, je nutno zvážit, zda má smysl jej využívat vzhledem k mnoým specifikám pracovních postupů.

Začněme popořádku. Počítačově generovaný, či upravený obraz je tvořen z mnohých podsložek. Pozadí, jako jsou nebe, mraky či cizokrajné visty bývají vytvořeny za pomocí tzv. mattepaintingu. Rekvizity, objekty, květiny, či hrdinové jsou reprezentováni trojrozměrnými sítěmi neboli 3D modely. Pokud daná věc krvácí, kape, či stříká, bavíme se o tzv. particle effects, neboli efekty částic. Za řekami, nápoji, vodou či mlékem stojí fluidní simulace, a za mlhou volumetrické efekty. V tomto popisu bychom mohli pokračovat, a také budeme, ale nyní jde pouze o ilustraci různorodosti komponent obsahu obrazu.

Každá tato komponenta je vyrobena buď v externím programu, například mattepainting, malovaná oblaka, v Adobe Photoshop či Krita, a nebo přímo v herním enginu, příkladem může být volumetrická mlha. V některých případech si můžeme vybrat mezi vytvořením obsahu přimo v daném herním enginu, v případech jiných – častěji - tento obsah vyrábíme v externím programu za účely pozdějšího importu a využití v daném enginu.

Výrobu tohoto obsahu musíme patřičně plánovat a přizpůsobit se tak náležitostem s prací námi zvoleným přístupem. Abychom tak učinili správně, je nutné znát vlastnosti nástrojů, kterými engine reprezentuje vyrobený obsah.

Nejčastějším obsahem, se kterým pracujeme, jsou 3D modely. Modely jsou reprezentacemi objektů v trojrozměrném prostoru. Jako takové jsou tedy definovány body, tzv. vertexy, v kartézských souřadnicích. Tyto body dále tvoří přímky (edges, hrany), a roviny, tzv. faces, ze kterých se skládá topologie geometrických sítí, které tvoří samotný model. Modelům se tedy často říká pouze „sítě“, správně meshes. S touto nomenklaturou zachází i Unreal Engine, ve kterém jsou modely rozděleny do dvou základních skupin. Static Meshes a Skeletal Meshes.



Obr. 17 – Menu FBX Import

Static Meshes jsou modely, jak již označení napovídá, statické. Takové modely neobsahují data o lotuce, nebo rigu. Nemají tedy žádnou kostru a tím pádem se neanimují. Jde například o rekvizity ve scéně, nábytek, modely domků a podobně.

Skeletal Meshes jsou modely, které obsahují kosti a rig. Jde tedy o virtuální loutky, které mohou obsahovat animační data, a pravděpodobně jsou vyrobeny za cílem reprezentace pohybující se osoby, či objektu.

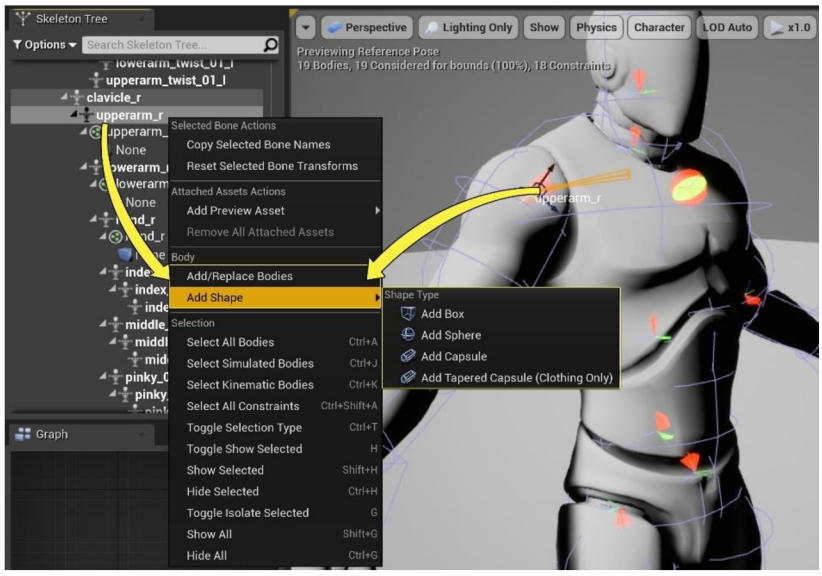
O jaký druh modelu jde můžeme sami definovat při importu modelu do enginu. Při importu souboru s danými příponami, jako je například .FBX či .OBJ, je operátorovi nabídnuto kontextuální okno, ve kterém definuje druh modelu, nastavení importu, zda importovat jeho textury, a jiné detaily, o kterých se budeme bavit při hlubším zkoumání modelovacích nástrojů v UE.

Zmiňované přípony souborů jsou často debatovaným tématem. Pokud jde o hotové modely, Unreal obecně preferuje standardně užívaný (a často nadužívaný) Autodesk .FBX (Kaydara Filmbox). FBX je užitečný především z důvodu datového obsahu, který je schopen uchovat. Na rozdíl od Wavefront .OBJ, FBX exportuje plný rig a animační data. OBJ exportuje pouze čistá data trojrozměrných objektů. To ale také znamená, že FBX

může exportovat i nechtěné chyby. Pokud pracujeme například v softwaru Autodesk Maya a nedaří se nám identifikovat chyba, díky exportu .OBJ jsme schopni vyexportovat pouze informace o trojrozměrné pozici a vlastnostech geometrie, které následně naimportujeme zpět. Tím jsme se zbavili „dat navíc“, a v lepším případě, i nechtěné chyby, například v rigování, kterou se nám nedařilo najít.

Unreal Engine také (do určité míry) podporuje poněkud novější extenzi .ABC (Sony Alembic). Alembic je schopen zastoupení dat FBX, kdy pojme jak animační data, tak samotnou geometrii, ale na rozdíl od FBX tato data navazuje přímo na vertexy (body) modelů. Tím pádem neuchovává původní kostru a rig modelu, a při animaci přímo pohybuje danými body, namísto kostí rigu. Tento přístup nazýváme „point cloud cache“. Proto je alembic používán především na přesun modelů ovlivněných fyzikální simulací. Typicky tedy jde například o oblečení, kožešiny, či vodní simulace – vše objekty, které využijí vertexové animace, tj. animace samotných bodů geometrie. V případě Unreal Engine alembic využíváme především (a prakticky pouze) pro tzv. Grooms, datové soubory pro definici vlasů, chlupů a kožešin.

Mluvíme-li o fyzice, musíme zmínit další nástroj Unreal Engine, tzv. PhAT. PhAT, neboli Physics Asset Tool, je nástroj pro výrobu fyzikálních reprezentací skeletálních sítí za účelem jejich následných interakcí, například s oblečením. PhAT funguje jako podpůrná komponenta pro systém skeletálních sítí, a tak definičně nemůže podporovat fyziku naimportovanou za pomocí dat alembic, které pouze popisují animované transformace bodů v souřadnicích (chápeme, že chybí kosti rigu).

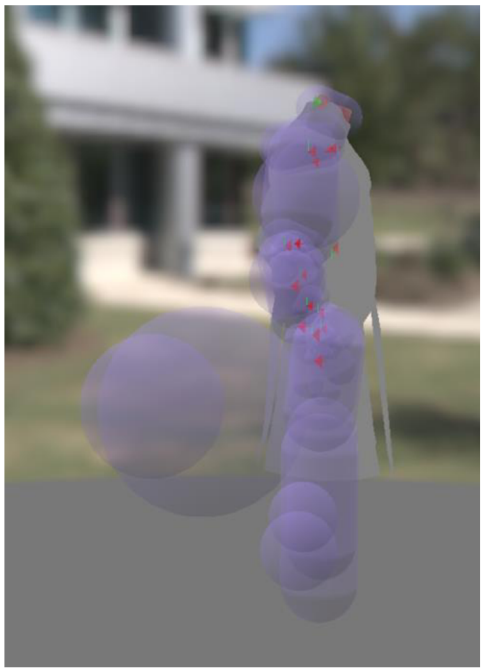


Obr. 18 – PhAT editor v Unreal Engine 4

Ke kostem rigu přiřazujeme tzv. physics bodies, fyzikální tělesa. Tato fyzikální tělesa nabírají geometrické reprezentace základních trojrozměrných těles. Krychle, sféry, kapsle (cylindr s kulatými konci) a zúžené kapsle, tj. kapsle s různými rádii ukončujících sfér. Tyto objekty jsou neviditelnými reprezentacemi fyzických bariér, jako je například lidské tělo, či tělo koně. Zjednodušené reprezentace nahrazují skutečný objekt a tím zjednodušují propočty fyzikálních interakcí, které by za užití skutečné, původní geometrie, byly velice výpočetně náročené. Ovšem, je nutno podotknout, že od verze Unreal Engine 5.3 v kontextuálním menu přibyla možnost „Collide with Environment“, která umožňuje využívat tzv. komplexní kolize, tj. kolize s původními modely, bez využití zjednodušených reprezentací PhAT.

PhAT soubory vyrábíme a upravujeme dle využití a scény. Není tedy nutno, aby fyzikální tělesa stoprocentně kopírovala kostru postavy, záleží především na potřebách animace a fyzikální interakce. Pokud například potřebujeme simulovat interakce kápi jezdce s koněm, ale model koně (a tím pádem rig i PhAT soubor) byl autorován (vyroben) za využití jiné základní rotace, tzn., místo „Front X“ (čelem po ose X) byl vyroben s přístupem „Front Y“ (čelem k ose Y), můžeme tuto situaci řešit aproximací polohy částí těla koně, se kterým potřebujeme interagovat, v rámci PhAT souboru samotného jezdce. Jednoduše vyplníme prostor, který očekáváme, že bude zastoupen koňským hřbetem, reprezentací přímo v fyzikálním souboru jezdce.

Obr. 19 – Fyzikální interakce jezdcovy kápě v praxi



Obr. 20 – PhAT asset jezdce

Systém Skeletal Meshes v Unreal Enginu je založen na tzv. Skeletons. Skeleton je rig dané postavy, ke kterému se vážou animace dané postavy, a se kterými byla postava naanimována. Každá animace se tedy dá úspěšně přehrát (potažmo i naimportovat) pouze za podmínky dodržení stejného rigu po dobu produkce.

V opačném případě Unreal nabízí systém retargetingu animací, tj. modifikace animačních dat tak, aby využívaly jiné kosti, než ty, se kterými byla animace autorována. Je ale dobrým pravidlem se retargetingu v jakémkoliv softwaru vyhýbat a raději vytvářet animace pro specifické rigy.

Skeletony mají další dobrou vlastnost. Při vytváření nového projektu Unreal Engine uživateli nabídne, zda má projekt obsahovat tzv. Starter Content. Jde o balík začátečnických dat, obsahu, na kterém se daný uživatel může učit, či jej využít jako základ pro svůj projekt. Jedním z assetů přítomných v Starter Content je Unreal Engine Mannequin. Tento manekýn, skeletal mesh, je neoficiálním maskotem Epic Games, ale především obsahuje hotový rig. Tento rig se, zaprvé, dá využít jako příklad správného rigu pro herní engine, ale především pro tento rig existuje celá řada hotových animací a animačních balíků na Unreal Engine Marketplace.

Marketplace je virtuální tržiště specificky pro Unreal Engine. Uživatelé zde prodávají hotové assety, modely, či celé systémy pro vývoj jak videoher, tak animované produkce. Ekosystém marketplace je přední výhodou programu Unreal Engine, jelikož místo hodin strávených autorováním již existujícího obsahu, onen obsah můžeme přímo koupit a ve svých projektech jej využívat, za podmínky dodržení licenční smlouvy.

„I Unreal Engine má své vlastní, proprieterní „tržiště“ – Unreal Engine Marketplace. Členové komunity zde mohou prodávat a nakupovat digitální obsah. Krom vysokého podílu pro umělce – při koupi assetu jde 88% z ceny umělci a 12% jako poplatek Epic Games – Epic také často pořádá slevy a výprodeje.

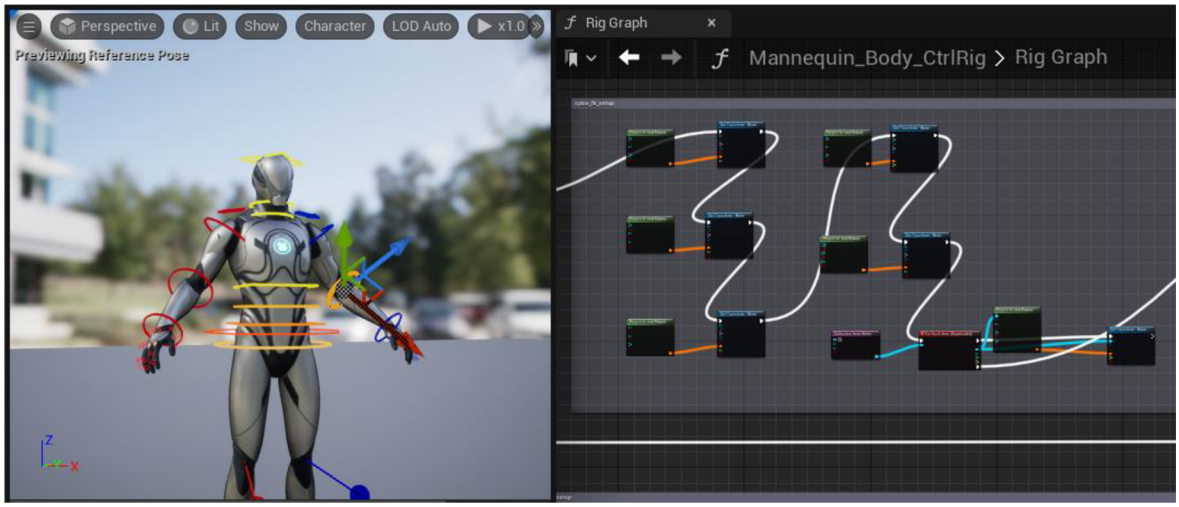
K tomu, každý měsíc Epic Games vyberou několik assetů, které onen daný měsíc budou zdarma. Často se jedná o velké a drahé balíčky assetů, které mohou nezávislým vývojářům pomoci při výrobě jejich projektů a koníčkářům přinejmenším udělají radost.“ (Honal, 2022)[24]

[24] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]

Propojenost Unreal Engine Marketplace s samotným enginem je sama o sobě impozantní. Zakupovaný obsah je již přímo vytvořen pro účely využití v Unreal Engine, a tím pádem nemusíme nic konvertovat, přepisovat či jakkoliv měnit. Po zakoupení je obsah okamžitě připraven pro import do enginu jedním kliknutím.

Animace pro postavy můžeme tedy sami autorovat, či koupit na Marketplace. Existuje ale ještě další možnost. V Unreal Engine se dají modely i přímo animovat a hotové animace se dají upravovat za pomocí Control Rig.

Control Rig je funkce Unreal Engine, která dovoluje napamovat virtuální in-engine ovladače na existující rig z exerního programu. Pomocí jej můžeme přímo postavy animovat, či jejich hotové animace upravovat. Jde o nejvýhodnější a nejefektivnější workflow, kdy jakékoliv změny provádíme přímo v enginu, čímž se vyhýbáme reimportu animací a s tím možným komplikacím a chybám.



Obr. 21 – Control Rig Graph v Unreal Engine

Pokud jde o animace tváří, tzv. facial animation, tu řešíme buď klasicky, manuálním klíčování, nebo můžeme využít plug-in LiveLink, který je nativně dostupný v Unreal Engine. LiveLink nám povoluje editor přímo propojit například s kamerou či telefonem, pro ilustraci například s iPhone a Apple ARKit, který snímá pohyby tváře a přímo je převádí na rig přítomný v Unreal Engine. Tomuto přístupu říkáme „Face Capture“.

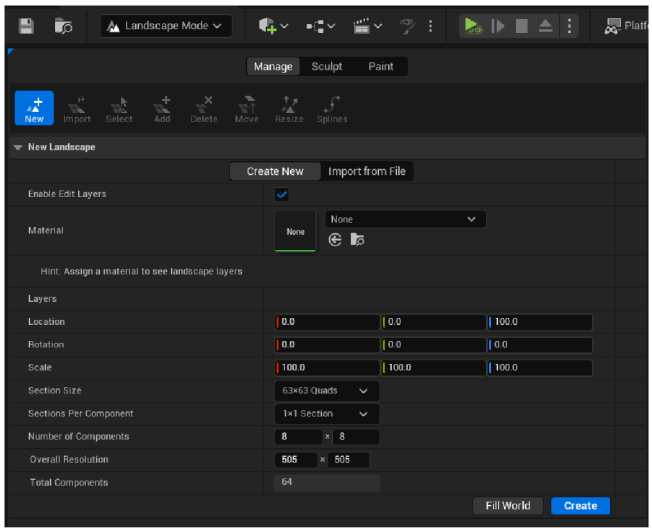
Face capture má prozatím mnohé mouchy, především pokud používáme bezdrátový přenos dat za pomocí Wi-Fi a pouhá videodata z telefonu. Můžeme si všimnout časové prodlevy v přenosu dat a také „sekaných“ pohybů. Ale v profesionálním prostředí, kdy

bychom měli k dispozici například profesionální Rokoko Headcam a herce, pro zachycení emocí na fotorealistické tváři, není nad herecký výkon. Jako u jiných forem face capture, i zde je nutné pohyby a grimasy přehánět.

Obrázek 22 a 23 – Testování facial capture za pomocí iPhone, LiveLink a Apple ARKit

Postavy a rekvizity zasazujeme do propracovaných trojrozměrných prostředí. Pro jejich výrobu můžeme využít buď klasického 3D modelu, což je varianta, ke které bych se sám osobně přikláněl, nicméně také můžeme využít systém tzv. Landscapes, který povoluje in-engine sculpting vlastních rozlehlých terénů.

Tvorbu landscapes zajišťuje tzv. Landscape editor, který, podobně jako ostatní editory v Unreal Engine, je podrozdělen do specifických pracovních modulů. Tyto moduly zahrnují sculpt mode, který operátorovi umožňuje sculpting (tj. „sochařství“) terénu pomocí různých virtuálních „štětců“ (buď jde o 2D reprezentace základních geometrických tvarů, či přímo o vlastní textury, které určují, erozi, výšku terénu apod – tzv. heightmapy), manage mode, který dovoluje přidávání, či odebírání dalších terénů, či existujícího terénu ve čtvercové konfiguraci (podobně jako je tomu u Blizzard world ADTs), paint mode, který spolu s materiálovým systémem a specifickými nódy pracuje jakožto texturovací nástroj, podobně jako například Substance Painter, tzn. dovoluje uživateli „malovat“ terén skalami, horami, či hlínou. Poslední významný modul je modul lanscape splines, který se dá přirovnat k nurbs-curves například v softwaru Autodesk Maya. Jde tedy o nástroj, který pracuje s vektorovými křivkami, které definuje autor. Tyto křivky následně deformují dané prostředí, daný landscape, čímž povolují perfektní zasazení cestiček, či silnic do scény.



Obr. 24 – Landscape Editor v Unreal Engine

S zmiňovaným „malováním“ terénu, tj. definice povrchů prostředí, nám pomáhají, resp. jej zaručují, tzv. materiály. Materiály jsou sady instrukcí pro GPU, které určují vzhled a vlastnosti daného povrchu. Často se setkávám se záměnou termínů „shader“ a „materiál“, cítím tedy nutnost tyto termíny odlišit a vysvětlit jejich rozdíly.

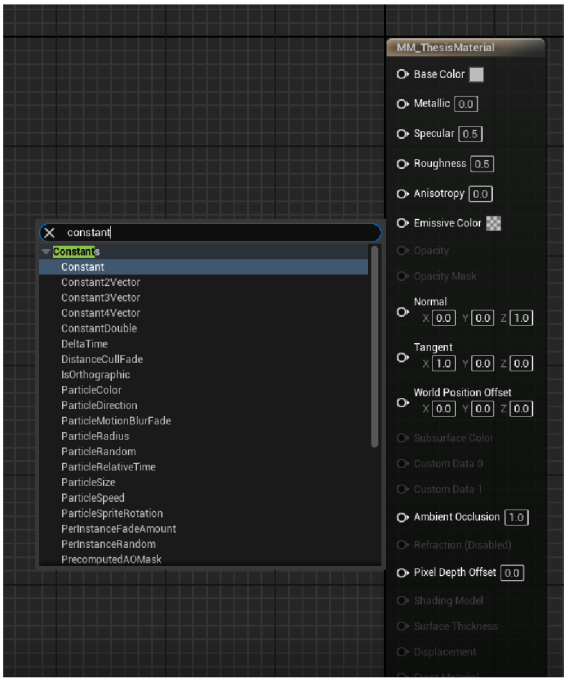
Shader je počítačový program, který nativně využívá grafickou kartu pro definice interakcí světla s povrchy materiálů. Naproti tomu, materiál pouze určuje vlastnosti daného materiálu. Materiál také určuje jeden konkrétní shader, který má použít, a tím pádem, shader definuje instrukce dostupné danému materiálu. Vztah shader-materiál se tedy dá (velice zjednodušeně) chápat jako vztah rodič-dítě (v kontextech herních enginů chápeme jako vztah idea-objekt, či „transcendentální vzor“ a „jeho reálná fyzická invokace“). Představme si, že máme scénu s modelem auta. Materiál řekne, že karosérie auta je metalická (PBR metallic), s hladkým povrchem (PBR rough) a je červená. Shader umělci dává k dispozici instrukce, jako je např. vektorový parametr pro určení barvy, Texture Sampler pro definici texturových dat, či skalární parametry a možnost násobení pro jejich úpravu, a dále zajišťuje, že interakce světla s tímto povrchem správně zobrazuje lesk, stíny, průhlednosti a jiné vlastnosti.

Pokud tedy pracuji s materiálem, pracuji potažmo i se shaderem a tím pádem s instrukcemi, které mi daný shader zajišťuje. Pracuji-li se shaderem ThinFilmIridescence (specifický shader inspirovaný přírodním fenoménem stejného názvu), mám k dispozici funkce a instrukce, které zajišťují iridescenci materiálu.

Nicméně, pokud vytvářím nový materiál, nevytvářím nový shader! Tvorba shaderu je komplexní, matematicky a logicky náročná činnost, kdy grafický inženýr, či programátor, sám programaticky definuje instrukce pro GPU. Jde tedy o přímou výrobu nástroje, který umělec dále používá.

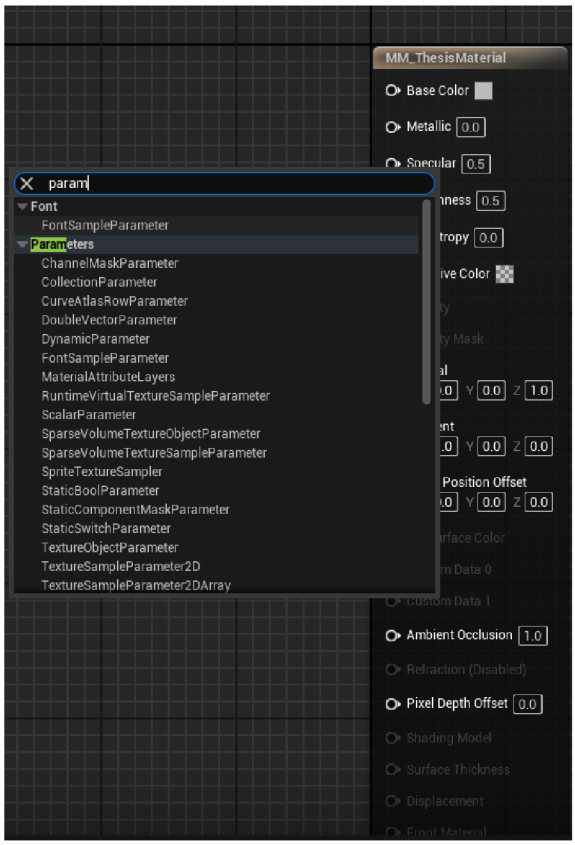
Nástroje pro manipulaci materiálů v rámci UE jsou jedním z nejdůležitějších částí enginu, a také jedním ze zásadnějších rozdílů mezi tradiční offline workflow a specifickou workflow Unreal Engine. Unreal totiž pracuje s takzvanými Master Materials a Material Instances. Jejich využití je specifické, avšak základním pravidlem je, že správná definice parametrů v Master Materiálu umožňuje jednoduchou parametrizaci Materiálových Instancí.

Totiž, každý materiál v prakticky jakémkoliv 3D softwaru, jak jsme si již definovali, je souborem instrukcí pro následný rendering povrchů, ke kterým onen materiál přiřadíme. Obecně platí, že tyto instrukce jsou statické, tj. neproměnné v čase, pokud na tom sami netrváme a materiál jinak neurčíme. Tedy, můžeme jej chápat jako sbírku konstantních hodnot v dané matematické rovnici.



Obr. 25 – Screenshot nabízených konstant v rámci UE Material Editoru

Nicméně Unreal Engine, jakožto herní engine renderující v reálném čase, má tu výhodu, že tyto hodnoty umí parametrizovat – tzn. proměnit hodnoty konstantní v hodnoty proměnné. Těmto proměnným říkáme parametry.

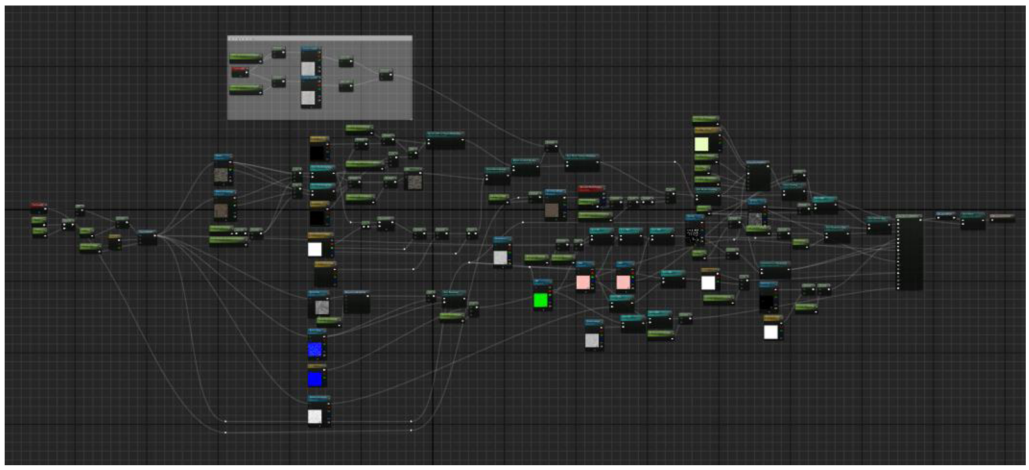


Obr. 26 – Screenshot dostupných parametrů v rámci UE Material Editoru

Parametrizace materiálů je často (a špatně) přehlížena. Parametrizovat se totiž dá prakticky cokoliv. Od textur, po skalárních intenzit, vektorové parametry, dokonce i změna samotného shaderu se dá parametricky ovládat.

Síla parametrizace spočívá v možnostech okamžitých změn pro art direction například všech objektů ve scéně, okamžitě, a ihned, bez zdlouhavých změn pod každým jedním materiálem každého jednoho objektu.

Pro představu, material instances se dají využít jako nástroj pro zrychlení výroby materiálů, dejme tomu, dřevěných objektů, kdy při výrobě dřevěného master materiálu parametrizujeme dané textury. Tím pádem, objekty, s různou topologií a různými texturami pořád sdílejí stejné vlastnosti dřevěného materiálu, a jedinou změnou je parametrická záměna textur každého objektu. Tyto změny aplikujeme přímo v material instancích, kdy každý objekt, který vyžaduje vlastní parametrické změny, získá vlastní material instance. Parametrizaci ale můžeme využít i zásadněji.

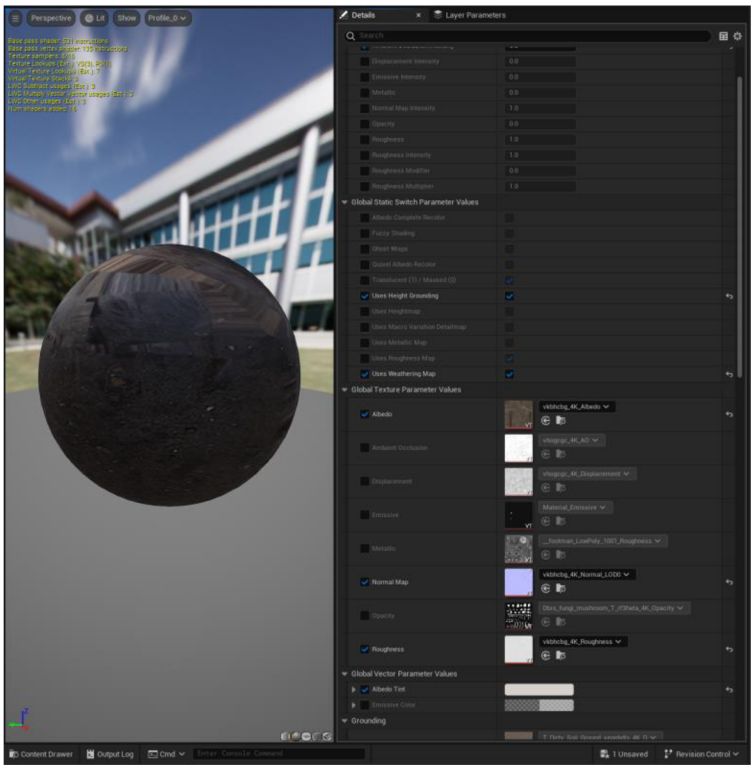


Obr. 27 – Ubermaterial pro projekt Darrowshire

Dejme tomu, že vytváříme město. Město stavíme z mnohých budov, zastávek, odpadních košů, vozidel a podobně. Pokud jsme důvtipní, a produkce nám dovoluje jistou „vatu“ ve výkonu a v času určenému pro render (nejde tedy o hru nebo jiný interaktivní obsah), adoptujeme workflow tzv. Ubermateriálu. Ubermateriál je Master Materiál, který využívá celá scéna pro prakticky všechny objekty. Hierarchická posloupnost takového Ubermateriálu je následující:

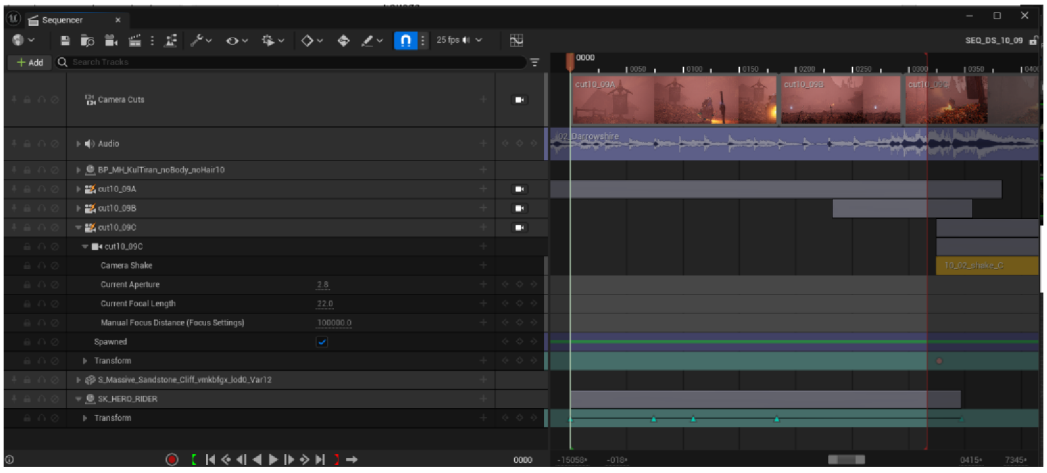
Master (Uber) Materiál > Master Material Instance > Material Instances

Tedy máme jeden Master (Uber) Materiál, kterému jsme přiřadili tzv. Master Instance – instanci, která bude parametricky ovlivňovat všechny své další instance, tedy instance této instance. Výsledných materiálových instancí může být neomezené množství. To znamená, že pokud po rozložení scény a aplikaci onech instancí vůči všem objektům (protože, jak jsme si řekli, s dostatečně zdravou logikou můžeme parametrizovat prakticky cokoliv), se nám vzhled objektů ve scéně nelíbí, můžeme jednoduše a v reálném čase měnit parametry Master Instance, a tyto změny se okamžitě propisují do všech instancí této Master Instance. Tím můžeme art directovat vzhled celé scény, okamžitě, bez prodlevy a bez čekání na úpravy různých samotných materiálů. Jedinou podmínkou je správná parametrizace a její plánování při výrobě původního ubermateriálu.



Obr. 28 – Material instance Ubermateriálu v projektu Darrowshire

Material Instances samozřejmě nevyužíváme pouze pro rychlý a pohodlný art direction. Čtenáře již určitě napadla možnost proměnné hodnoty měnit v čase pro účely animace. Parametry material instances skutečně animujeme a to za využití animačního nástroje nativního v Unreal Engine, tzv. Sequencer.



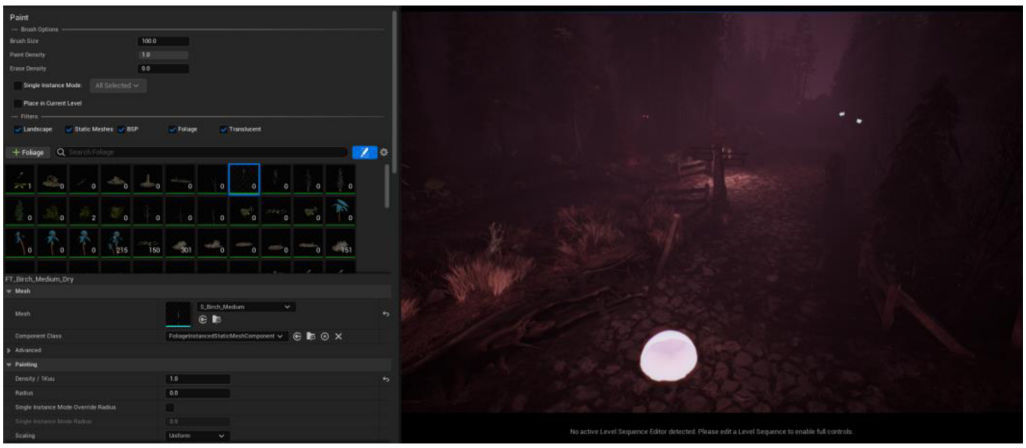
Obr. 29 – Náhled hlavního okna Sequencer v Unreal Engine

Sequencer je animační, či, chcete-li, režisérský nástroj, který propojuje možnosti non-lineárního editingu, či střihu, jako je tomu například u softwaru Adobe Premiere či Blackmagic DaVinci Resolve, s plně funkčním klíčovacím animátorským přístupem.

Pomocí tzv. Level sequences animujeme postavy, světla, objekty, či jakékoliv jiné součásti naší scény (=levelu, herní nomenklatura). S těmito level sequences následně pracujeme v tzv. Master sequences, které slouží jako kolekce level sequences, ve kterých můžeme sekvence volně, tzv. non-lineárně stříhat, proměňovat a měnit. Level sequences tedy chápeme jako tradiční záběry (analogicky k "shots" v offline produkci). Dále pracujeme s tzv. takes. Takes jsou reprezentacemi záběrů (shots), se kterými ale můžeme experimentovat a vytvářet jejich různé verze bez toho, anichž bychom ovlivňovali původní level sequence. Jde tedy o non-destruktivní přístup.

Sequencer také uživatele enginu nabádá k střihu a plné postprodukci audiovizuálního obsahu přímo v enginu, namísto jeho exportu a další editace v tradičních kompozičních programech, jako je Nuke, After Effects či Resolve. To ovšem neznamená, že kompoziting je nemožný, ačkoliv byl až do nedávna velice problematický. Toto téma prozkoumáme v dedikované kapitole.

Takzvaný scattering, tj. rozmístění různých objektů po scéně, především tedy porostu, je v UE zajištěn hned několika systémy. Prvním systémem, který nabízí precizní a přesnou kontrolu, ale je časově náročný a velice závislý na dovednosti samotného operátora a art directora, je tzv. foliage mode. Foliage mode slouží pro manuální rozmístění květin, stromů, kamenů a jiných statických objektů (tedy statických sítí) za pomocí ručního určení „hrubé lokace" výskytu onech objektů pomocí virtuálního štětce v podobě transparentní sféry. Krom této vizualizace se výskyt objektů řídí i dle pravidel přítomných v kontextuálním menu, jako jsou například density (hustota), radius (pomyslný poloměr prázdného prostoru kolem každého jednoho objektu za cílem vzájemného překrytu), či scaling (velikost). Kontextuální menu nabízí mnoho možností jak art directovat objekty vložené přes foliage mode.



Obr. 30 – Foliage mode v projektu Darrowshire

Všechny objekty vložené pomocí foliage mode spadají pod, zatím nezmíněný, třetí druh geometrických sítí přítomných v Unreal Engine. Jde o tzv. Instanced Static Meshes, instancované statické sítě.

Instance modelů nejsou nový koncept. Pracují analogicky stejně, jako například v softwaru Maya, či Blender. Základní myšlenka instancingu modelů je sdílení stejných vlastností a tím pádem eliminace nutnosti explicitního processingu všech stejných vlastností modelů ve scéně.

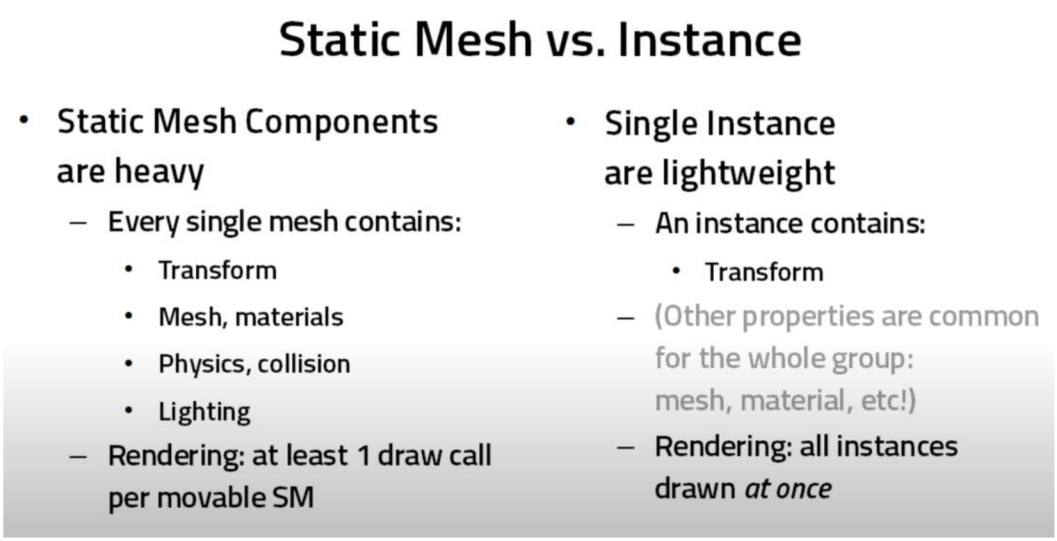
Řečeno lidsky, každý model vložený do scény obsahuje data o vlastních vlastnostech. Tyto vlastnosti zahrnují transformaci (poloha, rotace, velikost), identifikaci sítě a materiálů, fyzikální a kolizní (collision) vlastnosti a vlastnosti interakcí se světlem ve scéně (nezávislé na materiálu, například „Má stín / Nemá stín“). Nejdražší z těchto vlastností je samotná síť a její materiály.

„Komunikaci mezi procesorem a grafickou kartou nazýváme tzv. draw calls. A právě zde přicházíme k prvnímu kameni úrazu real-time renderingu. Jakákoliv instrukce od centrálního procesoru, řečeno extrémně zjednodušeně, nás „stojí“ tento draw call. Příkladem takového draw callu je materiál. Každý materiál na každém objektu je jeden draw call. V praxi toto znamená, že pokud chceme renderovat krychli, která má na každé své straně jeden materiál, stojí nás to šest draw calls.“

S adventem Nanite, a tedy od verze Unreal Engine 5.0.0 (především díky novému RHI, způsobu komunikace CPU, operačního systému a grafické karty DirectX12) již materiály sdílí draw call s celou renderovanou scénou. Se všemi identickými objekty identických materiálů je zacházeno jako s jedním draw call. I tak, pokud bychom

ignorovali instancing, museli bychom zbytečně zapisovat do paměti a v ní udržovat neúměrné číslo duplicitních vlastností.

Instance statických sítí obsahují pouze informaci o transformaci (poloha, rotace, velikost) každé dané sítě, a to proto, že všechny ostatní vlastnosti se sdílí.



Obr. 31 – Slide z prezentace Tech Art Aid, UE4 Optimization: Instancing[25]

Tento přístup využíváme pokaždé, kdy potřebujeme vyrenderovat velké množství identických objektů. Společně s Nanite jde o velice efektivní a rychlý způsob vykreslování.

Druhým nástrojem pro scattering lesů, trav a porostu je Procedural Foliage Tool. Procedural foliage tool je mým preferovaným nástrojem pro populování světů hustými porosty. Procedural Foliage Tool pracuje s tabulkami, které v enginu definujeme. Tyto tabulky obsahují seznam všech druhů stromů, květin, či čehokoliv, čím chceme scénu zaplnit. Každý tento seznam je tvořen individuálními datovými soubory, které také definujeme, ve kterých určujeme danou geometrickou síť a vlastnosti, které emulují chování a vzhled rostlin v přírodě, jako je disperze semínek, či věk porostu.

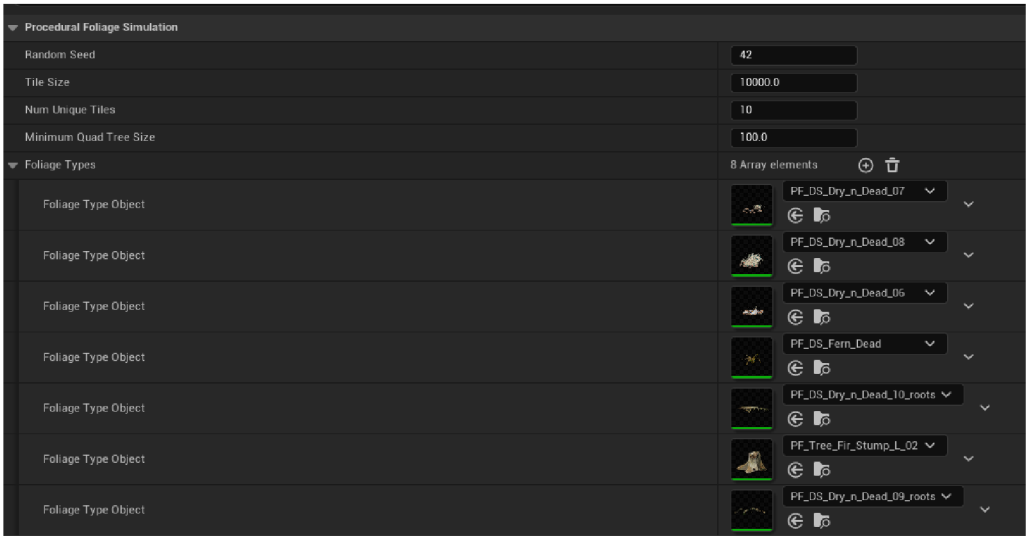
Tyto tabulky aplikujeme pomocí Procedural Foliage Volumes (PFV). Jde o 3D objemovou reprezentaci prostoru, který hodláme porostem zaplnit. Tyto reprezentace tvoříme přímo v enginu za pomocí BSP (binary space partitions) toolsetu. BSP je poněkud zastaralá metoda geometrie, která rozděluje prostor do dvou částí pomocí rovin (faces). Tento proces se opakuje rekurzivně (zpětně), čímž se prostor rozděluje na stále menší a menší části. V kontextu PFV se bavíme především o jednoduchých trojrozměrných

[25] Tech Art Aid. *UE4 Optimization: Instancing* [YouTube video] [Citace: 19. Květen 2024]

objektech, jako jsou krychle. Krom aditivních PFV máme také k dispozici PFBV, Procedural Foliage Blocking Volumes. Jde o přesný opak, tedy o reprezentace prostoru, který nechceme porostem zaplnit. Typicky je využíváme například pro definici vychozené cesty, jde tedy o subtraktivní nástroj.

Obr. 32 – Fialové sítě jako reprezentace Procedural Foliage Blocking Volumes

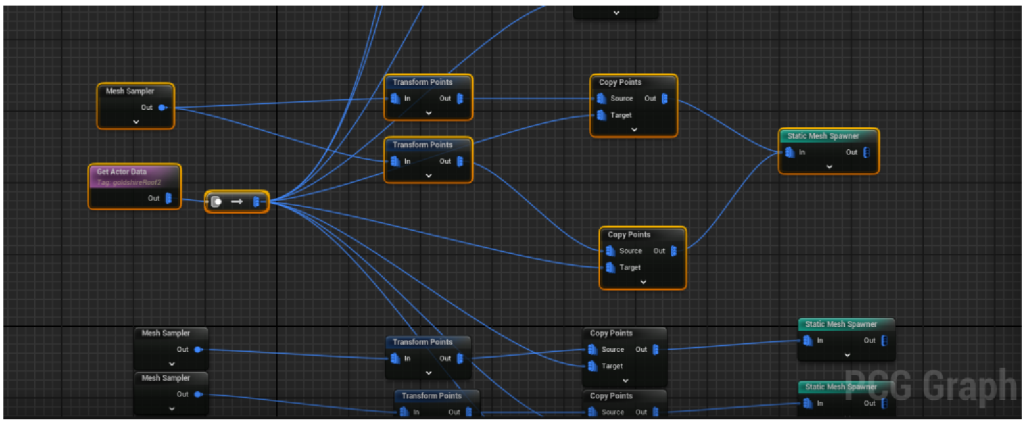
Jak napovídá název, proces je plně procedurální. V praxi tedy vyplníme tabulku a zmáčkneme ono přemocné tlačítko „Generate“. Dle komplexity scény, přesahu blocking volumes a výkonu počítače čekáme dokud není svět patřičně porostlý. Je nutné zmínit značnou výpočetní cenu tohoto procesu. Vlastnosti jako jsou disperze semínek zaručují naprosto autentický a přírodní vzhled generovaného výsledku. Dále, onen výsledek se dá manuálně upravovat pomocí Foliage Mode, do kterého se automaticky přidají všechny objekty, které PFT vytvoří.



Obr. 33 – Menu ProceduralFoliageSpawner

Posledním nástrojem pro scattering objektů je PCG, Procedural Content Generation Framework. Procedural Content Generation Framework je sbírka nástrojů, přirovnat ji můžeme k Geometry Nodes, které známe ze softwaru Blender. PCG pracuje s tzv. points – body. Tyto body generujeme za pomocí logiky, kterou vytváříme v PCG Grafech (PCG Graphs). Typicky body vytváříme dle topologie modelů, logická úvaha může být například následující.

Mám model domu. Jeho střecha je pouze jednoduchým kusem geometrie, nemá žádný detail ani jiné zajímavosti. Toto bych rád změnil, a střechu zaplnil šindely, jehož model již mám, ale pouze jeden, samostatný. Nechci toho ale docílit manuálně, protože na to nemám čas, a nebo naopak, chci čas trávit co možná nejefektivněji, zvláště, pokud vím, že mohu využít PCG.



Obr. 34 – PCG Graph vytvářející šindely pro střechu

Dle existující geometrie povrchu střechy nechám PCG vygenerovat body, na jejichž koordinátech bude PCG populovat šindely. PCG také nabízí mnohé parametry tvorby daných bodů, což povoluje velice slušný artdirection scén. Díky různým specifickým parametrům můžeme pomocí PCG procedurálně vytvářet celé scény.

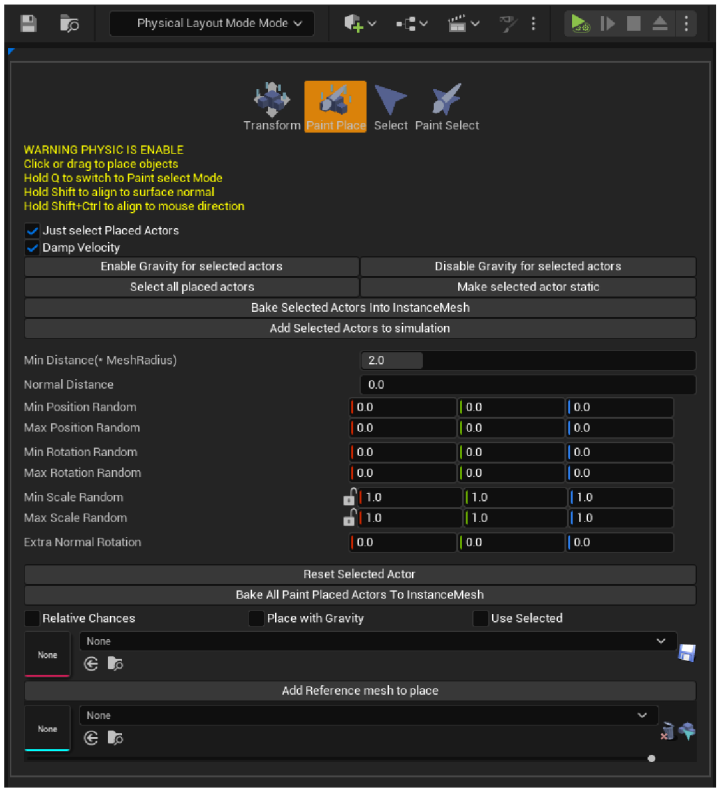
Podobně jako u animací, i samotné objekty, kterými hodlám scény vyplňovat, se dají pořídit na Marketplace. Ale nespěchejme, protože v rámci Unreal Engine je od verze 5.0.0 nativně zabudovaný Quixel Megascans Bridge. Jde o rozsáhlou knihovnu modelů, materiálů a rekvizit, které můžeme zdarma využít v našich scénách.

„Quixel byl založen v roce 2011 Teddy Bergsmanem a Waqar Azimem. Jejich vizí bylo zrychlení produkce virtuálních prostředí pomocí obrovské knihovny 3D

naskenovaných objektů, které by umělci následně mohli využívat ve svých scénách. Tak vznikla, dnes již notoricky známá knihovna, Quixel Megascans.“ [26](Honal, 2022)

Samotné nástroje pro scattering, a různé jiné systémy, se také dají na Marketplace pořídit. Ve formě plug-ins, tedy dodatků, které doplňují, přidávají, či mění chování enginu se nabízí mnohé nástroje. Po straně scatteringu si dovolím zmínit můj nejoblíbenější nástroj pro tuto činnost, tzv. Physical Layout Mode.

Physical Layout Mode je plugin, se kterým můžeme scatterovat objekty za pomocí fyzikálních interakcí. Pokud tedy chceme vyrobit například realisticky vypadající hromadu štěrku, jejíž manuální výroba by zabrala celé hodiny a především by přestavovala velice frustrující, nudnou a repetitivní aktivitu, využijeme Physical Layout Mode.

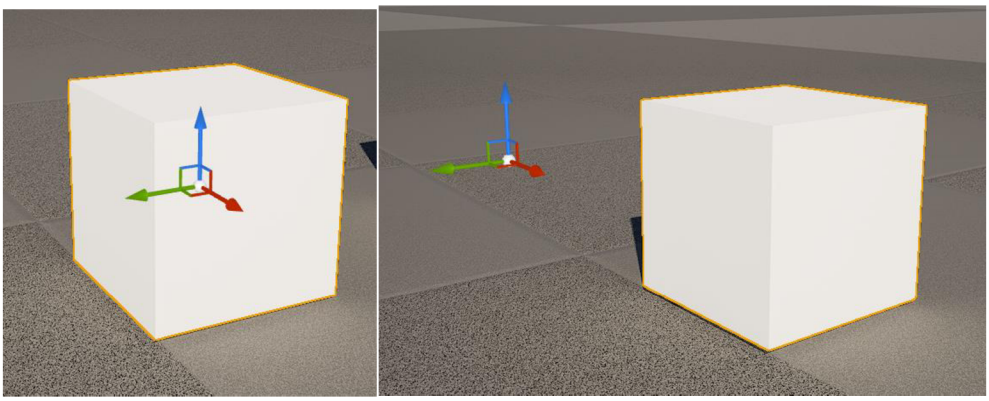


Obr. 35 – Physical Layout Mode

Pokud bychom měli již hotovou scénu se scatterovaným porostem v externím programu, jako je například Maya a tuto scénu bychom chtěli pouze transplantovat do Unreal Engine, stojí za to se zamyslet, zda tento postup dává smysl. Totiž, k vytvoření scény jsme nevyužili nástroje k tomu určené v rámci UE, tedy nástroje, které vývoj zrychlují, a zároveň, ačkoliv scéna jde jednoduše přesunout mezi UE a Autodesk Maya,

[26] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 19. Květen 2024]

postup tohoto přesunutí vyžaduje vynulování nenulových transformací objektů ve scéně. Totiž, herní enginy počítají s objekty jako s modulárními kusy, ze kterých se dále scéna staví. Pokud naimportujeme již existující scénu, kde tyto objekty jsou již rozloženy, tzn., nemají nulové koordináty, pokud chceme jejich polohy a transformace zachovat, musíme tyto nenulové koordináty považovat za jejich nulovou hodnotu. Tedy, pokud se strom nachází na X: 20, Y: 20, Z: -2, budeme tyto koordináty nově považovat za nulové, a strom tím pádem zůstane na svém místě. To ale znamená, že jakákoliv následná úprava je velice problematická a vyžaduje znovu vynulovat tzv. pivot point, tedy polohu v lokálních kartézských souřadnicích. Toto není problém s jedním objektem, ale pokud se bavíme o scéně se stovkami či tisíci objekty, pouze tyto úpravy budou časově náročnější, než celou scénu rovnou nechávat v Maya a vyrenderovat ji offline například za pomocí rendereru Arnold.



Obr. 36 (na levo) – Objekt s vynulovaným pivot point

Obr. 37 (na pravo) – Objekt bez vynulovaného pivot point

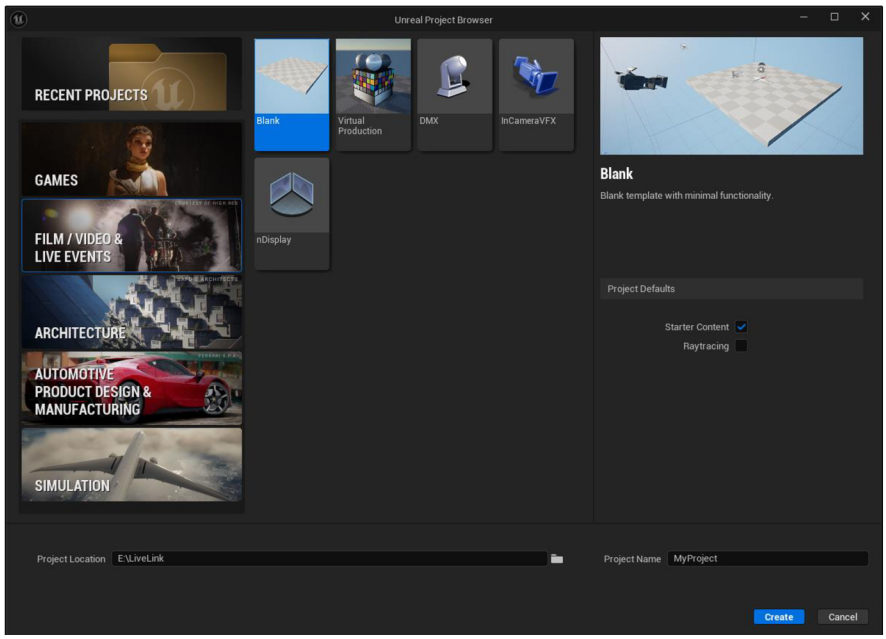
Možná překvapivým nástrojem pro animační práci s Unreal Engine jsou tzv. Blueprints, původně Kismet, či UnrealScript. Jde o systém vizuálního skriptování, chtete-li „zjednodušeného programování za pomocí nódových systémů“ (ačkoliv termínu „programování“ bych se v kontextu blueprints vyvaroval). Pomocí nódů, které reprezentují specifické akce, je blueprinting pravděpodobně hlavním důvodem popularizace UDK (Unreal Development Kit), UE3 a UE4. Díky němu můžeme naskriptovat různé akce, herní mechaniky a obecně logiku, ať už hry, či skriptované sekvence, např. pro krátký film, bez znalosti programovacího jazyka, jako je C++.

Opomineme-li propojení a využití s animačními nástroji Unreal Engine, mají Blueprints ještě další skvělou vlastnost. Blueprints mohou obsahovat reprezentace 3D modelů ve vlastním lokálním 3D prostoru, separátním od 3D prostoru scény. Tedy,

můžeme vytvořit blueprint „scénu“ se specifickou skupinou modelů, kterým následně populujeme svět naší hlavní „hlavní“ scény. Tento přístup k worldbuildingu značně zrychluje lookdev scén, a zároveň za propojení materiálového editoru a blueprintů, můžeme vytvářet specifické nástroje pro art direction daných objektů.

Pro hlubší pochopení musíme nejdříve uchopit hierarchii práce s Unreal Engine.

3. PROJECT MANAGEMENT V UNREAL ENGINE

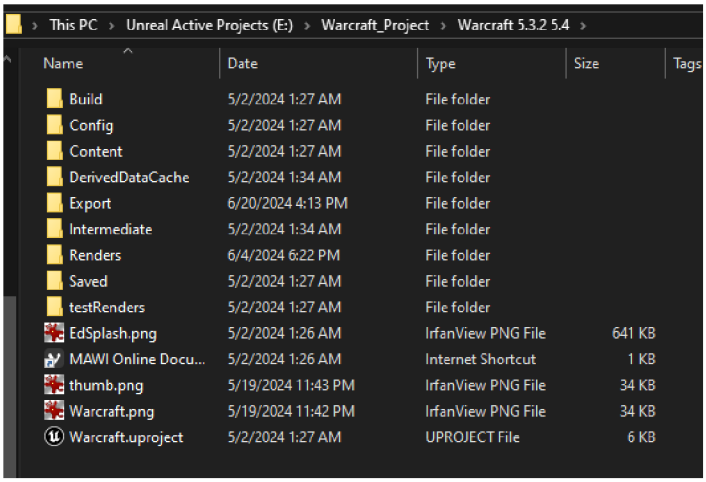


Obr. 38 – Unreal project browser

Software pro výrobu 3D assetů jako je Autodesk Maya či Blender ve svých základních, neupravených verzích pracují na bázi individuálních souborů, kterým říkáme scenes (scény). Obsah těchto scén buď vyrábíme sami přimo v dané scéně, nebo ho do scény importujeme, typicky z repozitáře určeného k vývoji a úložišti daného projektu, pod který scéna spadá. K tomu využíváme tzv. reference, kdy pracujeme s celkovým obsahem dané scény, ale nemáme možnost ji editovat, čímž se vyvarujeme nechtěným chybám a zaručujeme pružnost práce s projektem. Management takových repozitářů zajišťujeme například pomocí softwaru OpenPYPE. Takový software zajišťuje propojenost a především centralizaci projektů, přenos souborů mezi softwary, které využíváme a tím zajištění kompatibility. V případě Unreal Engine, a herních enginů obecně, tuto roli z velké části zastupuje engine samotný.

Namísto scén Unreal pracuje s tzv. project files, tedy projektovými soubory. Samotné scény spadají pod onen projekt a říkáme jim „levely“, dle nomenklatury her. Nicméně pro uchování konzistence je nadále nazývejme scénami. Project file konfigurujeme a nastavujeme při prvním otevření programu Unreal Engine. Při spuštění jsme konfrontováni s výběrem z pěti konfigurací projektu, dle našeho plánovaného využití. Za léta vývoje byl Unreal Engine adaptován k různým případům užití, čímž vznikl celý ekosystém plug-ins (volitelné dodatky přidávající, či upravující funkce engine) a specifických nastavení. Pokud by každý projekt využíval všechny plug-ins a volitelné konfigurace, byl by zbytečně a neefektivně obrovský, což už i tak je jedna ze slabin Unreal Engine. Tedy, vybereme si jednu z pěti konfigurací. Tyto konfigurace se dají měnit během vývoje projektu pod kontextuálním menu Project Settings, tudíž nejde o destruktivní, „locked-in“ systém, který se již nedá měnit.

Předkonfigurovaná nastavení projektů zahrnují herní konfiguraci, konfiguraci pro video a eventy (např. koncerty), architekturu, automotivní produkt design, a simulaci. Při nastavování konfigurace nás uživatelské rozhraní také vyzve k nastavení adresáře celého projektu. Tímto adresářem rozumíme místo na disku, kde bude uložen samotný projekt Unreal Engine a jeho importované assety. Zároveň ale nejde o repozitář, kde musejí být uloženy i assety před importem, jako například .FBX soubory vyrobené v Blender.

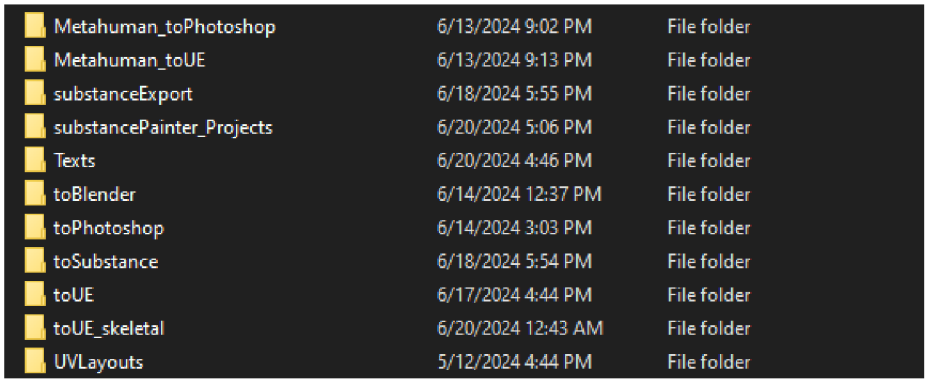


Obr. 39 – Adresář Unreal projektu Darrowshire

Totiž, herní enginy obecně, a tedy i Unreal Engine, přepisují užitečná data, jako například .FBX soubor popisující geometrickou síť, do vlastních datových rozhraní, v případě Unreal Engine jde o tzv. Uassets s příponou souboru .uasset. Pokud tedy v softwaru Blender vyrobím 3D model, který vyexportuji pro následné využití v herním

enginu, při importu do daného enginu onen engine model vezme a, pro nedostatek lepších slov tento model „zduplikuje“, či vytvoří vlastní reprezentaci, která bude nadále uložena v repozitáři projektu engine. To ovšem neznamená, že samotný exportovaný model nemá své využití.

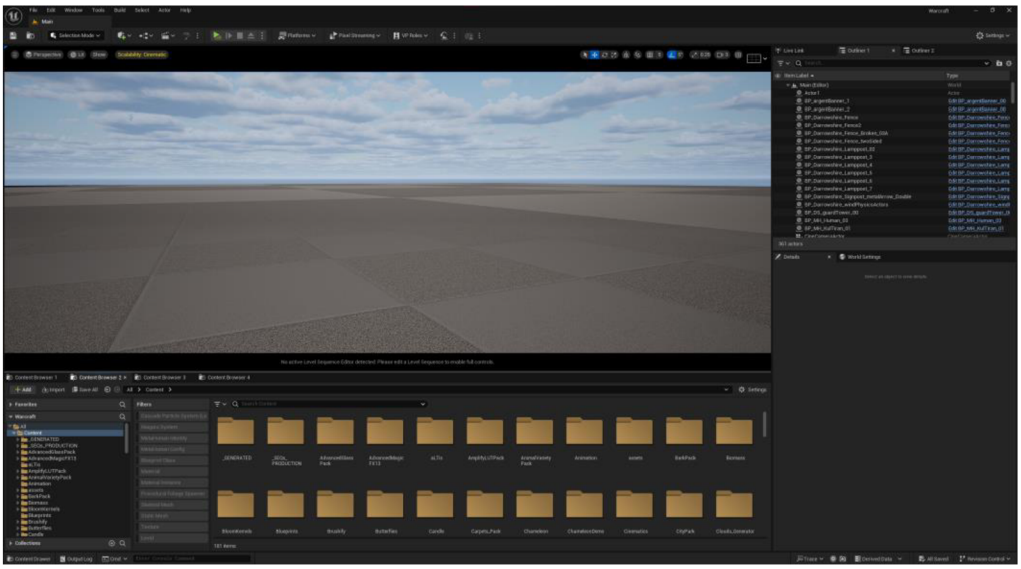
Takovému modelu se v této situaci říká tzv. Source Asset. Zdrojový asset, zdrojový soubor. Engine zná cestu k němu a proto je správné vést si i samotný pracovní repozitář assetů, které pomyslně „posílám“ z 3D programu (či jiného pracovního programu, např. Photoshop) do herního enginu.



Obr. 40 – Workdir projektu Darrowshire

Management právě tohoto „posílání“ (tzv. publish) assetů zajišťují buď interní nástroje, nebo nástroje jako již zmiňovaná OpenPYPE, v rámci UE AYON. Bohužel, tyto nástroje pro integraci Unreal Engine zatím nejsou pro produkci připraveny, a tudíž se musíme spoléhat na staré dobré „workdir“, kam manuálně modely ukládáme. To vyžaduje vysokou míru organizace.

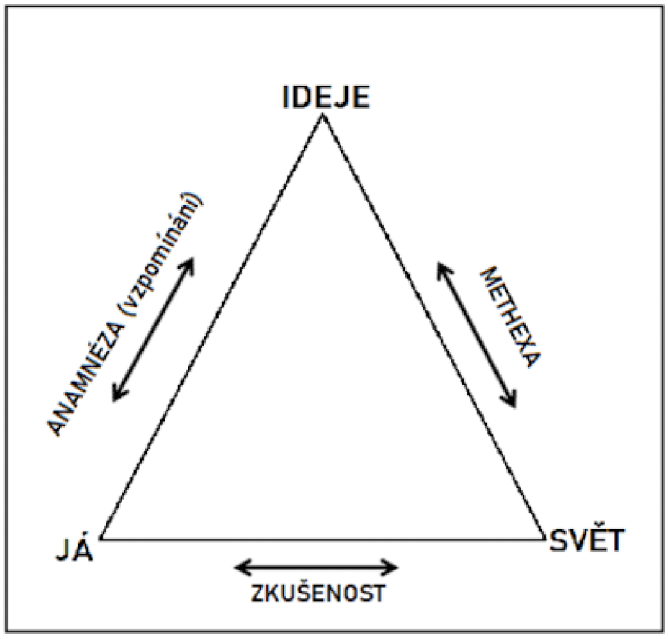
Organizace není pouze zájmem digitální a mentální hygieny, ale je také užitečná z čiře praktických důvodů, a to kvůli iteraci práce. Například při iterací modelů se často setkáváme s, někdy až desítkami verzí, toho samého modelu. Rozumíme tedy dům_v000, dům_v001, a podobně. Pokud jsme již onen domek zakomponovali do scény a tedy se rozumí, že jej nechceme pokaždé manuálně vyměňovat, posouvat, a celkově jej do scény vždy ručně přidávat, využíváme toho funkce tzv. reimport. Předbíháme.



Obr. 41 – Náhled Unreal Engine Editor a Folder View

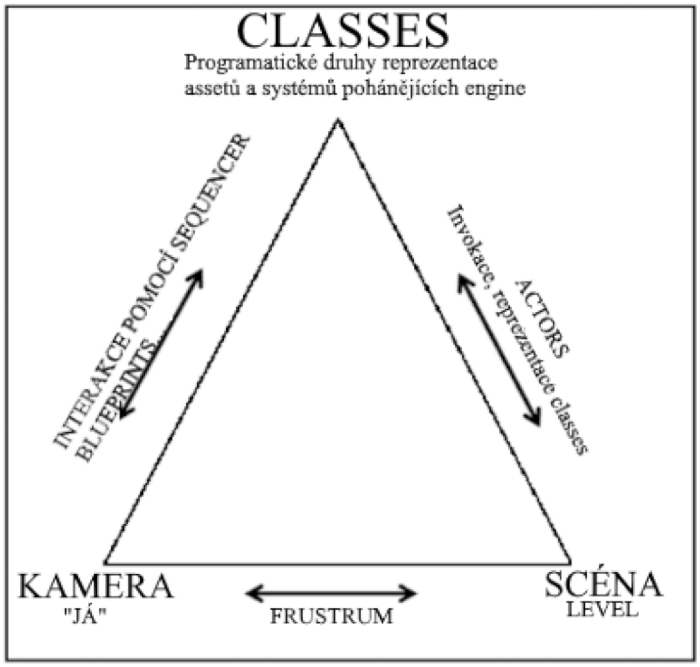
S projektovými soubory, tedy daty, se kterými Unreal Engine pracuje, které jsou součástí samotného úložitště projektu, rozumíme tedy .uassets, pracujeme za pomocí tzv. Folder View v rámci tzv. Content Browser enginu. Jde tedy o prohlížeč a sadu nástrojů, se kterou ovládáme projektové soubory Unreal Engine.

Paradigma herních enginů můžeme přirovnat k paradigmatu platonského trojúhelníku. Ano, toto přirovnání se může nejdříve zdát poněkud kuriozní, ale prosím o strpení.



Obr. 42 – Zjednodušená reprezentace platonského trojúhelníku

Odmyslíme-li si koncepty duše a esence, v případě obrázku, anamnézy a methexy, chápejme platonský trojúhelník ve světě herních enginů následovně.

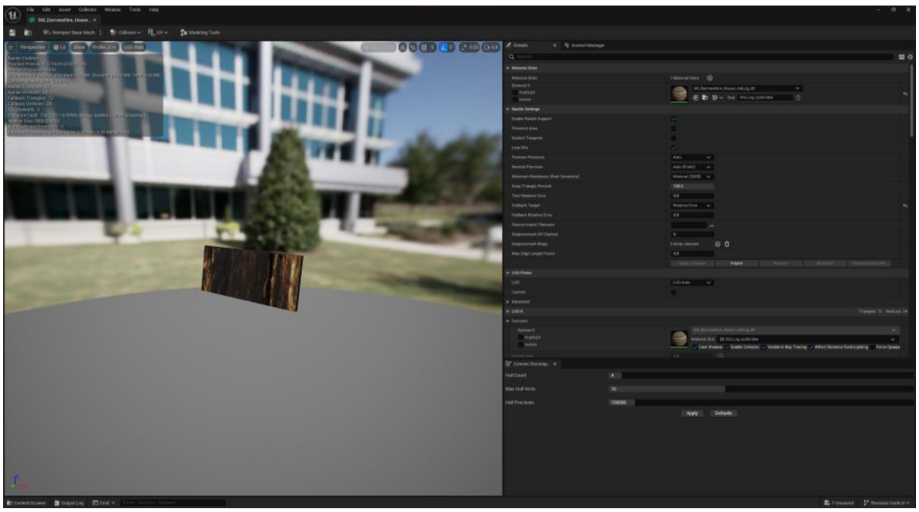


Obr. 43 – Adaptace Platonského trojúhelníku na herní engine

Koncept Ideí (jak redundantní formulace) zastupují tzv. Classes. Classes, převzáno ze světa programování, jsou základními konstrukty objektově orientovaného programování. Tyto konstrukce definují šablony pro vytváření objektů, či „subclasses“, které sdílejí společné vlastnosti a chování, jako jsme se již zmínili např. u Master Materials a Material Instances. Classes obsahují parametry a funkce, či logiku, které určují, jak se jejich objekty (subclasses) budou chovat a jak budou fungovat. Pokud bychom se bavili o statickém modelu importovaném do enginu, tento model by spadal do kategorie Static Mesh Class. Classes jsou tedy „prapůvodními idejemi“ celého herního enginu.

Invokace těchto classes, tedy, například pokud onen model vložím do scény, nazýváme tzv. Actors, česky „herci“. Actors jsou skutečnými reprezentacemi classes, tedy sdílejí jejich vlastnosti, ale jakožto invokací daného class tyto vlastnosti mohou měnit, podobně jako Material Instance. Dům ve scéně by tedy spadal do kategorie Static Mesh Actors. Tedy, pro představu, chápeme, že „Class“ je „idea modelu“, a „Actor“ je samotný model ve scéně, v levelu. Assety, které jsou přítomny v Content Browseru, které jsme buď naimportovali, nebo pořídili, jsou tedy classes. Pouze pro rekapitulaci, při jejich vložení do scény vytváříme actors, jejich reprezentace. Vlastnosti actors a samotné class, které reprezentují, se mohou rozcházet, pokud to tak chceme. Avšak při prvotní referenci class

pomocí actor, bude actor kopírovat původní vlastnosti class. Tyto vlastnosti můžeme měnit v editoru příslušné class, například v Static Mesh Editoru.



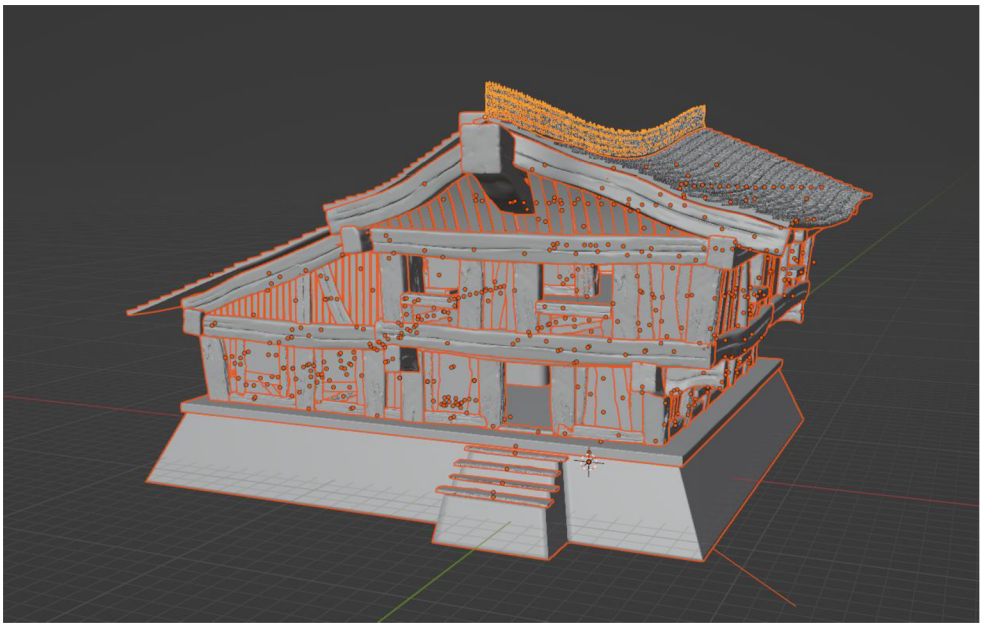
Obr. 44 – Static Mesh Editor v Unreal Engine

Pomocí kamery vnímáme scénu jejím zorným polem, tedy přes tzv. frustrum. Toto není pouze analogie, a to z důvodu již zmiňovaného frustrum culling, kdy se svět skutečně proměňuje a mění na základě toho, co kamera potřebuje vidět a co tedy musíme renderovat.

V případě videoher se o změny ve světě stará interakce světa s hráčem na základě naprogramovaných akcí nebo akcí naskriptovaných za pomocí blueprints. Ve světě animace vnímejme interakci jako klíče ovládané přes Sequencer.

Pochopitelně si zde nebudeme vysvětlovat celé uživatelské rozhraní enginu, ale tato část UI UE je důležitá pro pochopení asset managementu pro projekty vyrobené za pomocí enginu.

3.1 Asset management



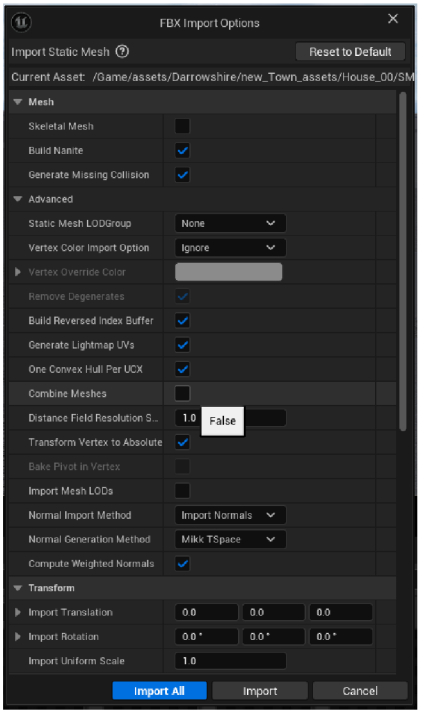
Obr. 45 – Náhled domu z projektu Darrowshire v Blender 3.4

Jak jsme již zmiňovali, při přípravě obsahu pro engine se musíme řídit některými pravidly. Ironicky, pokud se bavíme o UE5 a využití geometrie Nanite, pak je přístup k geometrii analogický přístupu offline VFX produkce. Tedy, modely můžeme stavět ze stovek modulů, které společně pomyslně tvoří onen objekt, v ilustrovaném případě, dřevěné plaňky a zdi domu.

Pokud bychom nevolili přístup virtualizované geometrie, museli bychom model vytvořit dle pipeline pro per-object online render, kdy jsou pravidla následující. Celý objekt musí být jedna vodotěsná kohezivní a spojená geometrická síť. Podle nároků draw calls se snažíme využít co možná nejmenší množství materiálů a tomu podvolujeme přípravu UVs, tedy přípravu rozloženého modelu do 2D prostoru, podobně jako origami, pro texturování, tedy malování povrchů danými materiály.

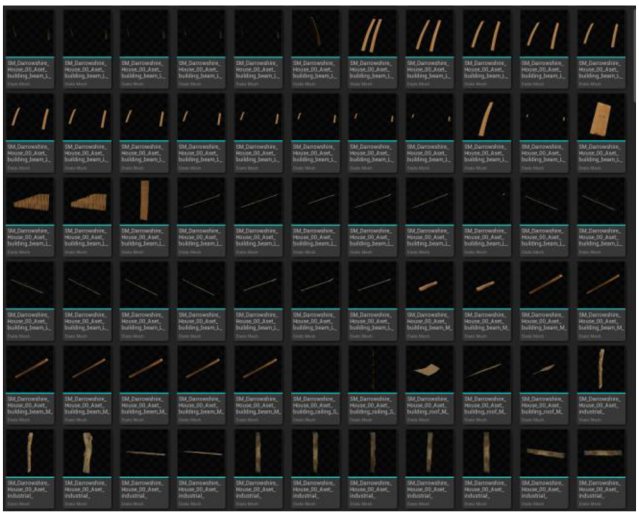
Avšak my zkoumáme především integraci Unreal Engine do existujícího pracovního postupu daného animačního studia, tedy věnujme se cestě nejmenšího odporu a využití Nanite.

Model, který je složený z mnoha dalších subobjektů exportujeme jako .FBX. Preferujeme možnosti exportu vybraných objektů dle možností zdrojového programu, tudíž při nabídce kontextuálního menu vybíráme možnost „Export Selected“ s vybranými objekty scény. Exportovaný model ukládáme do adresáře pracovního úložiště. Při importu engine nenecháme sjednocovat objekty, naopak možnost „Merge Meshes“ necháváme vypnutou.

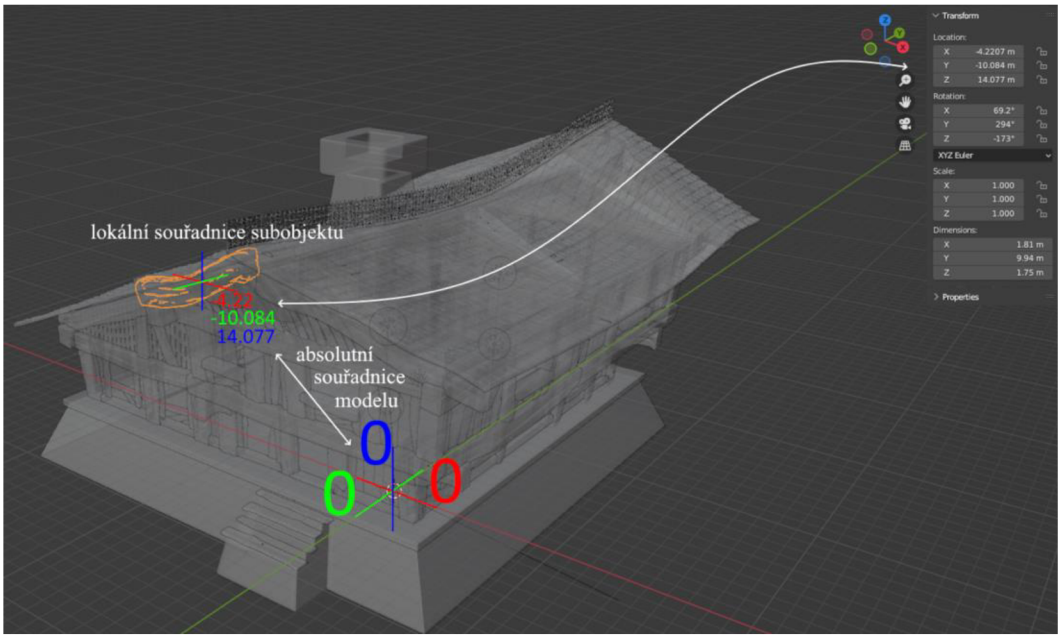


Obr. 46 – Kontextuální menu importu modelu do Unreal Engine

Po importu v Content Browser vidíme importovaný model rozdělen do individuálních stavebních kostek ze kterých byl vyroben v zdrojovém programu, tedy, v našem případě, v softwaru Blender.



Obr. 47 – Náhled Content Browser a importovaných modulů modelu



Obr. 48 – Souřadnice modelu v Bleder

Export modelu složené z více modulů, tedy subobjektů, či individuálních statických sítí chápeme jako export celé dané scény. Na toto téma jsme již narazili při diskusi o exportu scény s lesem, či jiným porostem, v předchozí kapitole. Ale na rozdíl od exportu scény za cílem následné editace, nyní z vlastností lokálních souřadnic budeme těžit.

Pokud jsme tímto způsobem vyexportovali celý model složený ze subobjektů, nulové souřadnice, tedy pivot point, celého objektu se stává nulovými souřadnicemi všech subobjektů. Tedy souřadnice všech objektů budou 0, 0, 0 ačkoliv jejich pozice bude různá. Rozdíly v jejich reálné pozici („offsets“) jsou totiž započítány do daných souřadnic.

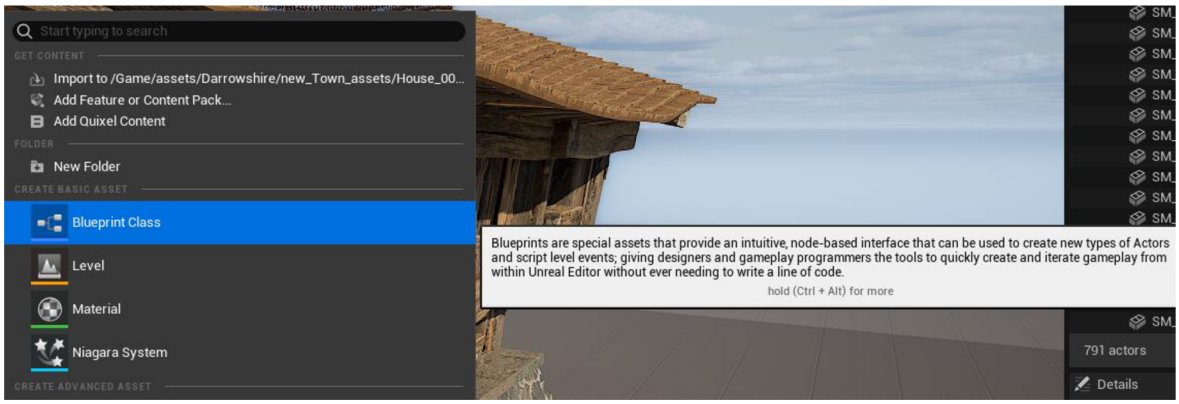
Díky tomu můžeme bez obav zvolit všechny subobjekty a vložit je do scény. Subobjekty budou ve svých náležitých, správných pozicích se zachovanými transformacemi.

Tento přístup k importu modelů volíme z důvodu rychlejšího artdirectingu, možnosti úprav individuálních subobjektů bez nutnosti předělávání jiných vlastností spojené sítě, a také díky vlastnostem Nanite geometrie, která preferuje tento modulární přístup z důvodů optimalizace.

Obr. 49 – Model domu v Unreal Engine

Čtenáře už nicméně jistě napadla jedna zásadní nevýhoda tohoto přístupu. Samotný velký počet modulů, se kterými pracujeme, namísto jednoho celku, se kterým by se pracovalo lépe.

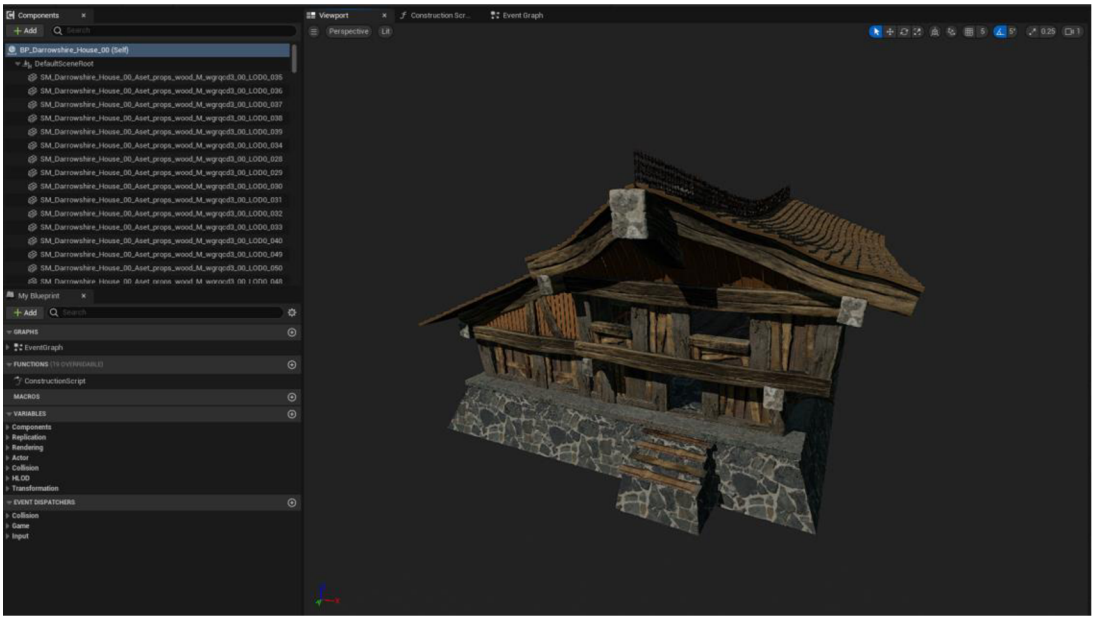
Využijeme již zmiňovaného systému referencí známého jako actors. S dovolením se vraťme mezi poslední řádky předešlé kapitoly, k diskusi o blueprints.



Obr. 50 – Kontextuální menu „Create New“ v UE s definicí Blueprint Class

Blueprint nám nyní poslouží jako „lokální scéna“ pro model. Vytvoříme tedy Blueprint Class, v našem případě Blueprint Class domku. Blueprint Class nám nepovoluje pouze pracovat s velkým množstvím modelů, či subobjektů, jako s jedním velkým modelem, ale také zajišťuje určitou míru interkativity vůči materiálům a jejich instancím, krom jiného. Pomocí blueprints můžeme skriptovat jakékoliv interakce, ať už jde o změny barev, textur, či o zapínání a vypínání fyzikální interakce. Díky tomu, můžeme například nahrazovat jednoduché animace fyzikální simulací ve jméně konzervace času.

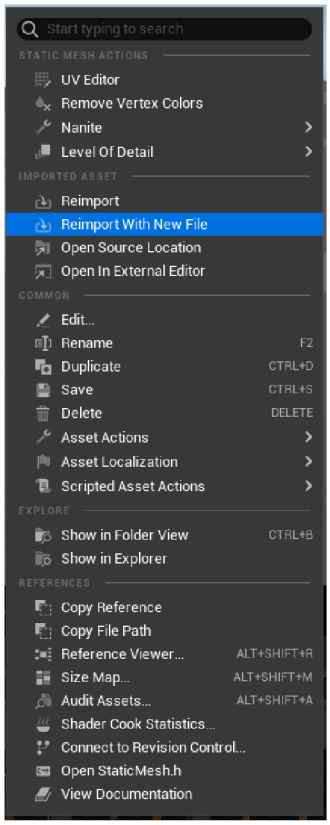
Do nové Blueprint Class vložíme všechny podobjekty domku, et voila.



Obr. 51 – Blueprint Class domu v projektu Darrowshire

Pokud bychom nyní potřebovali měnit původní source asset domu, není to žádný problém. S FBX souborem zacházíme jako s nosičem všech dat daného modelu, s tím, že můžeme editovat i pouze jeho části za využití stejného souboru.

Po editaci v daném externím programu stačí najít editovanou část modelu a využít již zmiňované funkce Reimport, či Reimport With New File, pokud využíváme různé soubory pro iterace vývoje, například House_v001, 002 a tak dále.

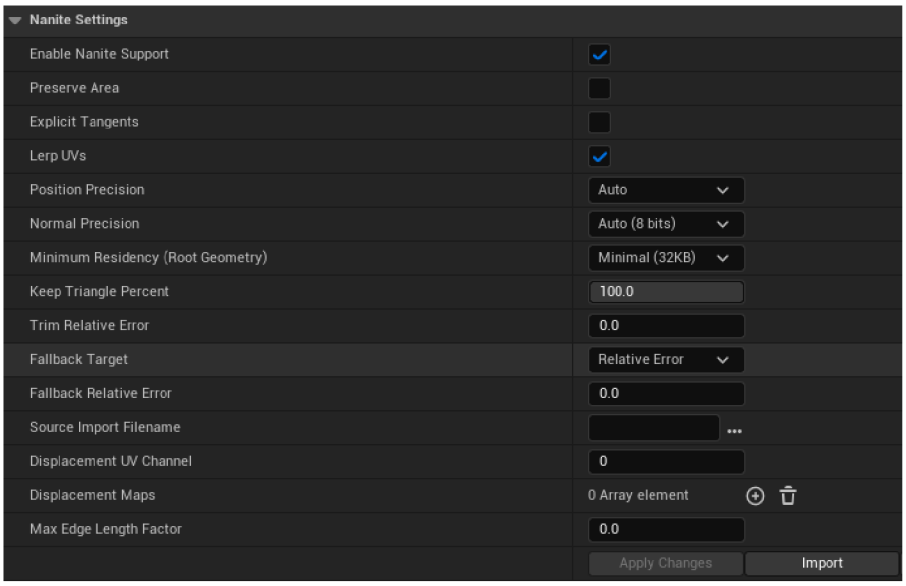


Obr. 52 – Kontextuální menu Static Mesh Class a možnost Reimport With New File

Při základním importu, tedy importu bez jakýchkoliv deviací od defaultních, základních nastavení nanite geometrie typicky objevíme špatný shading daných objektů. Toto je normální a způsobeno využitím tzv. Fallback Mesh pro zjištění světelných interakcí objektu. Vzpomeňme si na Mesh Distance Fields a Cards.

Technické odůvodnění tohoto vykreslení je fakt ten, že pro výpočet světelných interakcí a tedy i stínů je v rámci UE5 využit tzv. Fallback Mesh, tzn. zjednodušená geometrická síť objektu, se kterým pracujeme. Pokud Unreal donutíme tuto síť nevyužívat a místo ní využívat samotný původní model, získáme tak správný vzhled a shading.

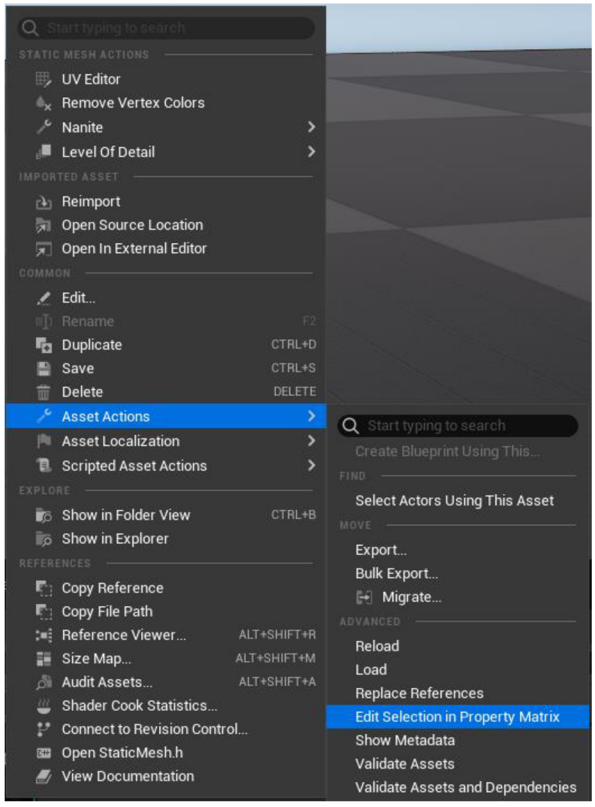
Je nutno podotknout cenu na výkonu, avšak ve světě VFX je čas strávený nad renderem v našem případě značně kratší, než by byl utracený čas strávený optimalizací fallback modelu pro správný shading.



Obr. 53 – Nanite Settings v Static Mesh Editor

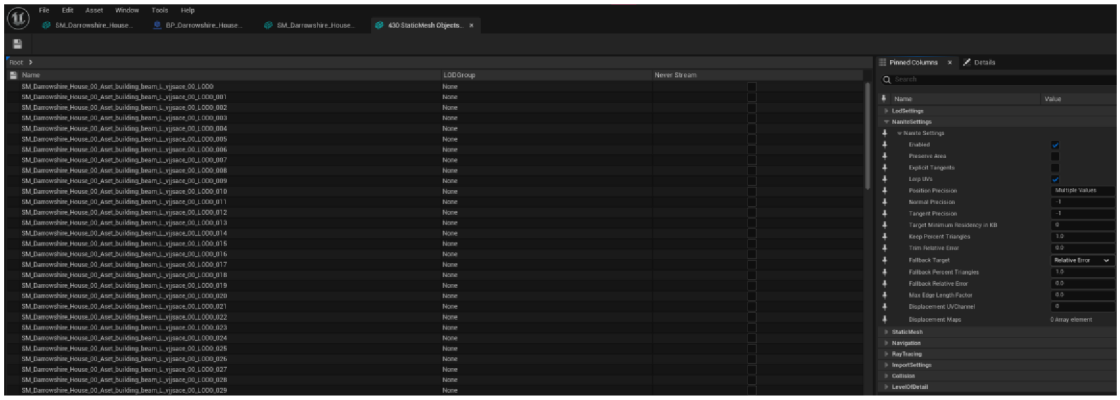
Tento problém opravíme pomocí Static Mesh Editoru. V sekci „Build“ změníme Fallback Target z původního nastavení na „Target Relative Error“ a hodnotu určíme 0, tedy minimální chyba.

Pro editaci velkého množství subobjektů, především pokud využíváme Nanite, je nezbytným nástrojem tzv. Property Matrix, tedy matice vlastností. Pochopitelně manuální úprava všech subobjektů, jak byla zmíněna, by byla extrémně časově náročná a frustrující. Proto pro tzv. „batch editing“, tedy editování velkého množství objektů, využijeme Property Matrix.



Obr. 54 – Zvolení Property Matrix v kontextuálním menu

Editace assetů je poté analogicky stejná k editaci jediného modelu, s lehkým rozdílem v uživatelském rozhraní.



Obr. 55 – Property Matrix

Při následovných reimportech již tato nastavení nemusíme měnit. Asset se naimportuje a respektuje předem určená nastavení.

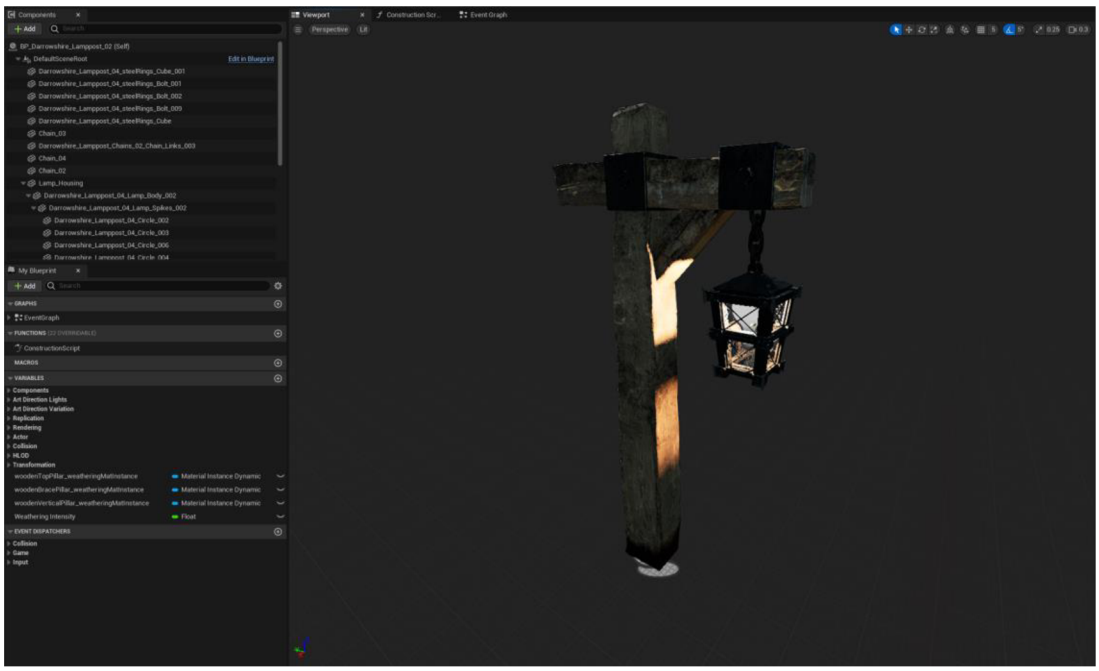
Reimportu využíváme také při práci s texturami. Dle počtu materiálů a využití tzv. multipurpose map, jinak známo jako stacked maps, může být počet textur poněkud vysoký, ačkoliv se vždy snažíme jejich počet minimalizovat.

Iterace textur tedy řešíme stejně jako iterace modelů. Dané verze textur reimportujeme a vybereme iterace nové, které ty původní (pochopitelně pouze v projektu Unreal Engine, nikoli v pracovních adresářích) přepíšou. Tím pádem nemusíme dále měnit materiálové instance.

3.2 Art-direction za pomocí Blueprints

V předchozí kapitole jsme také zmiňovali art direction a výhody blueprints při art-directingu scény. Pojďme si toto téma přiblížit.

Při výrobě nástrojů pro art direction s blueprints využijeme již zmiňované Material Instances a blueprint scripting. Při otevření jakékoli blueprint class se setkáváme s třemi hlavními kartami uživatelského rozhraní. Jde o Viewport, Construction Script, a Event Graph. Viewport zobrazuje trojrozměrnou reprezentaci virtuální, či, chcete-li lokální „scény“ blueprint. Construction Script je logika, kterou daný blueprint provádí pokaždé, kdy je „zkonstruován“– tedy v momentě, kdy je Blueprint Class přidán do levelu a tím vytváříme Blueprint Actor. Logika zde přítomná je také aktualizovaná při jakékoliv změně vlastností daného blueprintu.

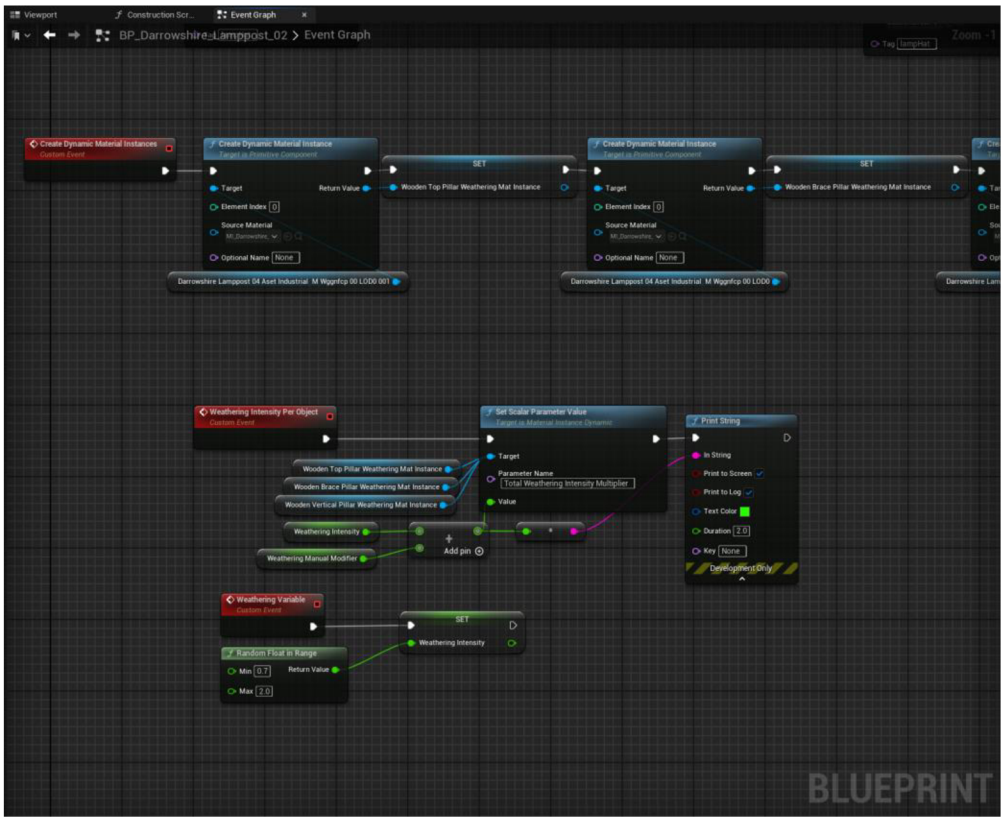


Obr. 56 – Blueprint Class pouliční lampy z projektu Darrowshire

Event Graph je prostor pro vytváření logiky závislé, jak napovídá název, na tzv. Events, tedy „událostech“ či děních, které buď sami definujeme, a nebo kterých využíváme za pomocí tzv. Hooks a Event Dispatchers.

Hooks, „háky“, jsou funkce nebo metody, které přiřazujeme k různým logickým událostem v logice blueprints či samotné aplikace, hry, nebo za pomocí klíčování v Sequencer. Hook můžeme „zaháknout za akci“ a tím pádem společně spouštět, či reagovat na dění ve scéně.

Event Dispatchers jsou logické mechanismy, které informují, vysílají a zachytávají události mezi různými blueprints, či částmi samotné aplikace. Pomocí nich informujeme proměnné, či je měníme, nebo voláme různé funkce a jiné události, čímž je zajištěna logická kauzalita a interaktivita.



Obr. 57 – Event Graph Blueprint Class lampy z projektu Darrowshire

Paradigma je následující. V sekci Construction Script „voláme“ události, tedy spouštíme logiku, která je přítomna a definována specifickými Events, událostmi, které jsou definovány v Event Graph. Tyto Events mohou navazovat na jiné Blueprints, či přímo definují dění samotného Blueprintu.

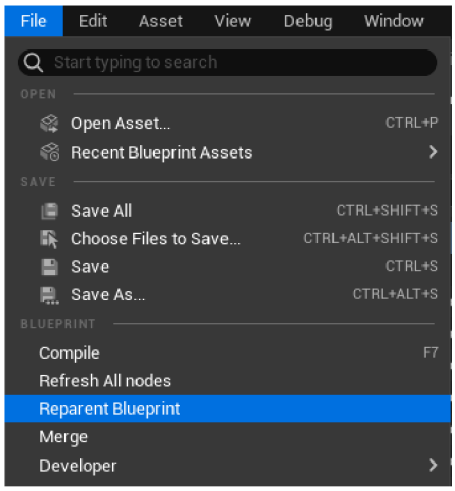
Podobně, jako u Materialů a Material Instances, můžeme také vytvářet Blueprint Classes, které budou „ideí“ jiných Blueprint Classes, tedy vytváříme „Classes classes“ za cílem sdílení vlastností a informací mezi těmito Classes. Vysvětlíme si to na specifickém příkladu z produkce projektu Darrowshire, praktické části diplomové práce.

Technická vize projektu Darrowshire zněla: Co nejméně animace, co nejvíce muziky. Vytvoření projektu, který by mohl sloužit jako šablona pro profesionální aplikace a z pohledu Technical Artist vytvoření systému, který celou scénu prováže a scéna tak dynamicky reaguje na jakékoliv změny, například změny počasí.

Pro samotnou simulaci počasí a atmosféry jsem využil systém z Marketplace jménem Ultra Dynamic Weather. Tento systém využívám již řadu let, a to i v profesionální produkci. Produkt obsahuje spoustu parametrů simulace počasí a plný blueprint systém, který člověk může využít pro art directing svých scén.

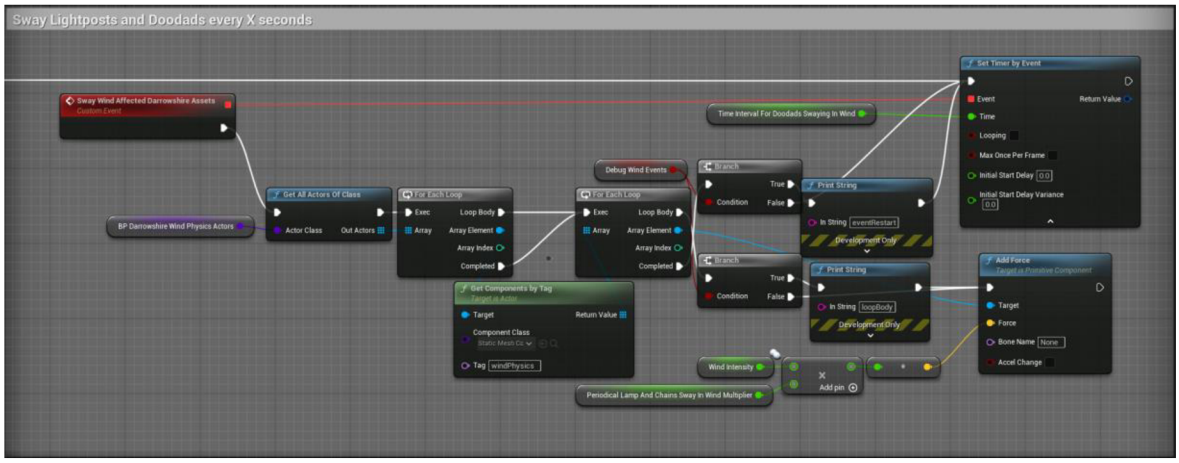
Reakce na změny počasí by se měly projevovat například na praporech nebo pouličních lampách, které visí na řetězech. Věděl jsem tedy, že budu potřebovat vyrobit Blueprint Class, který bude „Parent Class“ všech objektů, které bude počasí ovlivňovat. Tento Blueprint Class jsem nazval „BP_Darrowshire_windPhysics_Actors“. Nemá žádné zvláštní vlastnosti! Slouží pouze jako identifikátor blueprints, které jej budou využívat jako svůj Parent Class pro potřeby dalších referencí v logice, kterou nyní prozkoumáme.

Abychom mohli ovládat a upravovat data v blueprints například pouličních lamp, potřebujeme tyto lampy nějak označit. Na to nám slouží onen identifikační blueprint class BP_Darrowshire_windPhysics_Actors“. Nicméně, v době, kdy jsem tento blueprint vytvořil, jsem již lampy a různé jiné blueprinty vyrobil. Potřeboval jsem tedy určit, že tyto blueprinty budou sdílet identifikátor blueprintu BP_Darrowshire_windPhysics_Actors. Jinými slovy, z blueprints, které jsem již vytvořil, jsem potřeboval udělat child blueprints tohoto blueprintu. K tomu jsem využil funkce „Reparent Blueprint“.



Obr. 58 – Reparent Blueprint

Následně jsem vybral BP_Darrowshire_windPhysics_Actors blueprint. Tento krok jsem opakoval pro všechny blueprints, které měly být ovlivněny dynamickým počasím.

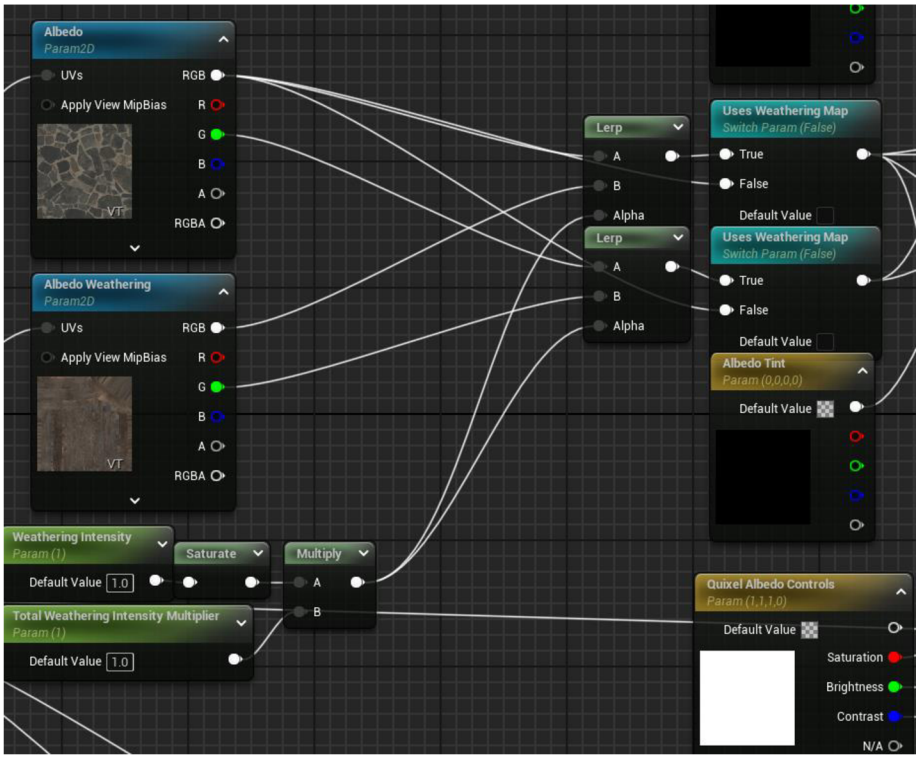


Obr. 59 – Logika vybrání specifických classes, ve kterých budeme aplikovat sílu větru (počasí)

V ten moment jsme schopni logice v systému Ultra Dynamic Weather říci, že potřebujeme vybrat všechny actory scény, které jsou „subclass“ blueprintu BP_Darrowshire_windPhysics_Actors. Tím získáme všechny objekty, které mají být ovlivněny počasím. Z této skupiny vybereme za pomocí označení, tzv. Tag, ty specifické subobjekty, které mají mít fyzikální interakce. V našem případě to byly například řetězy, které se měly pohybovat v závislosti na síle větru. Tag je označení za pomocí textu, tedy „String“ (z programování). V blueprintech objektů jsem tedy vybral specificky subobjekty, o které šlo, a těm jsem přiřadil Tag „windPhysics“. Následně, přímo v systému počasí, aplikujeme sílu pomocí funkce addForce a tuto akci opakujeme v závislosti na předem určeném intervalu, který definujeme jako proměnnou, jejíž hodnotu můžeme dynamicky měnit například v Sequencer.

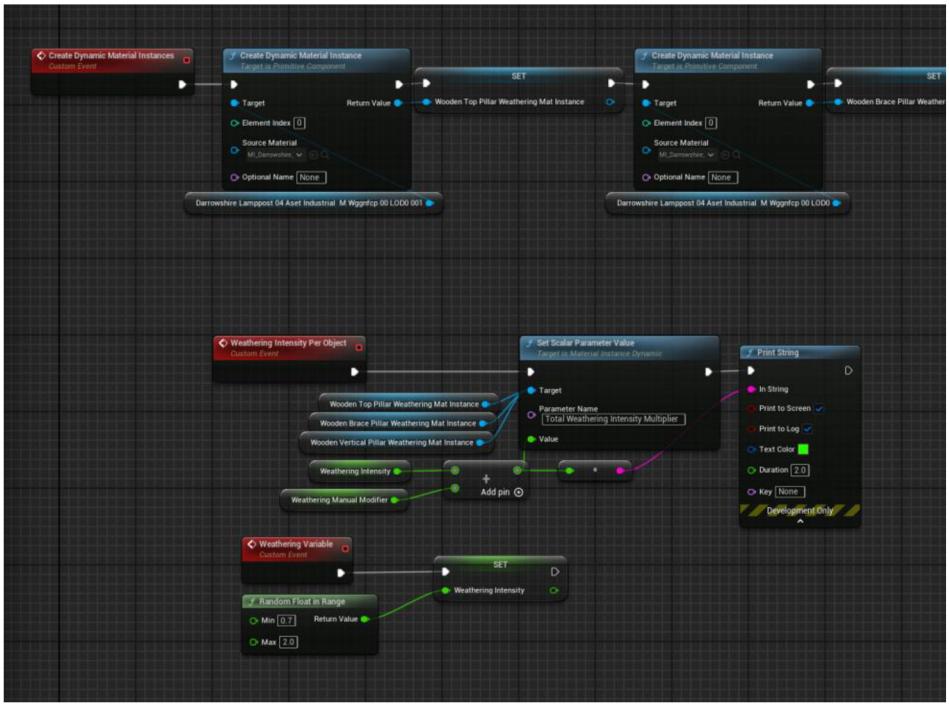
Analogicky stejný přístup jsem využil abych vyřešil tzv. „grounding“ a „weathering“ modelů. Grounding je proces, kdy umělec zasazuje daný objekt do scény za využití zašpinění, či různých technik prolnutí objektu s pozadím. Weathering, jak napovídá název, představuje opotřebování vlivem počasí, tedy jde o různé nečistoty, špínu apod. V mém případě šlo o zasazení starých dřevěných kůlů, tyčí a domků do terénu filmu. Buď jsem mohl vytvořit mnoho textur plně manuálně, kdy každá textura by také potřebovala své variace, aby každý kůl nevypadal stejně, nebo jsem mohl zvolit přístup s blueprintovým využitím materiálových instancí a promítnutím jedné jediné plně špinavé textury za využití náhodných hodnot, aby každý kůl a každý objekt vypadal unikátně.

Pochopitelně jsem zvolil přístup Tech Artisty, tedy ten druhý, vzhledem k efektivitě využití času a nevoli procvičovat texturování na pěti různých druzích dřevěných kůlů. V programu Substance Painter jsem vytvořil velice špinavou verzi jednoho kůlu. Jelikož kůly byly vyrobeny pomocí stejného modelu, byl tento proces poněkud rychlý a jednoduchý. Do Ubermateriálu, který využívá celé městečko, jsem přidal proměnné, ovládající zda daný objekt vyžaduje weathering, a pokud ano, pak na základě proměnné určujeme intenzitu plně zašpiněné verzi textury.



Obr. 60 – Weathering v Ubermateriálu Darrowshire

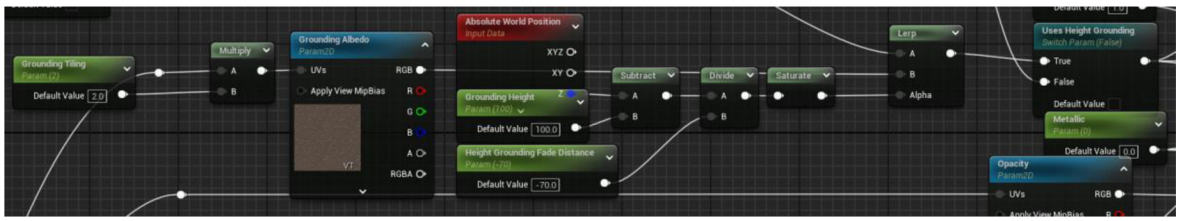
V Event Graph blueprintů objektů, které vyžadovaly weathering jsem následně tyto proměnné naskriptoval za využití Material Instances. Pomocí funkce náhodné generace hodnot jsem tuto hodnotu mohl na základě každého objektu získat během runtime. Touto hodnotou, která je sčítána s hodnotou, kterou mohu sám měnit, tzn. nahodilost má také svůj art direction, násobím intenzitu špinavosti. Tím je zajištěno, že každý kůl je trošku jinak špinavý, a vypadá unikátně ačkoliv využívá tu samou texturu.



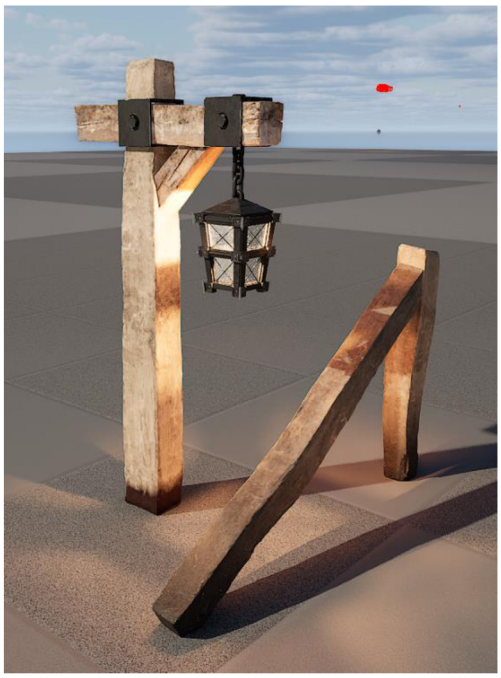
Obr. 61 – Dynamické Materiály a ovládání Weathering v Blueprint Class

Problém představoval grounding plotů. Grounding, na rozdíl od weatheringu, vyžaduje větší míru art direction. Především, protože potřebujeme, aby se „prolnutí“ kůlu a terénu dělo přesně ve výšce, ve které onen kůl protíná hlínu.

Zvolil jsem tedy jednodušší přístup. Místo vytváření specifické textury a jejího následného násobení a míšení s texturou kůlu jsem využil informací o transformaci daného objektu, jeho výšky, za využitím funkce Absolute World Position, která vrací absolutní transformaci XYZ objektu, ke kterému je přiřazena.



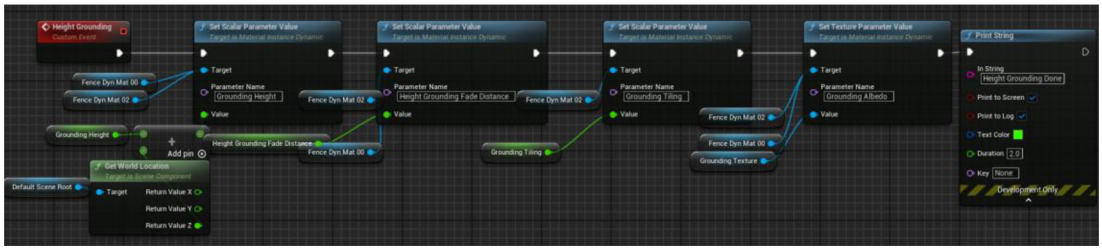
Obr. 62 – Height Grounding v Ubermateriálu Darrowshire



Obr, X - Lampa a část plotu využívají weathering a grounding

Pokud od výškové transformace, tedy absolutní hodnoty Z, následně odečteme námi určenou požadovanou výšku „groundingu“, tedy, prominete-li, „zahlinění“, výslednou hodnotu můžeme využít jako alpha hodnotu lineární interpolace mezi texturou čistou a špinavou. Jinými slovy, jde o překryt špinavé hlíny přes čistý plot dle hodnoty výšky objektu vůči vyžadovanému zašpinění.

Dále, místo unikátní zašpiněné textury plotu jsem využil jednoduše texturu hlíny, která byla dostatečně podobná svými vlastnostmi hlíně, kterou využívá terén filmu. Pro plnou schopnost art direction jsou tato textura společně se zmiňovanými hodnotami plně parametrizovány a přístupné z Blueprint Class a nastavení Blueprint Actor ve scéně. Tedy můžeme měnit míru zašpiněnosti a samotnou texturu špíny přímo ve scéně, a od objektu k objektu.



Obr. 63 – Logika Grounding pro plot

Díky zakomponování groundingu přímo do Ubermateriálu celého městečka měly nyní všechny objekty možnost grounding využívat, za předpokladu toho, že by pro ně Tech Artist, tedy já, vyrobyl svou logiku.

Tato specifika ohledně blueprints vysvětluji především z důvodu uvědomění si samotné propojenosti a síly Unreal Engine, pokud s ním zacházíme tak, jak je to zamýšleno, tedo jako s komplexní sadou nástrojů, které zrychlují jak vývoj tak rendering scén. Nicméně ono zrychlení má svou cenu v formě preprodukce, testování a dokazování teoretických konceptů, které dobrý Tech Artist předloží.

3.3 Preprodukce a plánování

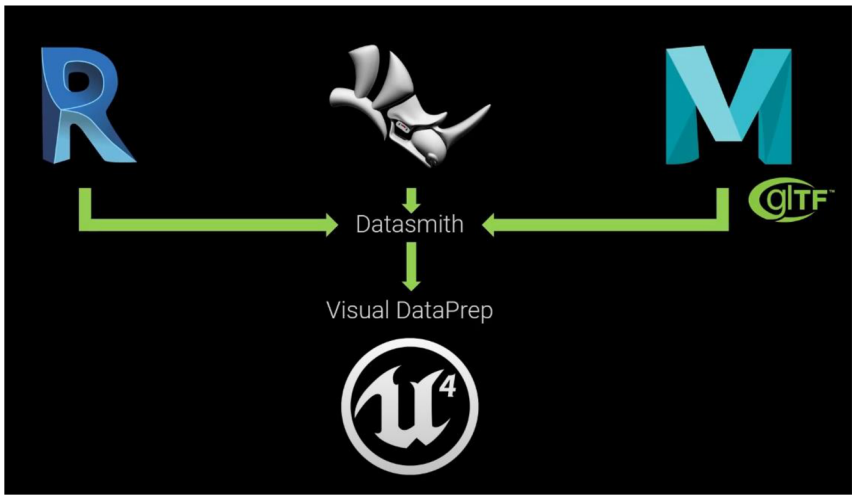
Při využití herního enginu pro delší, či komplexnější projekt je preprodukce nejdůležitějším stádiem vývoje. Během pre-produkce tým zjišťuje své cíle a a při znalosti cílů testuje využití daných workflow a daného programu pro dosažení oněch cílů. Jde tedy o určitou „Research and Development" fázi vývoje.

Speciální pozornost věnujme již v této fázi plánovanému způsobu finálního renderu. Ačkoliv real-time deferred render a offline propočítaný deferred render mohou vypadat velice podobně, s výhodou offline deferred renderu v klaritě a detailech scén, například využití Path Tracingu oproti Ray Tracingu způsobí značný rozdíl ve vzhledu scén.

Pamatujme, že investice do této fáze produkce je investicí, která se X-násobně vrátí. Jak praví staré české moudro, „Těžko na cvičišti, lehko na bojišti".

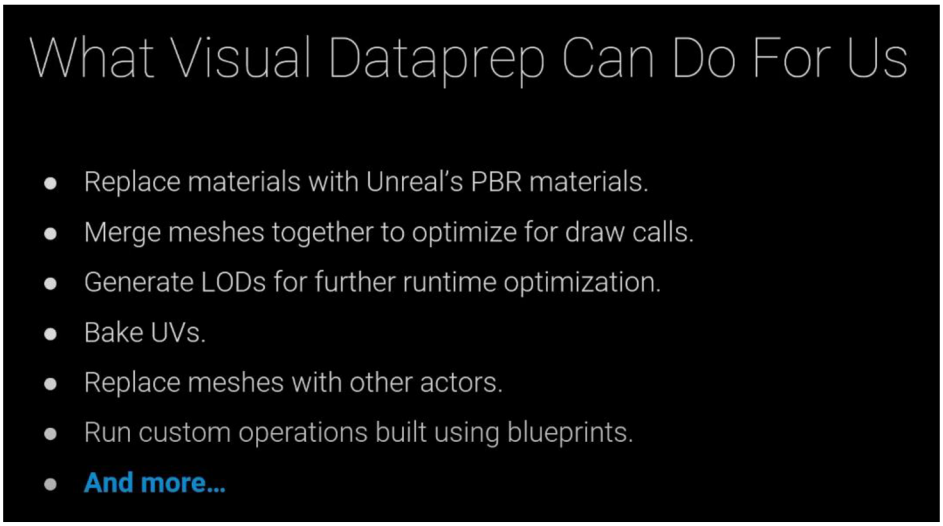
Pokud víme, že budeme využívat převážně vlastní modely a scény, či assety vyrobeny v externím softwaru, jako je např. Maya, krom manuálního managementu, který jsme si již popsali, můžeme ve fázi preprodukce připravit systém integrace assetů za pomocí tzv. Visual Dataprep, dalšího nástroje Unreal Engine.

3.3.1 *Visual Dataprep*



Obr. 64 – Flowchart Visual Dataprep z Using Visual Dataprep in Unreal Engine | Webinar

„Tento nástroj je určen pro přípravu dat pro neprogramátory. Uživatelské rozhraní umožňuje vytvořit řetězec akcí skládáním bloků „selection filtrů“ a operátorů. Používání je velmi jednoduché.“[27]



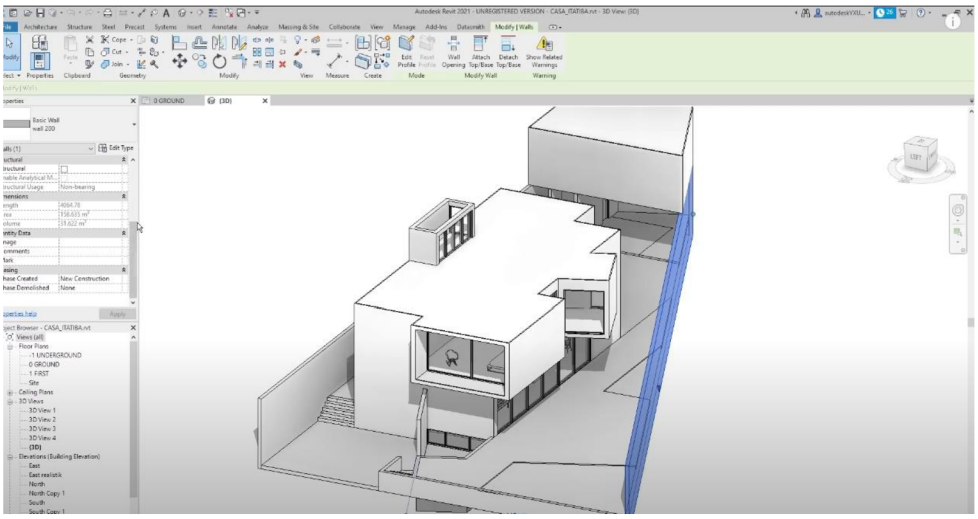
Obr. 65 – Slide z Using Visual Dataprep in Unreal Engine | Webinar

Visual Dataprep je nástroj Unreal Engine, který využíváme v situaci, kdy chceme automaticky měnit či modifikovat obsah, který importujeme do enginu. Umožňuje

[27] Unreal Engine. *Using Visual Dataprep in Unreal Engine | Webinar*. 2021 [YouTube video] [Citace: 20. Červen 2024]

například automatické zaměňování původních materiálů za materiály vyrobené v Unreal Engine a jiné akce, které šetří čas při převádění assetů a scén z jiných softwarů.

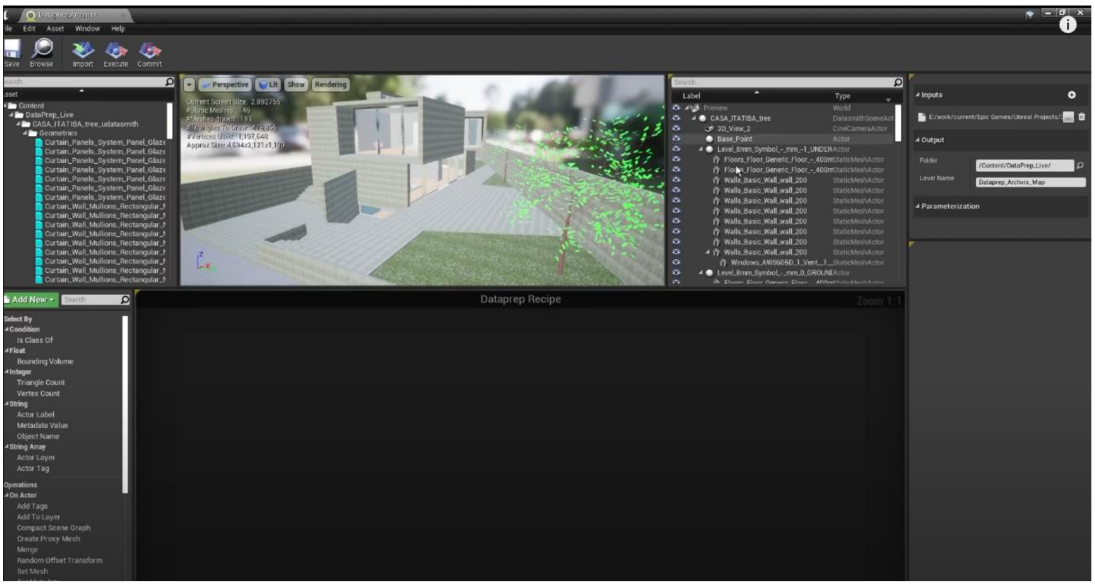
Dejme tomu, že scénu z Autodesk Revit potřebuji přenést do Unreal Engine. Vím, že se s touto situací budu setkávat častěji a analogicky podobné scény budu importovat i v budoucnu. Visual Dataprep je schopen vytvořit jakýsi „recept“ dle kterého se bude obsah importovat do enginu s úpravami a změnami, které předem určím.



Obr. 66 – Náhled Autodesk Revit z Using Visual Dataprep in Unreal Engine | Webinar

Pomocí Dataprep vytváříme jakýsi „meziprostor“ při importu obsahu. Obsah, který importujeme do enginu je držen v tomto prostoru, během procesu, kdy pomocí Visual Dataprep určujeme programatická pravidla daného importu. Pouze v momentě, kdy jsme s tímto programem, či, chcete-li, receptem, spokojeni, se obsah do enginu začne importovat.

Visual Dataprep se řídí informacemi zaznamenanými v tzv. metadata assetů. Metadata jsou popisná data, které obsahuje jakýkoliv digitální soubor, v našem případě 3D architektonickou vizualizaci domu a zahrady.



Obr. 67 – Náhled Visual Dataprep z Using Visual Dataprep in Unreal Engine | Webinar

V uživatelském rozhraní, vlevo dole, vidíme seznam selection filtrů a operátorů. Jde o seznam nástrojů, kterými vybíráme subobjekty a vlastnosti importovaného obsahu. Například můžeme vybírat obsah dle jeho objemu, počtu polygonů, či dle jeho strings, tedy textového označení, či názvu. Máme tedy plnou kontrolu nad vybíráním specifických částí importovaného obsahu a jeho následnou modifikací a proměnou. V demonstraci Epic Games pracují s příkladem změny materiálů scény.

"Velmi běžným pracovním postupem (v rámci Visual Dataprep) je substituce materiálů. Například nahradím všechny materiály stěn nějakým materiálem z Unreal Engine. K tomu je nejprve potřeba vybrat objekty. Mohl bych je zde vybírat jeden od jednoho podle jména a tento proces pořád opakovat, ale právě zde přijdou na řadu selection filtry."[28]

Vybereme filtr "Dle Metadata". Editor Visual Dataprep nám dovoluje přidávat logiku, podobně jako je tomu u blueprints. Každý filtr a operátor funguje jako logický blok, který vyhledává a filtruje data, se kterými dále pracuje. Zatímco filtr, překvapivě, filtruje data, operátor je dále proměňuje. V případě prezentace Epic Games je využit operátor Substitute Material, tedy zaměnění materiálu.

[28] Unreal Engine. *Using Visual Dataprep in Unreal Engine | Webinar*. 2021 [YouTube video] [Citace: 20. Červen 2024]

Obr. 68 – Selection Filtr a Operátor z Using Visual Dataprep in Unreal Engine | Webinar

"Nyní můžete vidět, že jakékoli objekty ve scéně, které měly metatag nebo metadata stěny, měly své materiály nahrazeny."[29]

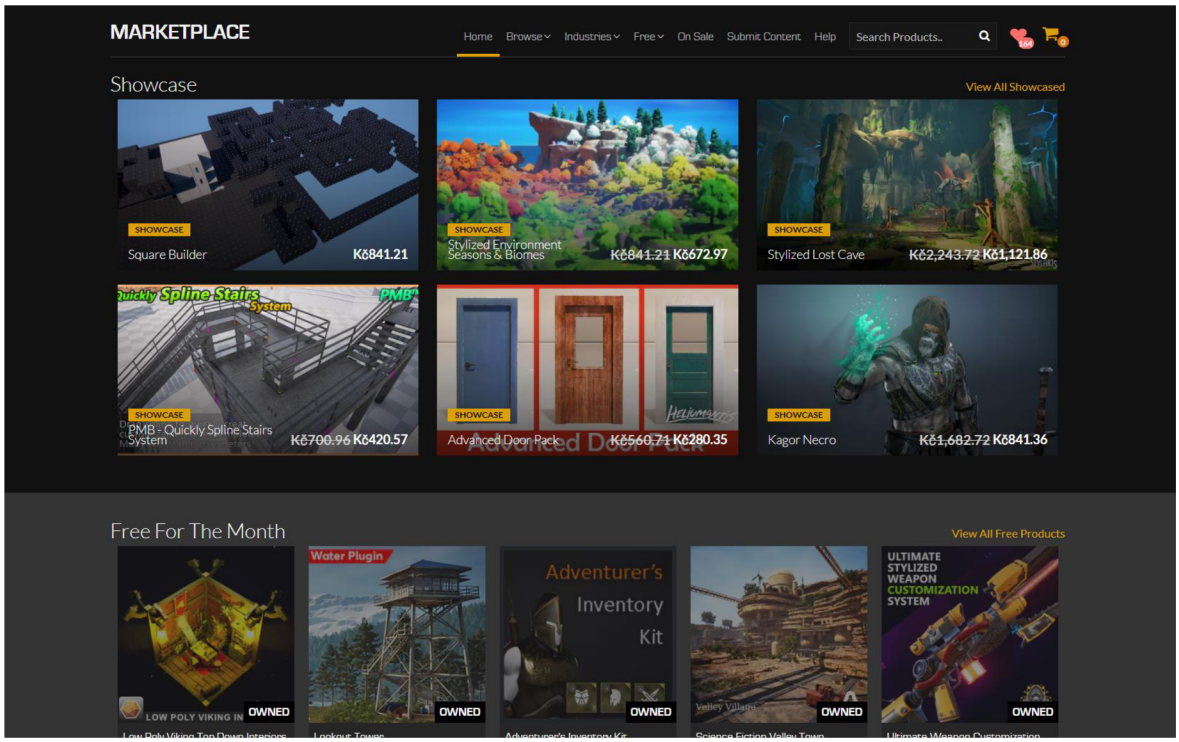
Pomocí Visual Dataprep tedy můžeme předpřipravovat procesy, které se budou odehrávat během importu contentu, namísto manuálního upravování např. materiálů po importu daného obsahu do enginu. Tento nástroj má potenciál ušetření desítek hodin strávených nad nevděčnými aktivitami otupující mysl. Jeho využití stojí za považenou v jakémkoliv projektu, který zahrnuje migraci obsahu z externího softwaru do Unreal Engine.

Pro vývoj můžeme vyvinout vlastní systémy a blueprints, podobné těm, které jsme již popisovali. Levnější a efektivnějším přístupem ale bývá přístup, kdy nejprve prohledáme Marketplace Unreal Engine a vyhledáme, zda problém, který hodláme řešit, již někdo nevyřešil za nás, a za poplatek ono řešení na Marketplace nenabízí. Právě možnost nákupu již existujícího obsahu je zásadní výhodou Unreal Engine pro studia, která v Unreal Engine vidí ušetření času a efektivitu.

[29] Unreal Engine. *Using Visual Dataprep in Unreal Engine | Webinar*. 2021 [YouTube video] [Citace: 20. Červen 2024]

3.3.2 *Ekosystém Unreal Engine a Marketplace*

Marketplace je domovem pravděpodobně milionů assetů, asset packů a systémů vytvořených specificky pro Unreal Engine. Sám se ani neodvažuji odhadovat fiskální hodnotu mého osobního Unreal Engine účtu, ale přiznám, že investice do mé profese utrácením za assety na Marketplace patří mezi mé největší výdaje. Jak říkají Američané (a můj táta), peníze dělají peníze.

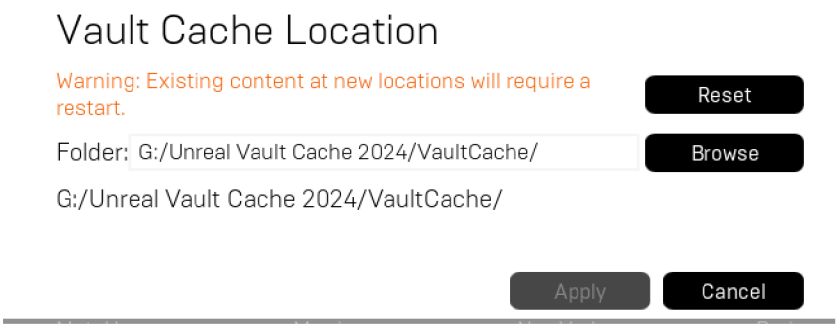


Obr. 69 – Screenshot z Unreal Engine Marketplace v Epic Games Launcher

Marketplace a Unreal Engine propojuje ekosystém závislý na programu Epic Games Launcher. Epic Games Launcher je programem analogickým například k programu Steam od firmy Valve. Kromě domova Unreal Engine jde o virtuální obchod pro videohry spadající pod vydavatelský department Epic Games.

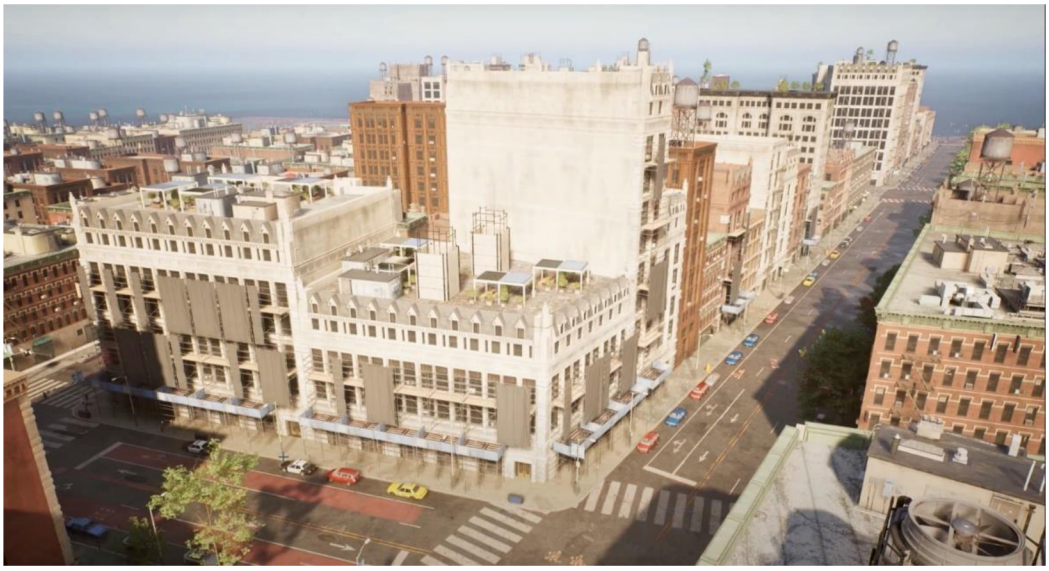
Část launcheru určenou pro Unreal Engine je zodpovědna za instalaci a kontrolu specifických verzí Unreal Engine na Vašem lokálním stroji a také je domovem pro Vault. Seznam assetů a služeb zakoupených na Unreal Engine marketplace. Z tohoto seznamu se také pomocí launcheru tyto assety, plugins apod. instalují přímo do enginu. Toto je důležité z důvodu managementu zakoupeného obsahu. Obsah je vázán na daný účet a z tohoto účtu by se měl také instalovat. Ačkoliv jde nastavit lokace Vault například na sdílený server na

pracovišti, či ve studiu, pohyb samotného obsahu, tj. vytažení z úložiště, se kterým Vault pracuje, je problematické. Pokud se chceme na pracovišti vyhnout opětovnému stahování identického obsahu, nastavme si Vault location v Epic Games Launcher do sdíleného repozitáře.



Obr. 70 – Nastavení lokace repozitáře staženého obsahu z Unreal Engine Marketplace

Marketplace nabízí assety, které mohou sloužit jako páteř daného projektu. Příkladem by byl projekt CityBLD, nástroj pro procedurální vytváření měst a urbanistického prostoru. Po zkušenosti s prací na projektu 7Energy, kdy jsem měl za úkol vytvořit typicky české město v Unreal Engine pro potřeby TV reklamy, jsem zkoumal procedurální generaci měst a systémy toho schopné. Bavíme se o době před existencí PCG. CityBLD v té době ještě nebyl dostupný, ale již existovala možnost projekt podpořit a stát se backerem koupí perpetuální license, což jsem také posléze udělal. Asset jsem již otestoval a všechny budoucí městské projekty plánuji vytvářet za pomocí CityBLD. Podobných příkladů jsou doslova tisíce.



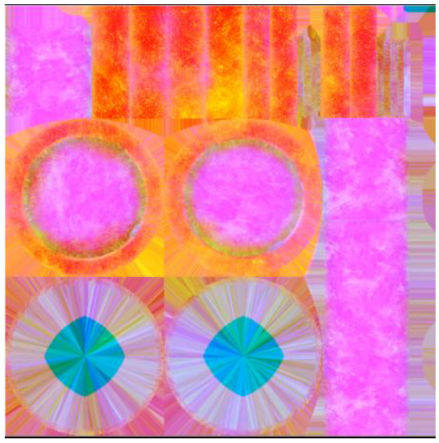
Obr. 71 – Obrázek města vyrobeného za pomocí CityBLD

3.4 Produkce a look development za využití Unreal Engine

Obecná pravidla výroby modelů pro Unreal Engine jsme již zmiňovali. Pro rekapitulaci, pro Unreal Engine 5 s využitím Nanite se nekladou žádné speciální limitace oproti klasické offline animační produkci. Nicméně, i tak se snažíme udržovat nízký počet draw calls a nízký počet objektů. Pokud bychom pracovali s Unreal Engine 4, musíme počítat s náročnější optimalizací pro draw calls, jak je do hloubky vysvětleno v *Unreal Engine a Průlomy v real-time renderingu*, a také v úvodních kapitolách této práce.

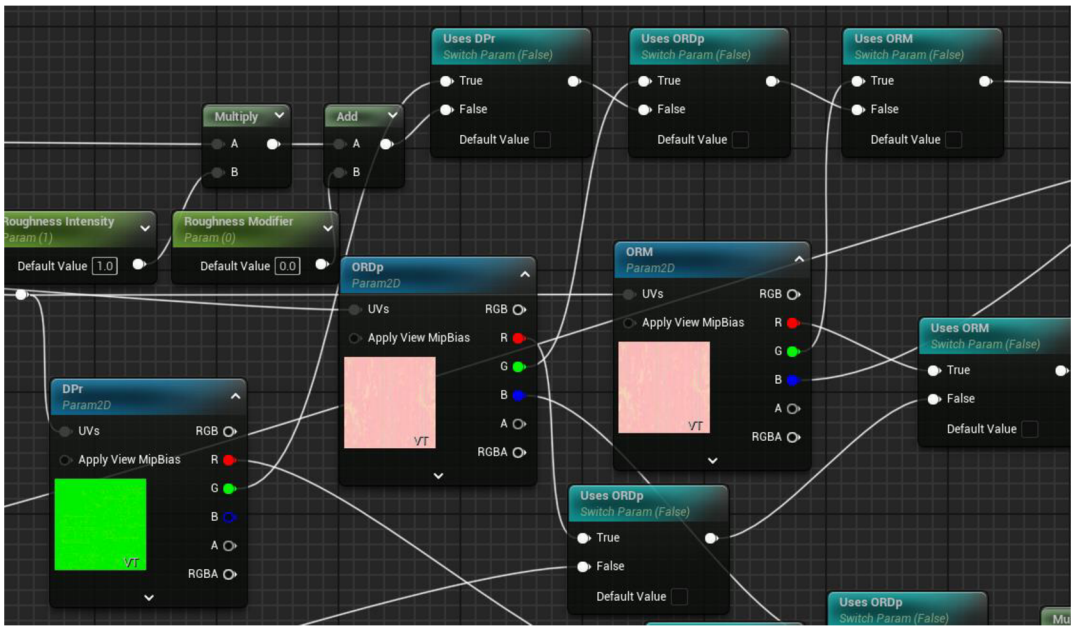
Produkci každého modelu provází i produkce jeho textur. Textury v Unreal Engine dělíme mezi klasické textury a dále známe tzv. Virtual Textures, virutální textury. Virtuální textury jsou přístup k renderování textur analogicky podobný k virtualizované geometrii. Fungují tak, že rozdělují existující textury na menší části, které pak využívají pro samotný rendering. Pokud tedy vidím pouze ¼ textury, není důvod, proč vykreslovat i zbylé ¾ v plném detailu. Jedinou změnou v rámci produkce při práci s Virtuálními Texturami je využití specifických texturových instrukcí při práci s materiály.

Při výrobě samotných textur preferujeme formát TARGA s příponou .TGA. Targu užíváme díky vlastnostem tohoto datového nosiče. Podobně, jako PNG, TGA s sebou nese čtyři datové kanály. R, G, B a A, tedy i tzv. alpha kanál. S každým datovým kanálem můžeme pracovat a uchovávat v něm různá, na sobě nezávislá data. V takovém případě se bavíme o již v úvodních kapitolách zmíněných multipurpose mapách. Tyto textury využíváme například při využití ORM nebo DpR, či ORDp workflow.



Obr. 72 – ORM textura z projektu Darrowshire

Tyto zkratky reprezentují různé druhy multipurpose map pro PBR workflow. PBR, tedy Physically Based Rendering, jak již bylo zmíněno, využívá především Roughness a Metallic dat ve formě textur. Z technických důvodů maximální počtu instrukcí pixel shaderu, a potažmo z důvodů optimalizace, se snažíme využívat co nejméně materiálových instrukcí, tedy se snažíme využít co nejmenší množství textur. Proto využíváme tyto multipurpose mapy. ORM, tedy, Occlusion, Roughness, Metallic uchovává data dle svého názvu. Písmena zkratky odpovídají pořadí RGB, tedy R, červený kanál nese data o Ambient Occlusion, G, zelený kanál, nese data o Roughness, a B, modrý kanál, nese data o metaličnosti objektu. Podobně, DpR vypovídá o Displacement (Height) a Roughness, a ORDp v sobě skrývá Occlusion, Roughness a Displacement texture data.



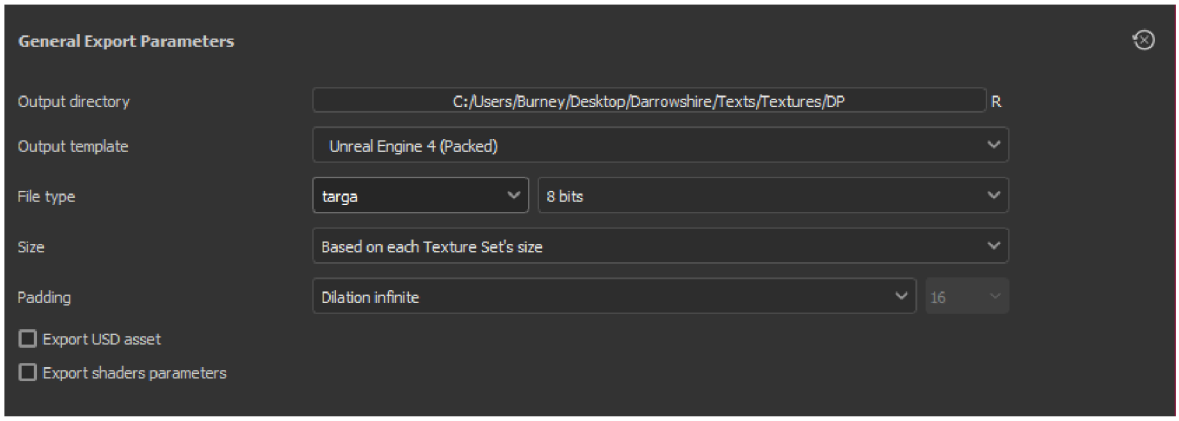
Obr. 73 – Využití multipurpose map v Ubermateriálu

PBR přidává materiálům nové vlastnosti – tzv. metallic a roughness. Místo využívání spekulárních dat využívá data nová, která určují metaličnost a hrubost daného povrchu. Tento přístup, jak již jméno napovídá, je fyzikálně korektním způsobem vykreslování materiálů a tak dovoluje umělcům se mnohem lépe přiblížit podobě objektů skutečných.[30] (McDermott, 2018)

[30] McDermott Wes. *The PBR Guide:A Handbook for Physically Based Rendering,* 2013. Parafráze.

V dnešní době je již při výrobě textur standardní využití programu Substance Painter. Nicméně i výroba textur se musí podvolit určitým pravidlům a požadavkům enginu.

Při exportu textur z programu Substance Painter využijeme oficiální Unreal Engine preset s jednou vyjímkou. Výstup textur budeme limitovat do 8bit colorspace.



Obr. 74 – Export nastavení Substance Painter

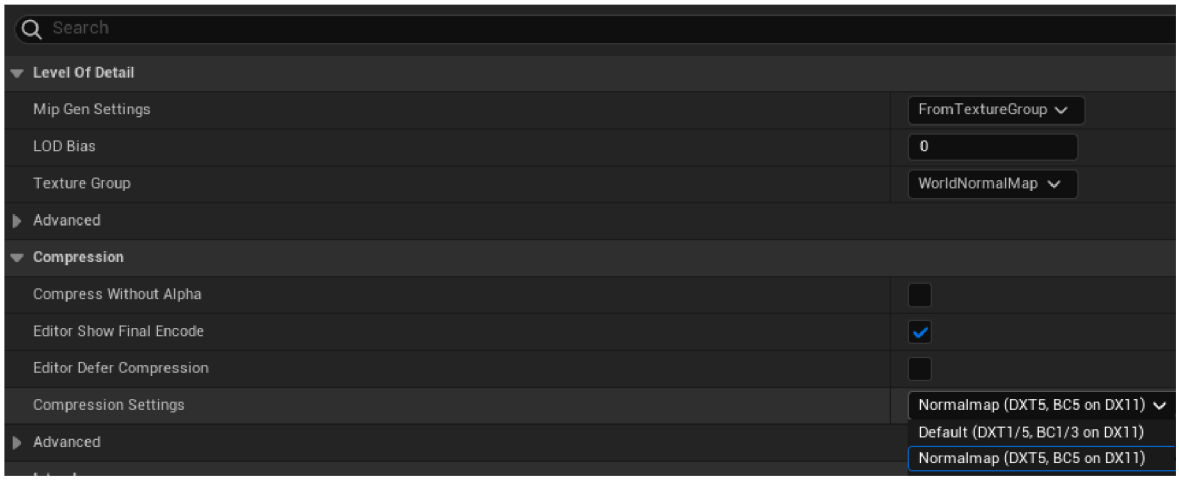
Dále, multipurpose mapy musejí využívat lineární colorspace. Pokud se naimportuje například ORM textura se zašrktlým sRGB checkboxem, pak ho změníme do lineárního colorspace odškrtnutím této kolonky.

RGB (tedy non-linear colorspace) se používá pro maximalizaci kvality omezeného 8 bitového obrazu. Pozvedne barevné hodnoty důležitějších tmavších částí obrázku, než se uloží. Protože lidské oči jsou citlivější na rozdíly v tmavších barvách, více než ve světlejších, získávají použitím tohoto colorspace tmavší barvy více prostoru, tedy více rozlišení, v omezených barevných datech RGB 0-255 (hodnoty, které může barva, vektor, nabírat).

PBR textury Roughness, Metallic, ale i Displacement či Normal mapy musejí mít matematicky korektní hodnoty, aby správně vyjadřovaly svá data. Tedy, nejde o barvy, které vidí člověk, ale o mapu počítačových dat, které vyjadřují metaličnost a jiné vlastnosti v rámci PBR pipeline. Tedy je musíme uložit jako rovnoměrně rozložené (lineární) hodnoty bez posunu v rámci sRGB. V lineárním rozložení mají jasnější hodnoty stejnou váhu jako ty tmavé.

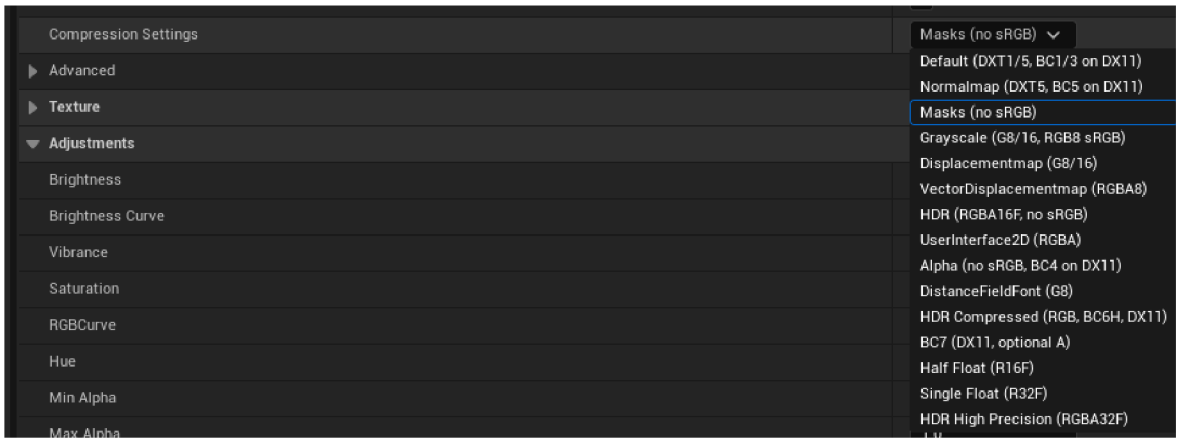
Automatický import textur a jejich nastavení můžeme buď řešit manuálně, či nastavením enginu. Ve verzi 5.2 dále, dle mého testování, se (většinou) nemusíme obávat

špatného importu textury. Vyjímkou, která se v mé zkušenosti opakovala, byl import ORM textur, které byly opticky převážně modré, a algoritmus automatického importu v Unreal Engine tedy usoudil, že jde o Normal mapy. Po uvědomění si této chyby stačilo změnit nastavení dané textury v texture editoru z NormalMap na Default.



Obr. 75 – Možnosti Texture Editoru v Unreal Engine

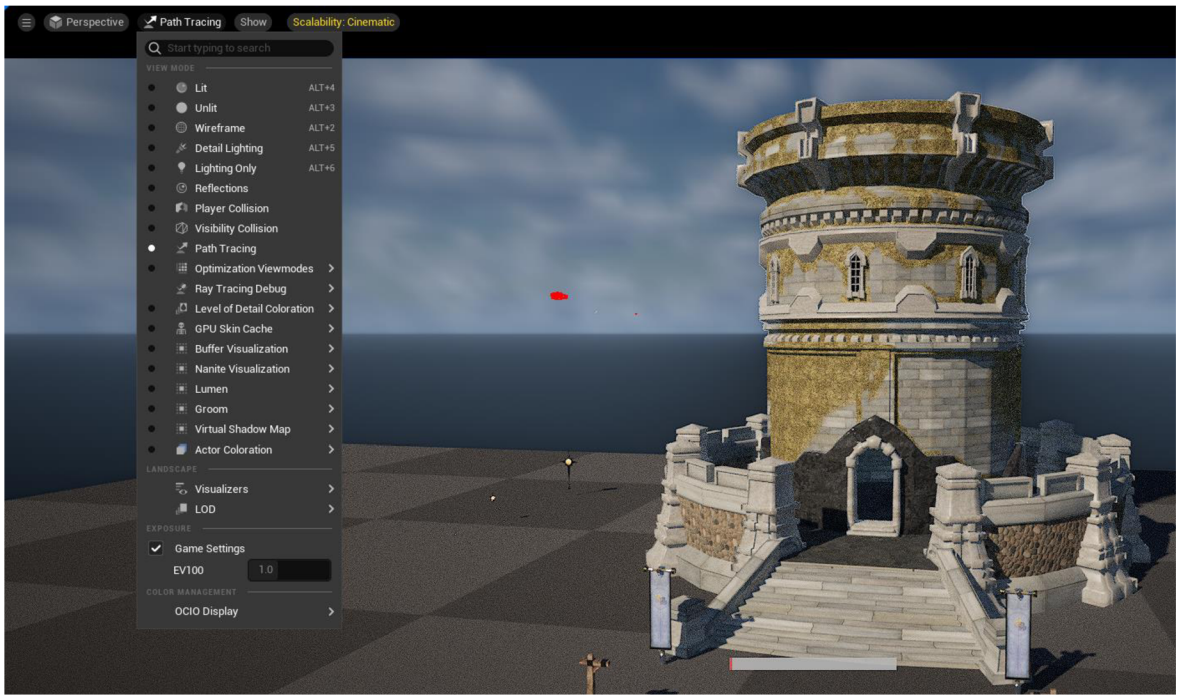
Poslední důležitou poznámkou u textur je využití tzv. VectorDisplacement nastavení komprese u displacement map. Displacement mapa, zaměnitelná za height mapu, je textura, nesoucí data ve formátu greyscale, tedy hodnoty 0-255, černá až bílá, pro vyjádření výšky. Bohužel, i pokud jde o import displacement textur z Quixel Bridge, Displacement Mapy automaticky nezískávají nastavení komprese VectorDisplacement, což znamená, že proces jejich komprese enginem způsobí zkreslení dat a tím pádem špatně vypadající materiály. Toto se dá opravit pouhým nastavením textury v Texture Editoru.



Obr. 76 – Compression settings v Unreal Engine Texture Editoru

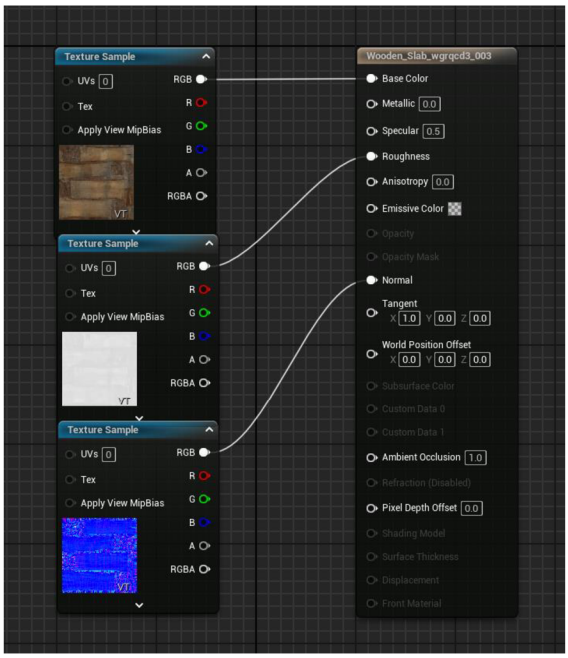
Look development, neboli lookdev, a jeho rychlost, je největší předností Unreal Engine. Díky okamžitému renderu můžeme získat představu o finálním vzhledu již

v náhledu 3D Viewportu při editingu scén. Především u prostředí, tedy environments, je lookdev velice rychlý a také jednoduchý. Žádné čekání na render, render máme přímo před očima. Pokud využíváme specializovaný renderer, jako např. Path Tracer, má viewport Unreal Engine speciální viewport i pro tento způsob offline renderingu.



Obr. 77 – Path Tracing Viewport v Unreal Engine

Import assetů, scén a modelů jsme si již popsali v předchozích kapitolách. Dalším stádiem look developmentu, ihned po importu a vytvoření Actors ve scéně, je vytvoření materiálů pro zmiňované actors, například modely a jejich blueprinty.



Obr. 78 - Jednoduchý materiál v Unreal Engine

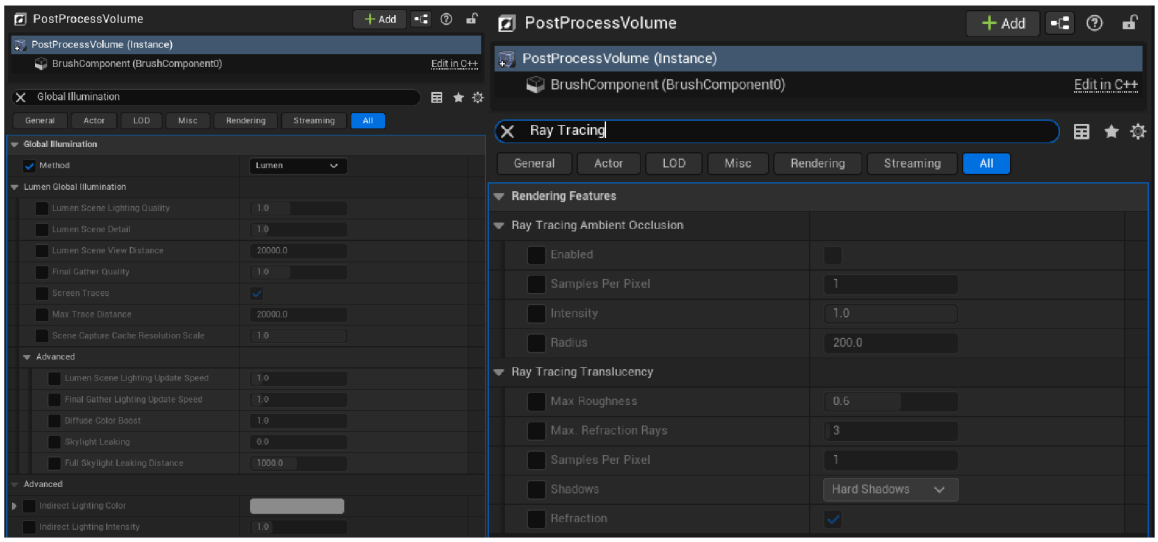
Rychlý lookdev zajišťují již zmiňované materiálové instance, Material Instances. Především, pokud využíváme Ubermaterial, při změnách v Master Material Instance měníme vzhled všech objektů, které využívají instance zmínění Master Material Instance najednou a okamžitě. Tento postup jsme již popisovali.

Lookdev také velice zásadně ovlivňuje in-engine post processing za využití tzv. PostProcessVolume, neboli PPV.

Obr. 79 – Možnosti PostProcessVolume v Unreal Engine

Jak napovídá název, jde o volumetrickou reprezentaci prostoru, ve které PPV ovlivňuje vzhled a rendering nastaveními, které bychom klasicky očekávali od finálního stádia pipeline studia, postprodukce. Můžeme tedy měnit atributy jako je Gamma, Highlights, Lens Flares a jiné vizuální efekty a tím měnit osvětlení a náladu scény. To ovšem není jediné využití PPV.

PPV, krom jednoduchých post-processingových filtrů a color-correctingu, má také vliv na samotný technický rendering scény. PPV totiž odhaluje spoustu parametrů renderu, a tím nám dává kontrolu, například nad počtem bounces při raytracingu. Jde tedy o naprosto nezbytný a neodmyslitelný nástroj pro lookdev.



Obr. 80, 81 – Nastavení global illumination a raytracing v rámci PostProcessVolume

Základní verze Unreal Engine také obsahuje atmosferické systémy, které urychlují lookdev, nechceme-li využít vlastní HDRi nebo jinou formu oblak. Prvním nástrojem je tzv. SkyAtmosphere, simulace atmosferického osvětlení scén. SkyAtmosphere je závislá na centrálním, hlavním zdroji světla, Slunci, neboli Directional Light. Pokud do scény přidáme Directional light a správně jej referencujeme v SkyAtmosphere, všimneme si, že v závislosti na rotaci tohoto světla se mění i jeho barva, potažmo barva atmosféry scény.

Krom bounce lightingu za využití raytracing, indirektní osvětlení enginu zajišťuje tzv. SkyLight. SkyLight využívá buď dat z textury, tedy HDRi mapy, či data zachycená při renderu samotné scény pro výpočet nepřímého osvětlení scény. Tato funkce je přímo propojena s SkyAtmosphere a zatím nezmíněným systémem Volumetrických mračen. Propojenost a vzájemná závislost systémů v praxi znamená, že lookdev atmosféry scény měníme pouhou změnou rotace slunce.

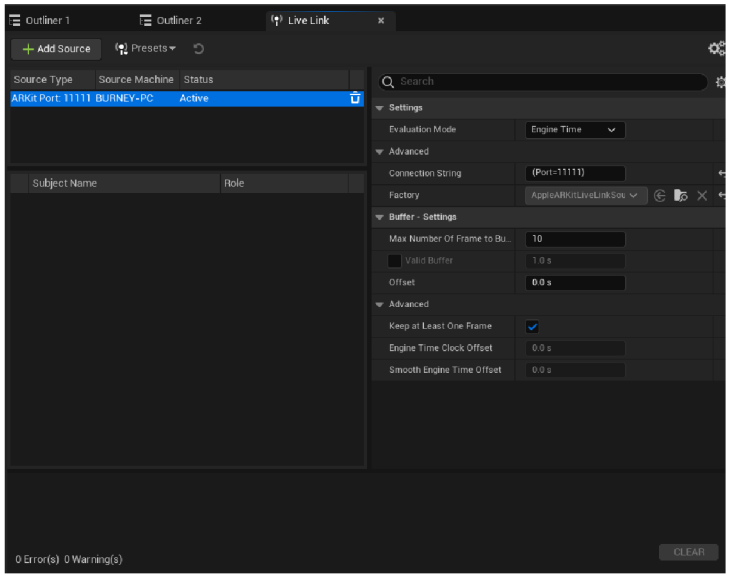
Volumetric Clouds, tedy volumetrická mračna, jsou dalším propojeným systémem pro look development v Unreal Engine. Název mluví za vše. Mraky interagují se světlem scény a mají simulovaný objem. Je nutné podotknout, že jejich art direction není tak propracovaný, jak by měl být, avšak jeho limitace se dají obejít za pomocí úpravy některých částí enginu.

Významný je také Exponential Height Fog. Simulace volumetrické mlhy závislé na definovaných hodnotách výšky mlhy. Height Fog je jedním z nejdůležitějších nástrojů pro look development a diskutabilně největším udavatelem nálady a atmosféry scén.

Zmiňované nástroje se také dají nahradit či doplnit ať už vlastními systémy, či systémy zakoupenými na Unreal Engine Marketplace, jako je například již zmiňovaný Ultra Dynamic Sky a Ultra Dynamic Weather systém.

Zbytek look developmentu prostředí je stejný jako v jakémkoliv jiném softwaru, tedy záleží především na dovednostech operátora.

3.4.1 LiveLink a Autodesk Maya



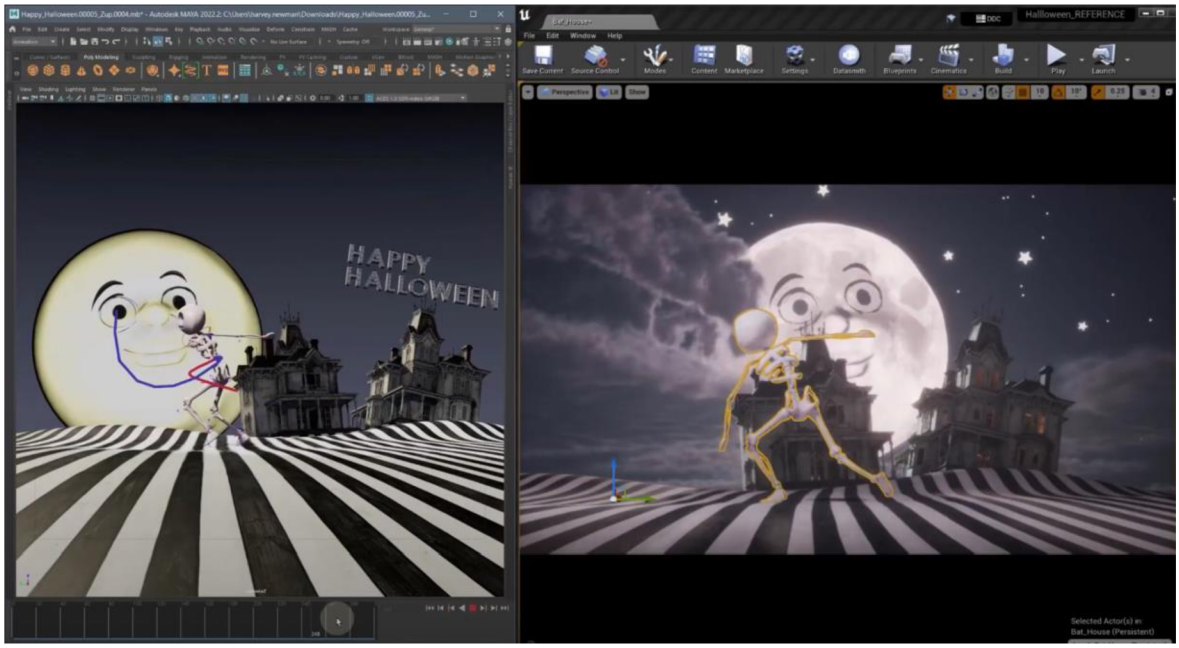
Obr. 82 – Rozhraní pluginu LiveLink v Unreal Engine

„LiveLink existuje již několik let. Původně byl vyvinut společností Epic Games. LiveLink pro Maya je tvořen ze dvou částí. Jde o jeden plugin pro Unreal Engine a druhý plugin pro Autodesk Maya. Při práci v tandemu Vám umožňuje streamovat data z Mayi do Unreal Engine.“[31]

LiveLink je nástroj namířený především na práci s animací, kdy rigovanou postavu animujeme v Autodesk Maya a tyto data jsou v reálném čase přenášena do Unreal Engine. Jedinou podmínkou je správně nastavený LiveLink a Animation Blueprint v Unreal Engine. Jako animační nástroj je LiveLink nejlépe využitý pouze pro přenos animované

[31] Unreal Engine. *Animation Workflows Using Unreal Engine and Maya | Webinar.* 2022 [YouTube video] [Citace: 20. Červen 2024]

postavy. Prostředí scény přenášíme pomocí FBX, OBJ, nebo USD, o kterém si ještě povíme.



Obr. 83 – Funkční LiveLink mezi Unreal Engine a Autodesk Maya

3.4.2 Budoucnost look developmentu – Universal Scene Description

Obr. 84 – Slide z Animation Workflows Using Unreal Engine and Maya | Webinar

„Pro ty z Vás, kteří USD neznáte, jde o rozšiřitelný popis scén a zároveň formát souboru. Open source řešení USD softwarů Maya a Unreal Engine umožňují uživatelům vytvářet, upravovat, pracovat a spolupracovat na datech USD. Existuje mnoho důvodů,

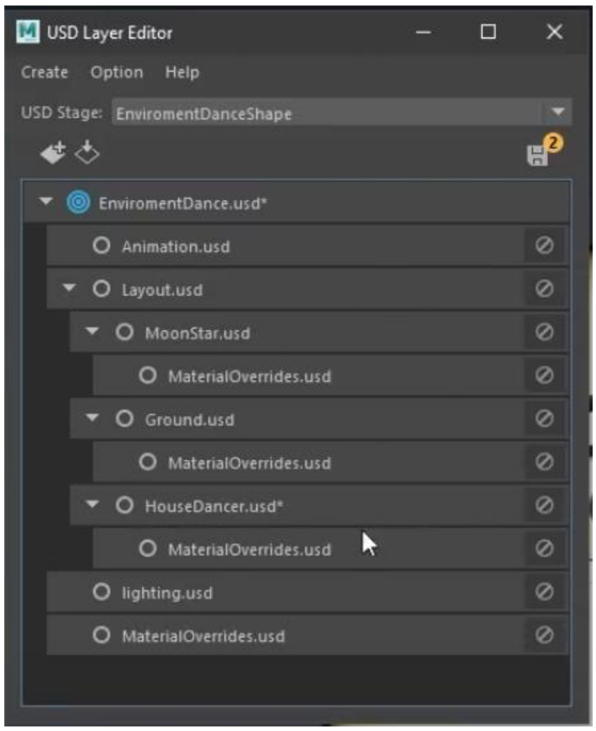
proč používat USD. Jeho rychlé načítání velkých datových sad, snadná spolupráce a fakt toho, že jde o společný jazyk, umožňující bezproblémový přesun dat mezi různými aplikacemi.“ [32]

Scénu a geometrii z Autodesk Maya exportujeme pomocí USD Export, který je nativně podporován. Ostatní nastavení jsou stejné, jakobychom exportovali FBX nebo OBJ. Klíčový rozdíl mezi OBJ nebo FBX a USD je spočívá v tzv. USD Stage.

„Chceme využít USD Stage. USD Stage je klíčovým konceptem. Na rozdíl od přímého importu vám umožňuje referencovat data USD, aniž byste je převáděli do nativního formátu aplikace, se kterou pracujete. Takže data nekonvertujeme do nativního formátu Maya nebo nativního formátu Unreal Engine. Data v USD přenášíte přímo a umžňuje Vám s nimi takto pracovat.“ [33]

Svým způsobem jde o rozdíl mezi editací geometrie souboru FBX přímo v Maye oproti editace modelu v Unreal Engine, ačkoliv jeho source asset stále existuje nezměněn na disku. Ještě lepším přirovnáním je rozdíl mezi práci s referencemi scén v Maya oproti editaci původní scéni v Maya. S USD daty pracujeme za využití USD Layer Editoru.

[32], [33] Unreal Engine. *Animation Workflows Using Unreal Engine and Maya | Webinar.* 2022 [YouTube video] [Citace: 20. Červen 2024]



Obr. 85 – USD Layer Editor v Autodesk Maya

„USD Layers (vrstvy) umožňují nedestruktivně provádět úpravy, pracovat s různými odděleními ve stejném kanálu za účelem spolupráce a aktualizace assetů bez vzájemného zasahování do práce, takže jde o celý management systém, který Vám umožňuje na velmi vysoké úrovni pracovat s daty USD.“ [34]

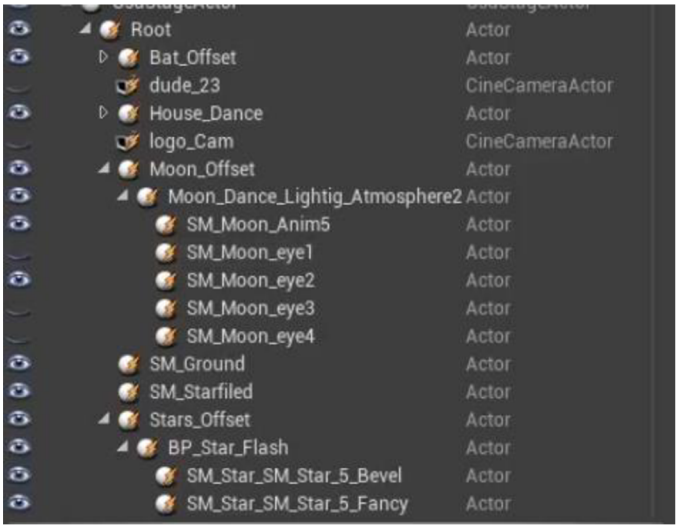
USD Stage soubor následně „importujeme“ do Unreal Engine, lépe řečeno jej otevřeme pomocí programu otevřeme.

„Zatímco se USD otevírá, mějte na paměti, že nejde o import těchto dat do Unrealu. Jde pouze o otevření reference na tato data, stále se bavíme o čistě datech USD, jen je máme možnost prohlížet, manipulovat s nimi a pracovat s nimi v rámci Unrealu. Takže nic nebylo importováno do Content Browseru, nebyly vytvořeny .uassets uvnitř Unreal projektu. Je to podobné jako chování, které jsme viděli uvnitř Maya, takže v podstatě jde o stejný pracovní postup v jakémkoliv programu, do kterého jdete s USD.“[35]

Právě ona univerzalita a nulová nutnost jakýchkoliv úprav jsou předností formátu USD. Co máme v Maya, budeme mít i v Unreal, a s daty se dá dále pracovat naprosto stejně, jako u

[34], [35] Unreal Engine. *Animation Workflows Using Unreal Engine and Maya | Webinar.* 2022 [YouTube video] [Citace: 20. Červen 2024]

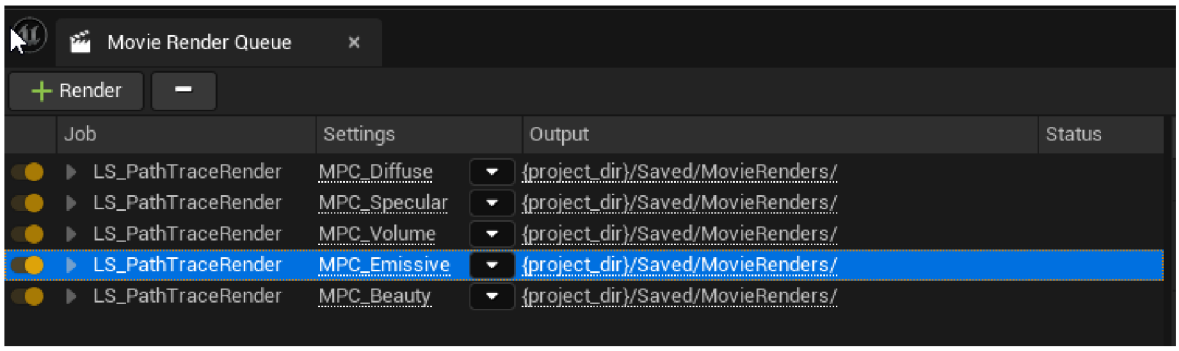
jiného formátu uvnitř Unreal Engine. Refernce USD dat jsou ve scéně reprezentovány formou Actors, stejně jako u jakýchkoliv jiných dat.



Obr. 86 – USD Actors v Unreal Engine

3.5 Render, render passes a postprodukce

Kvalitní render v Unreal Engine zajišťuje tzv. Movie Render Que (MRQ). Movie Render Que je nástroj, který je společně s Sequencer a Post Process Volumes zodpovědný za nastavení renderů a jejich postpném renderování. MRQ nabízí celou škálu nastavení a parametrů, které můžeme nastavovat dle vlastních potřeb a přání.



Obr. 87 – Menu Movie Render Que

Tradičně největším rozdílem mezi offline produkcí, např. za využití rendereru Arnold a rendery z herních enginů je neschopnost herních enginů vyprodukovat tzv. „render passes“ a „crypto-mattes“ pro následné užití a úpravu obrazu v postprodukci.

Tyto render passes dělíme na Diffuse, Specular, Volume, Emissive a celkovou tzv. Beauty. Offline produkce a postprodukce se bez nich jednoduše nemůže obejít a představují obrovskou výhodu tradičního offline přístupu k renderingu.
Jde o výstup dat, kdy renderované snímky představují data interakcí simulovaných světelných paprsků s povrchy ve scéně. Při jejich následném složení vzniká celý finální obraz, tedy Beauty pass. Pokud má postprodukce přístup k těmto datům, může za pomoci kompozičního softwaru, jako je např. Nuke nebo DaVinci Resolve tyto data manipulovat a tím měnit i vzhled vyrenderovaných dat a tím mít kompletní kontrolu nad výsledným obrazem. Protože jde o interakce sledovaných paprsků, Ray Tracing z definice není schopen outputu těchto dat. Pro fyzikálně korektní chování světelných interakcí se ohlédněme do práce *Unreal Engine a Průlomy v real-time renderingu* za Path-Tracingem.

Path-tracing je forma renderingu, která se pevně řadí do kategorie offline renderingu. To znamená, že výstup (obraz) se nezobrazuje na obrazovce ve skutečném čase a tudíž nemůže být interaktivní. Používá se výhradně pro filmovou produkci, či produkci, se kterou nebude nijak interaktivně zacházeno.

I path tracing se pořád drží pravidla vysílání paprsků z kamery. Avšak nyní nejde o jeden paprsek za každý jeden pixel, ale tisíce paprsků za každý jeden pixel. Dovedeme si tedy přestavit, o jak obrovských číslech se nyní bavíme.

Místo toho, aby se při dopadu paprsku generovaly paprsky další, path-tracing při průniku objektem dále odráží jeden a ten samý paprsek, co možná nejvícekrát. Tím se získávají informace o objektech, které jsou blízko u sebe a tím je možná i již zmiňovaná globální iluminace. Jedním z jejich největších vlastností je tzv. color bleeding. Jde o jev, kdy barva jednoho objektu se vlivem světla promítá na okolní povrchy.

Jednoduše řečeno, paprsky fungují naprosto stejně, jako světlo ve skutečném světě, když opomineme jejich vysílání z kamery. [36](Honal, 2022)

Naštěstí, od verze Unreal Engine 5.2, je Path Tracer Unreal Engine schopen výstupu Render Passes, i když jejich příprava je poněkud zdlouhavá.

Pro správnou přípravu renderu Render Passes musíme vytvořit nový Blueprint Class, který bude obsahovat Post Process Volume. Post Process Volume nastavíme tak,

[36] Honal, Kryštof Šimon. *Unreal Engine a Průlomy v real-time renderingu,* 2022. [Citace: 20. Červen 2024]

aby měl neomezený objem, tedy fungoval globálně a prioritně v celé scéně, kterou chceme renderovat. Uvnitř Event Graph Blueprintu, který daný PPV obsahuje, vytvoříme logiku, která určí rozdělení komponent Rays a Paths renderu.

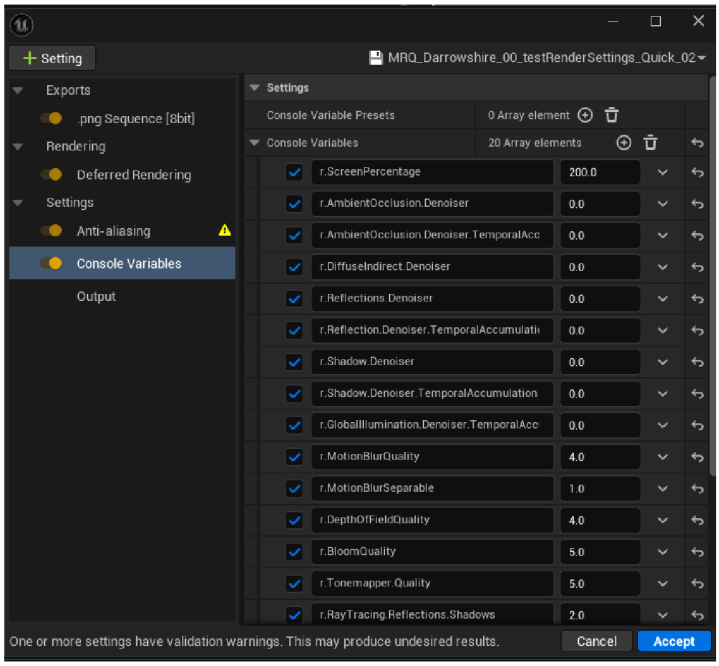
Možnosti nastavení PostProcessVolume nabízejí různé možnosti pro vytvoření různých Render Passes.

Bohužel, využití Render Passes vyžaduje X-násobný render té samé scény, kdy se při každém daném renderu renderuje jiný render pass. Číslo počtu renderu tedy odpovídá počtu render passes, které požadujeme.

Tyto render passes přidáváme do render que, tedy do „fronty“, chete-li, seznamu sekvencí k renderu. Díky funkci Render Que můžeme takové fronty vytvářet a nastavovat, a později tak renderovat scény bez supervize dle předem daného plánu.

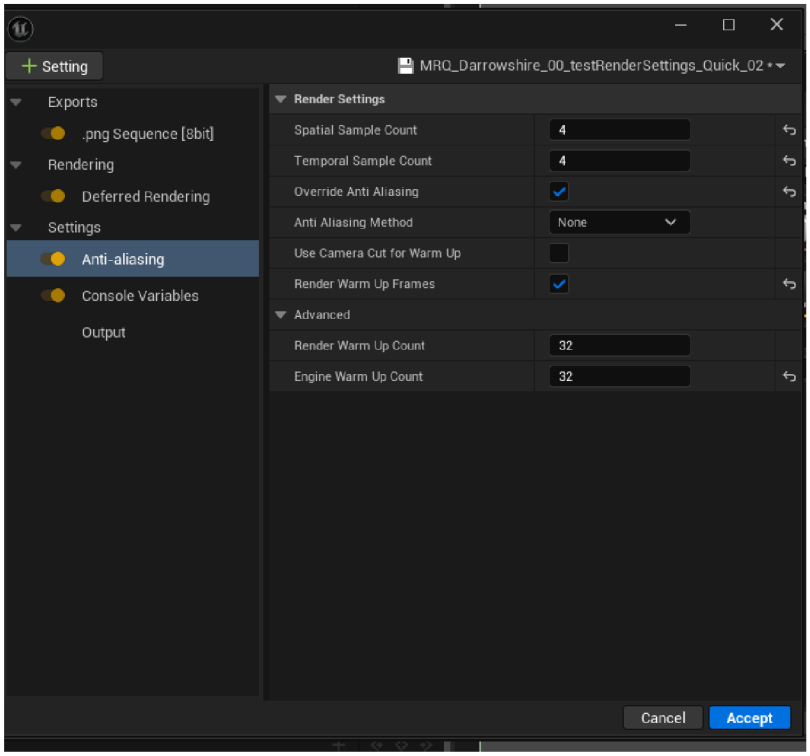
Obr. 88 a 89 – Logika Blueprintu obsahujíc PPV za cílem rozdělení paths a rays do různých render passes.

Movie Render Que také nabízí nesčetně možností a nastavení pro render obrazu.



Obr. 90 – Console Variables nastavení Movie Render Que

V rámci nastavení MRQ můžeme měnit výstupný formát, tedy EXR, PNG, či jiné formáty, způsob renderingu, tedy základy Deferred Rendering (Raytracing), či Path Tracer. Důležitým nastavením je Anti-Aliasing.



Obr. 91 – Nastavení Anti-Aliasing v Movie Render Que

Anti-Aliasing v real-time renderingu představuje pojem, kdy se matematicky snažíme zahlazovat ostré a zubaté hrany pixelů aniž bychom zvyšovali rozlišení obrazu, který renderujeme. V případě Unreal Engine renderujeme stejný snímek X-krát za využití tzv. subsamples, tedy jakýchsi „podsnímků“, jejichž data dále interpolujeme.

V Unreal Engine rozlišujeme dva druhy subsamples. Temporal Subsamples a Spatial Subsamples. Temporal Subsamples, jak napovídá název, jsou snímky zachycené během určitého časového úseku. Tím získáváme vysoce dobrou kvalitu obrazu velice levně, tedy za malou výpočetní cenu. Ze své definice mají ale tyto subsamples problémy s rychlými a prudkými akcemi. Využití Temporal Subsamples v takovém případě bude mít za následek rozmazané a špinavé snímky.

Spatial Subsamples považujeme za „skutečnější“ snímky, alespoň v porovnání s Temporálními snímky. Nejsou totiž závislé na časovém úseku. Spatial Subsamples fungují na základě mírných pohybů kamery, kdy při renderingu každého subsample se kamera nepatrně posune po osách prostoru. Interpolace těchto podsnímků pak vytváří kvalitní anti-aliasing, zato jde ale o výpočetně drahý přístup.

Je nutno podotknout, že počet Subsamples se navzájem násobí! Tedy, pokud využívám čtyř Spatial Subsamples a čtyř Temporal Subsamples, výsledně renderuji každý „skutečný“ snímek šestnáckrát! Mám tedy 16 Subsamples pro každý jeden renderovaný frame.

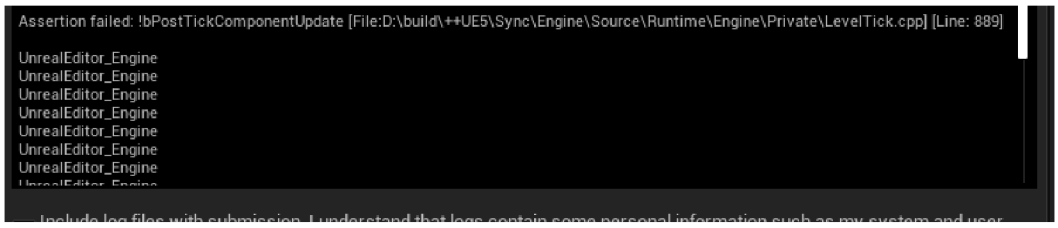
Dalším důležitým nastavením je možnost nastavení tzv. Render a Engine Warm Up Frames. Jde o počet snímků, který dovolíme enginu před tím, než začneme renderovat. Této funkce využíváme pro maskování načítání textur, či pokud logika systému, který využíváme, požaduje časovou prodlevu, například z důvodu načítání dat z internetu.

MRQ také nabízí možnost úpravy instance enginu, který bude scénu renderovat, pomocí tzv. CVARs, tedy Console Variables, česky, konzolových příkazů. Jde o textové vyjádření nastavení, které za normálních podmínek zajišťuje vizuální uživatelské rozhraní. V enginu jsou ale nastavení, která přes UI nejsou dohledatelná nebo se z jiných důvodů nenastavují či resetují při spuštění renderu. Taková nastavení zde pomocí konzolových příkazů nastavujeme. Seznam dostupných nastavení najdeme v oficiální dokumentaci Unreal Engine a učíme se je praxí.

Krom klasické postprodukce značnou míru efektů a změn obrazu zajišťuje již zmiňovaný Post Process Volume. PPV se zabývají i některé Marketplace assety. Mezi mé oblíbené patří FILMIFY, nástroj pro přidání zrna, či efektu halace do UE, nebo Chameleon, který je zaměřený spíše na postprodukci pro videohry.

4. BUGY, MOUCHY A CHYBY, ANEB KDYŽ TO NEJDE

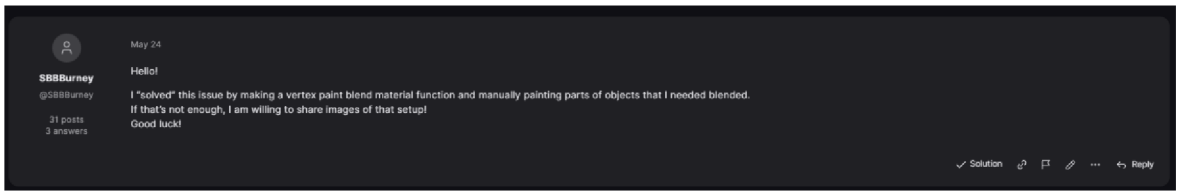
Chybám, bugům, špatně či vůbec nefungujícím funkcím a i crashes enginu jsem se rozhodl věnovat vlastní kapitolu. Bohužel je nutno podotknout, že Unreal Engine je často dost temperamentní a s jeho Crash Reportérem se tak v praxi setkávám často.



Obr. 92 – Crash report Unreal Engine

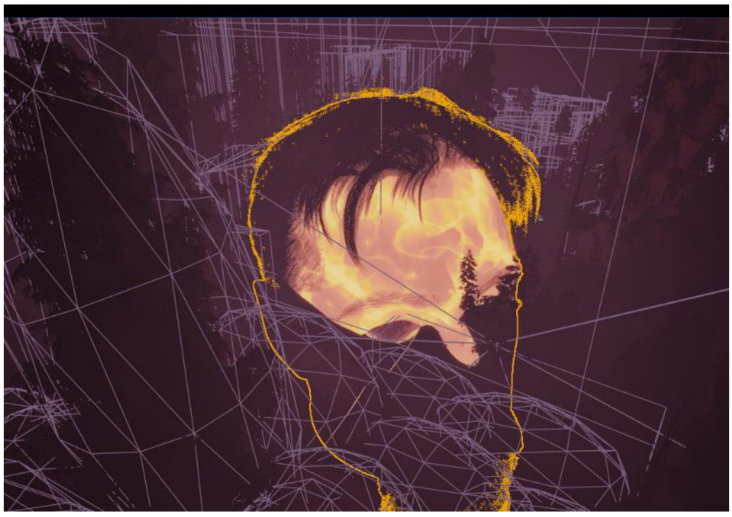
Unreal Engine je nástroj, který se rychle mění a s každým novým update přinese nové a někdy odnese staré. Mezi nové a staré často patří i chyby, omyly a nedopatření. S chybami „Assertion Failed" bohužel nemůže konečný uživatel nic dělat, není-li programátor. Je mou zkušeností, že tyto chyby se začaly ve velké míře vyskytovat po vydání verze 5.3, a verze 5.4.2, v době psaní této práce nejaktuálnější verze, jejich frekvenci bohužel zatím nezlepšila.

Krom crashes, je od verze 5.3 rozbitý systém Runtime Virtual Texturing. Jde o nástroj, díky kterému engine mohl manipulovat virtuální textury ze scény pro automatické prolínání povrchů s jemným přechodem. Jde o podobný systém, jako můj systém pro zahlinění plotů a weathering v projektu Darrowshire. Onen systém jsem vytvářel právě kvůli rozbitému RVT. Epic Games jsem již poslal několik bug reports a v zájmu diskuse jsem založil vlánkno na oficiálních forech Unreal Engine, ve kterém na daný problém upozorňuji, dokumentuji možný workaround a radím ostatním uživatelům, kteří se s chybou setkali.



Obr. 93 – Unreal Engine Forums

Tzv. „Attach Tracks“, tedy klíčování připojených objektů ve scéně v Sequencer je v aktuální verzi Unreal Engine rozbité. Radím Attach Tracks vůbec nevyužívat, jelikož zdánlivě fungují v editoru, ale při poslání na render vypovědí službu. Pokud se tomu tak stane v polovině třeba šesnáctihodinového renderu, má člověk velkou radost.



Obr. 94 – Rozbitý Attach Track hlavy postavy a jeho následky

Další problémy přinesla funkce Nanite tessellation, kdy se při přílišnému přiblížení kamery vůči teselovanému (=podrozdělenému povrchu v reálném čase) povrchu zhroutí ovladače GPU. Snad není třeba zmiňovat frustraci, kterou tento bug způsobil.

5. ZÁVĚR TEORETICKÉ ČÁSTI

Práce se věnovala pracovním postupům za využití Unreal Engine a jeho integrace do pracovních postupů s jinými programy. Text jasně vysvětluje a popisuje specifika práce a integrace enginu a navrhuje způsoby a profesně otestované postupy, jak programy propojovat a jak engine integrovat do existující pracovní pipeline.

Zároveň práce dává najevo svou hlavní myšlenku, a to, že pokud chceme využívat Unreal Engine v produkci, musíme se naučit jeho specifika, a k jeho využití mít pádné důvody. Unreal Engine je středobodem jakékoliv produkce, která jej využívá.

Text práce zachází do specifických podrobností a slouží jako manuál pro asset management v rámci Unreal Engine a na konkrétních příkladech tak tvoří šablonu pro práci s tímto programem. Tím práce naplňuje svůj cíl.

6. PRAKTICKÁ ČÁST

Praktická část diplomové práce má formu animovaného filmu vytvořeného v Unreal Engine, za využití pracovního postupu s propojením programů Blender, Autodesk Maya a samotného Unreal Engine.

Na tomto filmu jsem zastupoval roli „armády o jednom muži." Zpracovával jsem vše od 3D modelů, po animace. K projektu jsem přistupoval především jako Technical Artist, ale sám jsem zastoupil všechny role pipeline produkce animovaného filmu.

Námět na film je zpracován dle příběhu z počítačové hry World of Warcraft od Blizzard Entertainment. Jde o kultovní klasiku, ke které bylo během let vytvořeno mnoho komunitního obsahu, a zároveň i hudby. Darrowshire, píseň vyprávějící o stejnojmenném prokletém městečku, tvořil, nahrál a nazpíval písničkář Cranius již před šestnácti lety. Tato píseň, společně se hrou World of Warcraft, mne provázela většinu mého dosavadního života, a i proto jsem se rozhodl písni a autorovi vzdát hold formou animovaného filmu.

Obr. 95 – Screenshot z renderu Darrowshire

Píseň je adaptací příběhu z World of Warcraft. Duch malé Pamely je uvězněn v prokletém městečku Darrowshire poté, co je její otec zabit a proměněn v stín jeho bývalého já. Je osamocena a smutná, protože neví, kam se poděla její rodina. Hrdina náhodou narazí na jejího ducha při projíždění prokletou scenérií a slíbí jí, že najde jejího otce a panenku. Následuje dobrodružství s dobrým koncem, kdy jsou Pamela i její otec vysvobozeni a mohou konečně v pokoji odpočívat.

Postup výroby filmu je z velké části v práci již popsán. Práce využívá offline deferred renderingu za využití Unreal Engine 5.4.2. Postprodukce byla provedena přímo v enginu, nikoliv v postprodukčním programu.

7. ZÁVĚR

Cílem diplomové práce byla tvorba struktury, potažmo manuálu, pro bezproblémovou integraci programu Unreal Engine pro studia nebo jedince, kteří nejsou s pipeline herních enginů obeznámeni.

Abych odůvodnil a vysvětlil navrhované postupy, předložil a postavil jsem základní vlastnosti herních enginů a jejich kontrasty proti vlastnostem dedikovaných 3D animačních programů.

Čas jsem dále věnoval specifickým pracovním postupům s herním enginem Unreal Engine za účelem překlenutí a překonání oněch rozdílů.

Velká část práce je věnována specifickým workflows v rámci Unreal Engine a asset managementu v enginu, což zahrnuje vše od samotného importu obsahu, po jeho art direction, až finální render. Také jsem popsal všechny části vývoje animovaného projektu v herním enginu, a čtenáři jsem co možná nejvíce přiblížil svět enginů optikou VFX profesionála, zvyklého na klasický offline pipeline.

Úplným závěrem bych rád podotknul následovné:

Vzhledem k tomu, že jsem věnoval značné úsilí praktické části diplomové práce, mohl jsem díky získaným zkušenostem a datům precizně a podrobně zpracovat i část teoretickou.

Děkuji.

8. TERMINOLOGICKÝ SLOVNÍK

Rendering - vykreslování, renderování.

Renderer - program zodpovědný za renderování

Real-time rendering - vykreslování v reálném čase, okamžitě

Offline rendering – předpočítávání renderovaných snímků

Ray-tracing - proces renderingu

Path-tracing - proces renderingu

Herní engine - program pohánějíc počítačovou hru

Vertex - vrchol polygonu, bod v kartézské soustavě souřadnic

Face - rovina (geometrie)

Edge - hrana (linka)

Polygon - mnohoúhelník

Pixel - nejmenší součást obrazovky

Pixel shader - počítačový program, využívaný pro proces renderingu

Textura - data ve formě obrázku

Materiál – sada instrukcí Pixel Shader

PBR - rendering na základě fyzikálních vlastností, druh renderingu

Albedo - druh textury

Specular - druh textury

Metallic - druh textury

Roughness - druh textury

Normal - druh textury

AO , ambient occlusion - Metoda 3D grafiky, také stejnojmenný druh textury

Vektorový paramtr - soubor dat o třech parametrech

Skalární parametr - číselná data, určují velikost něčeho

Difúzní barva - jednolitá barva

Shading - stínování, metoda 3D grafiky

Alpha kanál - určuje průhlednost

Global illumination - nepřímé nasvětlení 3D scény

Lineární interpolace - metoda prokládání křivek (hodnot)

Vertex color - barva (parametry RGB + alpha kanál) s hodnotami určenými každému vertexu geometrické sítě

Mesh, neboli síť - triangulovaný model, specifikum herních enginů

Triangulace - proces konverze polygonů na trojhelníky

Ray-bouncing - odrazy paprsků

Post-processing - úprava obrazu prováděna jako poslední fáze renderingu

Render farma - skupina počítačů využívaná pro offline rendering

Asset - "aktivum", jakákoliv věc vyrobena za účelem užití v produkci

Level of detail - úroveň detailu.

Skeletální animace - animace za užití skeletálního rigu - kostry

Light functions - specifické funkce pro zdroje světla

Kismet - vizuální skriptovací jazyk

Blueprint - vizuální skriptovací jazyk

C++ - programovací jazyk

GPU – Graphics processing unit- grafický procesor

CPU – Central processing unit - centrální procesor

Draw call - forma komunikace mezi GPU a CPU

Baking - proces vytváření textur

Teselace - subdivize (podrozdělení) roviny aby vzniklo více geometrických tvarů

Occlusion culling - utrácení okluze - utracení, či zrušení schovaných (okludovaných) objektů

Visibility culling - utrácení objektů mimo zorné pole kamery

Draw distance - vzdálenost vykreslení

Clusters - skupiny trojúhelníků

Screen-space - postprocessingové kalkulace, pro jejichž vstup slouží data z prostoru obrazovky

Signed distance fields - zjednodušené modely 3D scén

9. BIBLIOGRAFIE

Conditional Zero. *Absolute Beginner's guide to Unreal Engine 5 Nanite,* 2021. [online]. [cit. 18-05-2024]. Dostupné z: https://youtu.be/9NgDle3J9lo

HONAL, Šimon Kryštof. *Unreal Engine a Průlomy v real-time renderingu.* Bakalářská práce. Praha: Vysoká Škola Kreativní Komunikace, Katedra vizuální tvorby, 2022.

Global Illumination | Unreal Engine Documentation, 2021[Online] Epic Games, Inc. [cit. 19-05-2024]. Dostupné z: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/GlobalIllumination.

Lumen Global Illumination and Reflections 2022 [Online] Epic Games, Inc. [cit. 19-05-2024]. Dostupné z: https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/.

Nanite Virtualized Geometry in Unreal Engine | Unreal Engine Documentation 2024, [Online]. Epic Games, Inc. [cit. 19-05-2024] Dostupné z:

Visibility and Occlusion Culling, 2021. [Online], Epic Games, Inc. [cit. 19-05-2024]. Dostupné z: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/VisibilityCulling/.

Haar, Ulrich a Aaltonen, Sebastian. *SIGGRAPH 2015: Advances in Real-Time Rendering in Games,* 2015. [Referát]. Los Angeles, California.

McDermott, Wes. *The PBR Guide: A Handbook for Physically Based Rendering,* 2018 [Kniha]. - [místo neznámé] : Allegorithmic. - 978-2490071005.

Lumen in UE5: Let there be light! | Unreal Engine, 2021. [Online] Unreal Engine [cit. 19-05-2024] Dostupné z: https://youtu.be/Dc1PPYl2uxA

Animation Workflows Using Unreal Engine and Maya | Webinar, 2022. [Online] Unreal Engine [cit. 20-06-2024] Dostupné z: https://youtu.be/Ddu7TAICAXw?si=OuzfDkvTQAe6c1yt

Using Visual Dataprep in Unreal Engine | Webinar, 2021. [Online] Unreal Engine [cit.20-06-2024] Dostupné z: https://youtu.be/g7LS8SaHmsI?si=A7G8WIq0VWlRweNV

UE4 Optimization: Instancing, 2017 [Online] Tech Art Aid [cit. 20-06- 2024] Dostupné z: https://youtu.be/oMIbV2rQO4k?si=B8bMAZfb5_iJOBu4