ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Diplomová práce

Aplikace pro prohlížení Langweilova modelu Prahy

Bc. Jan Tošner
Otevřená informatika - Počítačová grafika

Srpen 2023
Vedoucí práce: Ing. David Sedláček, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tošner** Jméno: **Jan** Osobní číslo: **483758**

Fakulta/ústav: **Fakulta elektrotechnická**

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

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Aplikace pro prohlížení Langweilova modelu Prahy

Název diplomové práce anglicky:

Application for viewing Langweil's model of Prague

Pokyny pro vypracování:

1) Seznamte se s daty Langweilova modelu Prahy. Jmenná konvence, struktura a rozsah dat, použité formáty. 3D Langweilův model dodá vedoucí práce.
2) Seznamte se s prostředím pro vývoj her Unreal 5. Soustřeďte se na optimalizační techniky pro zobrazení rozsáhlých a paměťově náročných modelů (např. virtuální textury, instance materiálů, LOD systém, progresivní načítání, paralelní rendering, pixel streaming, nDisplay, ...).
3) Navrhněte měření náročnosti zobrazování scény (fps, draw calls, paměť, ...).
4) Optimalizujte scénu s modelem s cílem snížit HW náročnost při zachování detailu a s předpokladem pro rendering na dvě obrazovky současně, jednu s 4K rozlišením a druhou FullHD (cca 2K). Postupně zaznamenávejte optimalizační kroky a zhodnoťte jejich důležitost a příspěvek ke snížení náročnosti scény.
5) Navrhněte a implementujte aplikaci pro prohlížení modelu na dvou obrazovkách. Jedna bude dotyková (horizontální stůl, 2K rozlišení), druhá bude pouze zobrazovací (4K rozlišení). Dotyková obrazovka bude umožňovat interakci s modelem, není nutné aby se na obou obrazovkách zobrazoval stejný pohled na model. Minimální implementovaná funkcionalita: pohyb po modelu (létání/chození), zapnutí předpřipravených průletů, znázornění významných míst a přepnutí do pohledu na ně, návrh nového významného místa, vytvoření si screenshotu z modelu, změna osvětlení.
6) Návrh a implementaci uživatelského rozhraní aplikace konzultujte průběžně s vedoucím práce a vybranými odborníky z Muzea města Prahy.
7) Proveďte uživatelské a výkonostní testování výsledné aplikace (s alespoň pěti uživateli).

Seznam doporučené literatury:

1] Kateřina Bečková, Miroslav Fokt. Svědectví Langweilova modelu Prahy. Schola Ludus Pragensia 1996.
2] 3D Reconstruction Data Set - The Langweil Model of Prague. Sedláček, D., Buriánek, J., Žára, J. International Journal of Heritage in the Digital Era. 2013, 2(2), 195-220.
3] https://docs.unrealengine.com/5.1

Jméno a pracoviště vedoucí(ho) diplomové práce:

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

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

Datum zadání diplomové práce: **17.02.2023** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **22.09.2024**

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

III. PŘEVZETÍ ZADÁNÍ

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

24. 5. 2023	
Datum převzetí zadání	Podpis studenta

Poděkování / Prohlášení

Děkuji Ing. Davidu Sedláčkovi, Ph.D. za vedení, pomoc a rady při zpracování této diplomové práce.

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

V Roudnici n. L. dne 14.08.2023

.....

Abstrakt / Abstract

Tento dokument je diplomová práce Jana Tošnera. Jejím cílem je vytvoření aplikace, která bude schopna zobrazit celý Langweilův model Prahy v Unreal enginu 5.

Hlavním úkolem je otestování všech možných způsobů optimalizace scény a následné vytvoření uživatelsky přívětivého grafického rozhraní.

Optimalizační kroky byly sjednocení materiálů, které využívali stejnou texturu, využití instancovaných materiálů, přeškálování textur na rozměry v mocninách dvou, převedení textur s velkým rozlišením na virtuální textury, LOD textur a Nanite. Pouze využití technologie Nanite mělo negativní vliv na výkon aplikace.

Po dokončení optimalizace se přešlo ke tvorbě samotné aplikace. Ta byla upravována na základě zpětné vazby. Nakonec proběhlo testování s pěti uživateli. Přestože všem chyběla jedna funkce, tak celkový dojem z aplikace byl velice pozitivní.

Klíčová slova: unreal engine 5, Langweilův model Prahy, optimalizace, render, diplomová práce.

This document is diplom thesis of Jan Tošner. Its goal is to create an application, witch will be able to show whole Langweil's model of Prague in Unreal engine 5.

The main task is to test bunch of ways of optimizing the scene and then create a user-friendly graphic interface.

Optimization steps were unification of materials that used the same texture, use of instanced materials, rescaling of textures to dimensions in powers of two, converting high resolution textures to virtual textures, LOD textures and Nanite. Only the use of Nanite technology had a negative effect on the performance of the application.

After the optimization was completed, it was time to create the application itself. It was modified based on feedback. At the end, testing was done with five users. Although everyone was missing one function, the overall impression of the application was very positive.

Keywords: unreal engine 5, Langweil's model of Prague, optimization, render, diplom thesis.

Title translation: Diplom thesis (Application for viewing Langweil's model of Prague)

/ Obsah

Tabulky / Obrázky

Kapitola 1
Úvod

Cílem této diplomové práce je vytvoření aplikace, která bude schopna zobrazit celý Langweilův model Prahy v Unreal Enginu 5[1]. Jelikož je tento model velice náročný na zobrazení, bude potřeba využít různé metody optimalizace.

Motivace

Vykreslování velkých modelů v počítači bylo vždy náročné. Pokusy o vytvoření aplikace pro zobrazení Langweilova modelu Prahy v reálném čase již proběhly, avšak nebyly úspěšné. Díky Unreal enginu 5 a jeho funkcím by mohla konečně tato aplikace vzniknout.

Největší problém celé aplikace je požadavek na zobrazení celého modelu z ptačího pohledu s možností plynulého přechodu na prohlížení detailů budov. Bez tohoto požadavku by šlo model rozdělit na několik částí a mezi nimi přepínat, což by velice snížilo nároky.

Využití

Výsledná aplikace bude součástí výstavy v Muzeu města Prahy[2]. Měla by umožňovat prohlížení modelu z blízka s velice uživatelsky přívětivým rozhraním.

Aplikace má využívat dva monitory. Jeden bude 4K televize, která bude sloužit k zobrazení modelu pro návštěvníky muzea. Druhý by měl být dotykový monitor, kterého účelem bude jednomu uživateli umožnit ovládání.

Jak již ze zadání vyplývá, tak musí obsahovat minimálně několik funkcionalit. Za nejdůležitější považuji umožnění pohybu ve 3D prostoru za účelem prohlížení modelu. Dále by měla aplikace umožňovat spuštění předpřipravených průletů scénou. Další vlastností jsou významná místa. Ta by se měla nějak zobrazovat, umožňovat přesun kamery na pozici s pohledem na dané místo a také dát uživateli možnost navrhnout nové místo. Návštěvník by si měl být schopen vytvořit snímek obrazovky a následně si ho nechat zaslat na email. Nakonec by měla aplikace obsahovat možnost změny osvětlení.

[1] Herní engine od firmy Epic Games `https://www.unrealengine.com/en-US/unreal-engine-5`
[2] Muzeum města Prahy `https://www.muzeumprahy.cz/`

Kapitola 2
Analýza

Je potřeba vytvořit aplikaci pro zobrazení Langweilova modelu Prahy. Abych dosáhl použitelného výkonu aplikace, budu muset optimalizovat různé prvky v projektu. Kvůli zjištění efektivity každého kroku je potřeba vymyslet způsob, jak sbírat data o výkonu aplikace po každé úpravě.

2.1 Langweilův model Prahy

Langweilův model Prahy vyrobil Antonín Langweil, který před jeho dokončením bohužel v roce 1837 zesnul [1].

Obrázek 2.1. Ukázka Langweilova modelu. Foto Miroslav Fokt

Papírový model

Antonín Langweil začal s výrobou papírového modelu Prahy v roce 1826. Inspiroval se sádrovým modelem Paříže, který zhotovil Symphorien Caron. Přestože rozpracovaný model pětkrát vystavoval, tak se mu nedařilo najít nikoho, kdo by jeho zálibu finančně podpořil [2].

Dva měsíce před svojí smrtí oslovil tehdejšího prezidenta českého gubernia hraběte Karla Chotku s prosbou, aby vystavil jeho dílo ve Vlasteneckém muzeu, které je dnes známo jako Národní muzeum. Tato prosba byla bohužel odmítnuta. Rozpracovaný model po jedenácti letech tedy skončil na půdě v devíti bednách.

Langweilova vdova v roce 1840 úspěšně prodala model Ferdinandovi V., který ho věnoval Vlasteneckému muzeu. Od něj ho následně převzalo Muzeum hlavního města Prahy, které ho vlastní dodnes. Model byl v letech 1963-1969 restaurován [2].

Na obrázku 2.1 můžete vidět Klementinum, kde Langweil žil a také zde pracoval na modelu Prahy.

Digitalizace

Na konci roku 2006 byla zahájena digitalizace modelu, která byla dokončena v roce 2009. Byl to velice technicky náročný projekt. Při rekonstrukci bylo nutné zvolit postup, který vyhovoval všem omezením. Papírový model mohl být mimo prachotěsnou vitrínu pouze v zimním období a také nesměl být venku déle než 3 měsíce. Bylo nutné využít pouze optickou metodu digitalizace, avšak barva na modelu je citlivá na infračervené a ultrafialové světlo [2].

Výsledný 3D model byl vytvořen za účelem prohlížení, zkoumání a vytvoření nových aplikací jako je například tato. Dále se dá model samozřejmě využít i jako podklad při rekonstrukci, kdyby se v budoucnosti nějaká část modelu poškodila.

2.1.1 Vlastnosti digitálního modelu

Model je vcelku veliký. Všechny potřebné soubory dohromady zabírají několik desítek GB[1]. Jelikož stěny jsou v podstatě obdélníky, což lze v počítači reprezentovat například dvěmi trojúhelníky, tak model nejde moc zjednodušit. Stromy byly velice zjednodušeny na válec s elipsoidem. Naleznete je na obrázku 2.2. Původní stromy byly štětinové, což v době digitalizace nebylo možné v takhle velkém množství použít, proto došlo ke zjednodušení.

Obrázek 2.2. Ukázka Langweilova digitálního modelu se stromy.

[1] Jednotka kapacity počítačové paměti.

Části

Model je rozdělen na 5 hlavních částí: Malá strana, Nové město, řeka, stromy a výplně. První dvě jsou následně ještě rozděleny na několik podsložek, které jsou pojmenované iniciály a číslovkou. Malá strana je rozdělena na ms_01–ms_04 a Nové město na nm_01–nm_06. Tyto složky již obsahují jednotlivé objekty. Budovy jsou mnohdy rozděleny do několika objektů a jsou umístěny v podsložkách částí města. Řeka uchovává povrh vody, ale i mosty. Výplně jsou povětšinou cesty nebo náměstí. Stromy i výplně jsou reprezentovány jako jeden objekt.

Materiály

Každý objekt v modelu má svůj takzvaný materiál. Ten určuje vlastnosti povrchu pro render. Příkladem těchto vlastností je například průhlednost, matnost, lesklost a hlavně barva. Barva je mnohdy nahrazena texturou. Většina objektů obsahuje více materiálů. Například jeden materiál může popisovat zeď a druhý střechu.

Názvy

Téměř celý model dodržuje pravidla pro názvy. Čas od času lze narazit na nějakou vyjímku, kterou je potřeba ošetřit. Většina objektů dodržuje šablonu: `část__blok_identifikátor`. Někdy záleží i na umístění, kde se soubor nachází.

Příkladem může být `N3__19_205850`:
- N3 => úmístěn ve složce *model_nove_mesto/nm_03/*
- 19 => identifikátor bloku budov
- 205850 => identifikátor daného objektu

Jak již bylo řečeno, tak objekty mají přidružené materiály, kterým odpovídají textury. Názvy těchto souborů mají šablonu: `_blok_identifikátor_typ-materiálu_typ-souboru`. Typ souboru může být buď *MG* nebo *tx*.

Typ materiálu může nabývat těchto hodnot:
- 0 => střechy
- 1 => stěny budov
- 2 => komíny
- 3 => samostatné zdi
- 4 => jiné označení pro stěny budov
- 6 => jiné označení pro stěny budov
- 9 => ostatní

Dále se v názvu mohou objevovat slova, která mají vyšší váhu a tudíž se vyhodnocují přednostně.
- base => povrh, převážně cesty a náměstí
- zeme => povrh, převážně zeleň

Pokud se daný objekt nachází v části řeky, tak zde platí jiné šablony:
- base => povrh řeky
- 9 => sochy
- 25 => mosty

Příkladem může být `_19_205850_0_MG`:
- 19 => identifikátor bloku budov
- 205850 => identifikátor daného objektu
- 0 => typ materiálu je střecha
- MG => jedná se o materiál

Následující soubor se nachází v části řeky `_00_base3_tx`:
- 00 => identifikátor bloku
- base3 => jedná se o třetí kus řeky
- tx => jedná se o texturu

Poslední vzor, který ještě nebyl zmíněn je pro komíny. Tento vzor je ve tvaru `_2_x`. Odpovídající objekty obsahují na konci svého identifikátoru ještě i písmeno navíc. Budova totiž může mít více komínů a takto se rozlišují.

Problém

Hlavní problém spočívá v texturách, které jsou ve vysokém rozlišení. Při renderování scény se totiž musí tyto textury načíst do mezipaměti grafické karty, ale v takovémhle množství se tam bohužel nevejdou všechny. Naštěstí je zde prostor pro zlepšení. Je zde mnoho textur, které se opakují. Dalším problémem je to, že spousta textur nemá rozměry v mocninách dvou, což umožňuje lepší práci s texturami.

Komíny

Ve scéně se nachází pouze 10 druhů komínů. Každý druh má stejný model, materiál i texturu. Bohužel každý komín má svoji kopii těchto souborů. To nám nabízí jednu velice jednoduchou optimalizaci. Můžeme vytvořit pouze 10 materiálů a ty následně přiřadit odpovídajícím komínům.

2.2 Herní engine

Herní engine[1] je program, který se využívá primárně k vývoji her. Má na starosti vykreslování snímků, fyzikální simulaci, přístup k periferiím a mnoho dalšího. Periferie je například monitor, reproduktor, klávesnice, myš nebo ovladač.

Na úplných počátcích se hry programovaly od samého základu. Později si velká herní studia začala vytvářet vlastní herní enginy[3]. Jelikož většina her potřebuje stejný základ, tak vznikly herní enginy určené k tomu, aby je využívala ostatní herní studia. Využitím tohoto nástroje se ušetří hodně času a peněz.

Další výhodou jsou takzvané assety (digitální aktivum). Jsou to balíčky k danému enginu, které většinou vytvořila třetí strana. Například balíčky 3D modelů se dají většinou použít ve všech dostupných enginech. Jiné assety mohou obsahovat scripty, které jsou ale závislé na enginu.

Herních enginů je spousta, ale většina jich není dostupná pro veřejnost. Dostupné jsou například Unity[2], Unreal Engine 4/Unreal Engine 5[3] a Godot[4]. Přehled těchto enginů jsem nalezl ve článku TOP 5 Herních enginů pro indie vývojáře [4].

[1] https://en.wikipedia.org/wiki/Game_engine
[2] https://unity.com/
[3] https://www.unrealengine.com/en-US/unreal-engine-5
[4] https://godotengine.org/

Základní myšlenkou je, že vývojář vytvoří scénu do které přidává herní objekty a těm upravuje jejich chování za pomoci takzvaných scriptů[1]. Je možné herní engine přirovnat k divadlu, kde scéna je jeviště, herní objekty jsou herci a scripty jsou scénář. Tyto scripty jsou vlastně malé programy, které reagují na dění ve scéně. Programovací jazyk, ve kterém se dají psát, je specifikovaný daným enginem. Součástí scény je také osvětlení, kamera a ozvučení.

Unity

Unity je herní engine, který bývá první volbou malých herních studií. Jeho hlavními přednostmi jsou velká komunita, cena a hlavně dlouholetost, díky které se postupnými aktualizacemi stal velice silným a uživatelsky přívětivým nástrojem. Využívání Unity je zcela zdarma, pokud je váš výdělek menší než $100 000 ročně. Z těchto předností také vyplývá, že je internet plný tutoriálů a řešení problémů na fórech, což velice usnadňuje vývoj. Scriptovacím jazykem je C#[2].

Unreal Engine 4 a Unreal Engine 5

Unreal Engine je na rozdíl od Unity spíše volbou velkých herních studií. Je vyvíjen společností Epic Games. Nejvíce se proslavil díky hře Fortnite[3], která je vyvíjena stejnou společností a to je také hlavní předností tohoto enginu. Díky vývoji této hry se engine zlepšuje neuvěřitelnou rychlostí. Jeho cena je stanovena na 5% ze zisku každé čtvrtletí. Programuje se v jazyce C++[4] nebo za pomoci takzvaných blueprintů[5], což je vizuální programování. Tyto dva způsoby se dají kombinovat.

Godot

Godot je open-source[6] engine, který je zcela zdarma. Tento velice uživatelsky přívětivý engine podporuje rovnou několik programovacích jazyků, mezi které patří například C++, C# a hlavně GDScript[7], což je jazyk vytvořený přímo pro Godot. Velice se podobá Pythonu[8]. Jelikož je z těchto tří nejmladší, má ještě hodně nedokonalostí. Přestože 2D grafiku zvládá výborně, 3D je znatelně horší a podpora VR je teprve v raných fázích.

Tento souhrn jsem převzal ze své bakalářské práce[5] a následně mírně upravil.

2.3 Unreal Engine 5

Jak již ze zadání diplomové práce vyplývá, tak pro tento projekt byl zvolen Unreal Engine 5[9]. Jeho vydání v dubnu roku 2022 umožnilo vzniku velice náročných aplikací. Tato verze posunula herní průmysl na novou úroveň, protože přinesla mnoho nových mocných nástrojů na vývoj her. Mezi ty nejdůležitější patří určitě Nanite a Lumen. Další výhodou Unreal Enginu je možnost spouštění Python scriptů, které mohou upravovat scénu, čímž ušetří spoustu práce.

[1] https://cs.wikipedia.org/wiki/Skript_(programování)
[2] https://learn.microsoft.com/cs-cz/dotnet/csharp/
[3] https://www.epicgames.com/fortnite/
[4] https://isocpp.org
[5] https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/
[6] https://cs.wikipedia.org/wiki/Otevřený_software
[7] https://docs.godotengine.org/en/stable/index.html
[8] https://www.python.org/
[9] https://www.unrealengine.com/en-US/unreal-engine-5

Nanite

Nanite[1] je nástroj, který umožňuje zobrazení mnoha modelů s vysokým počtem polygonů. Modely v počítačové grafice se obvykle zjednodušují na mnoho polygonů neboli mnohoúhelníků. Běžně je polygon trojúhelník.

Nanite v podstatě nahrazuje optimalizaci pomocí LOD[2]. Všechny úrovně detailů se totiž za normálních okolností musí dělat ručně nebo poloautomaticky. Složité modely většinou mají několik úrovní podle počtu trojúhelníků. Následně se během renderování vybere jedna úroveň podle vzdálenosti od objektu, protože model v dálce nemusí být tak detailní jako model metr od kamery. Nanite tutu funkci nahrazuje plně automatickou generací úrovní za běhu programu s minimální ztrátou kvality.

Nanite běží celý na grafické kartě. Nejdříve si rozdělí model na shluky 128 trojúhelníků. Pro každý shluk vytvoří bounding box[3]. Pomocí něho může následně určovat, zda je daný shluk viditelný na kameře. Dále vytvoří stromovou hierarchii shluků tak, že rodič je zjednodušenou verzí jeho potomků. Při běhu aplikace nalezne řez stromem podle procentuálního obsazení shluku na obrazovce. Ten nemusí být vždy na stejné úrovni, takže je možné, že model nebude mít jednotnou úroveň detailů.

Pokud má model veliké množství malých trojúhelníků, tak nastane problém při vykreslování. Když je totiž jeden trojúhelník malý jako jeden pixel na výsleném obrázku, tak staré techniky vykreslování přestanou být efektivní. Hardware je totiž přizpůsoben na rastrování trojúhelníků na několik pixelů. Proto tedy vytvořili softwarový rasterizér, který zvládá malé trojúhelníčky až 3x rychleji. Při vykreslování se tedy Nanite u každého shluku rozhoduje, který rasterizér využije podle jeho velikosti.

Informace jsem čerpal z Inside Unreal [6].

Jelikož Langweilův model je v základu velice jednoduchý a problém je spíše v texturách a materiálech, tak předpokládám, že Nanite moc nepomůže. Každopádně toto je pouze odhad, který budu muset ověřit.

Lumen

U objektů, které se nehýbou můžeme předpočítat osvětlení předem a následně těmito předpočítanými daty urychlit vykreslování. Avšak výpočet světel a stínů pohybujících se objektů je náročný, protože u nich ho nelze předpočítat. Naštěstí stačí rozdělit objekty na statické a pohyblivé. U statických si osvětlení předpočítáme a při vykreslování dopočítáme pouze pohyblivé, kterých nebývá mnoho. Problém nastává, když se pohybuje světlo. V tom případě není možné cokoliv předpočítat a aplikaci mohou klesnout snímky za sekundu na tak nízkou hodnotu, že se již ani nedá považovat za aplikaci v reálném čase.

Novým nástrojem Unreal Enginu 5 je Lumen, což je jejich řešení vypočítávání dynamického osvětlení v reálném čase. Renderuje jak difuzní odrazy s nekonečným počtem odrazů, tak nepřímé zrcadlové odrazy. Navíc funguje ve velkých detailních prostředích v měřítku od milimetrů až po kilometry. Objekty ve scéně blokují i nepřímé osvětlení, což vytváří nepřímé stíny. Pokud v nastavení zakážeme statické materiály, tak následně můžeme využívat „Material Ambient Occlusion“. Tato funkce umožňuje objektům vrhat stíny sám na sebe.

Základem fungování Lumenu je sledování paprsku (ray tracing) a voxelový systém. Při použití sledování paprsku dochází k simulaci chování světla prostřednictvím simu-

[1] `https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/`
[2] Level of Detail - úroveň detailu
[3] bounding box - kvádr, který se využívá k zjednodušenému vůpočtu kolizí

lace paprsků světla, což zahrnuje odrazy, lom světla, stíny a globální osvětlení. Tímto způsobem mohou být scény nasvěcovány mnohem realističtěji a detailněji. Tento styl osvětlení je však velice náročný na výkon a proto je vhodný spíše pro offline vykreslování.

Lumen je tedy zjednodušený tak, aby mohl být využit pro aplikace běžící v reálném čase. Využívá konkrétně 3 různé typy sledování paprsků. První kontroluje odrazy v hloubkové paměti, což je pole vzdáleností objektů v každém pixelu. Tento způsob sleduje pouze objekty na obrazovce. Zbylé pracují s celou scénou.

Druhý pak sleduje vzdálenostní pole všech objektů. Pokud je však objekt daleko od kamery, tak se využije třetí způsob a to pomocí globálního vzdálenostního pole. To je velice zjednodušená voxelová reprezentace celé scény.

Voxelový systém je klíčovým prvkem pro rychlé a efektivní zpracování osvětlení. Voxelová reprezentace scény umožňuje rozdělit prostor do malých voxelů, což jsou 3D ekvivalenty pixelů.

Požadavky na výkon vyžadují maximálně 0.5 paprsků na pixel. Avšak aby scény uvnitř budov vypadaly dobře, tak vyžadují alespoň 200 paprsků na pixel. Lumen tedy využije již zmíněné 3 způsoby pro sledování více paprsky na 1/16 pixelů. Výsledek pak spojí s vysoce detailní normálovou mapou, čímž vznikne finální obrázek. Pro lepší představu se podívejte na obrázek 2.3, který jsem získal z prezentace Inside Unreal [7], kde také naleznete více detailů.

Obrázek 2.3. Podvzorkování Lumenem. Zdroj: [7].

Ve výchozím nastavení je použit pro globální osvětlení i systém odrazů. Jeho výsledky jsou velice realistické. Lumen funguje i v případě, kdy je použit Nanite.

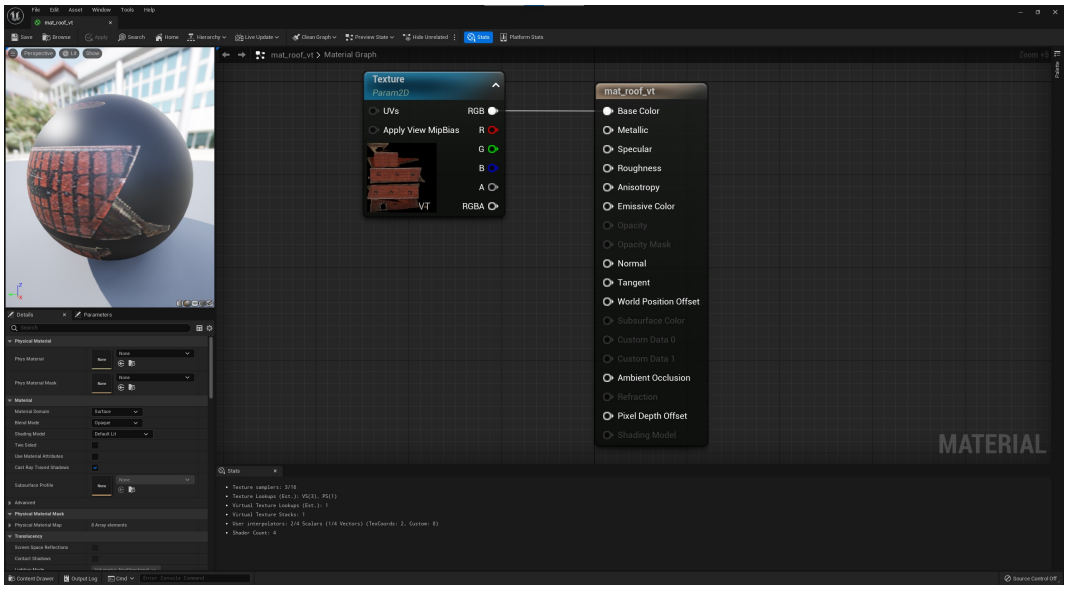
Materiály

Materiály určují způsob, jakým se bude daný objekt vykreslovat. Každý objekt může mít přiřazen několik materiálů. Kolik materiálů bude model využívat se určí již při tvorbě daného modelu. Ten lze vytvořit například v programu určeném pro tvorbu 3D modelů. Příkladem může být Blender[1], který je zcela zdarma, nebo placená Maya[2] od

[1] `https://www.blender.org`
[2] `https://www.autodesk.cz/products/maya`

firmy Autodesk. Propojení mezi materiálem a částí modelu se určuje také při tvorbě modelu.

Materiály v Unreal enginu se definují pomocí vizuálního programování. To spočívá ve vytváření bloků, které vykonávají jim odpovídající funkci. Bloky mohou být například Texture Sample nebo Vector. Jednotlivé bloky jsou následně propojovány a tvoří řetězec. Na levé straně bloku se nachází vstupy a na pravé výstupy. Poslední blok odpovídá výchozímu materiálu, který je upraven vstupy na levé straně. Na obrázku 2.4 je vidět materiál, který se skládá ze dvou bloků. Modrý blok upravuje barvu materiálu pomocí textury. Béžový blok představuje výchozí materiál.



Obrázek 2.4. Ukázka úpravy materiálu v Unreal Enginu.

Instancované materiály

Materiály jsou mnohdy téměř totožné až na nějakou malou změnu. Rozdíl bývá většinou v textuře. Unreal engine umožňuje tvorbu takzvaných instancovaných materiálů. Ty se vytvoří z jednoho normálního materiálu.

V nastavení instancovaného materiálu lze upravovat bloky, které mají vlastnost parametr. Tedy „TextureSample“ upravovat nelze, ale „TextureSampleParameter2D“ již upravit jde. Hlavní výhodou takto vytvořených materiálů je, že lze upravit jenom pár parametrů bez nutnosti znovu sestavit celý materiál.

Další výhodou je jednotnost všech materiálů stejného typu. Můžeme totiž mít jeden materiál pro střechu a následně několik instancovaných materiálů pro každou střechu ve scéně. Když se později rozhodneme, že chceme například upravit materiál střech tak, aby vypadaly mokře, stačí nám upravit pouze základní materiál a tato změna se projeví na všech střechách.

Textury

Textura je obrázek, který se využívá jako vstup pro materiály. Hlavně se používá jako zdroj pro barvu, avšak lze s ní upravovat i jiné parametry. Například průhlednost, normály nebo kovovost. V podstatě se definují hodnoty, které nejsou konstantní po celé ploše modelu.

Nejběžnějším způsobem definování modelu je pomocí trojúhelníků. Vrcholy těchto trojúhelníků se nazývají vertexy.

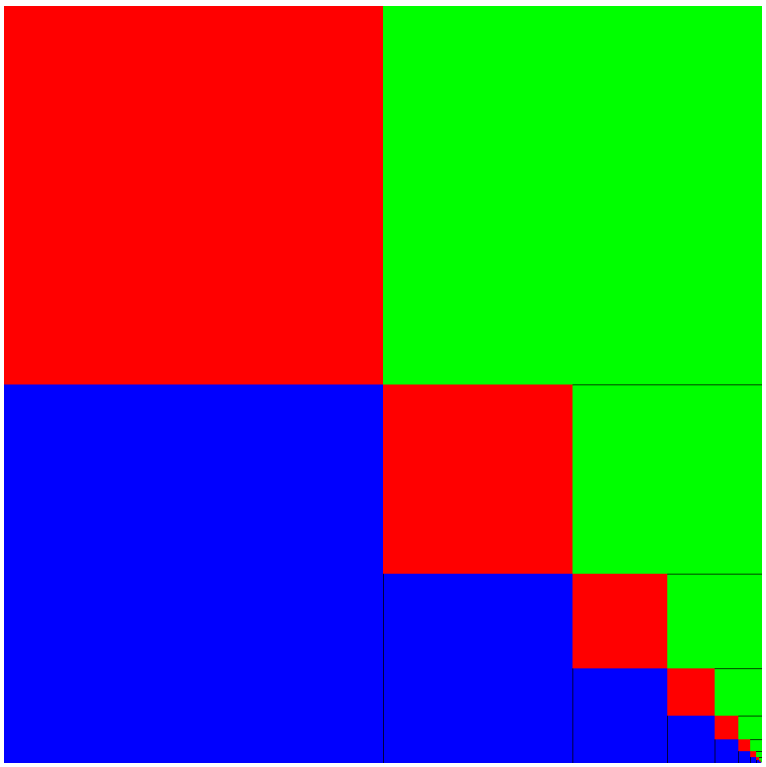
Textura se mapuje na objekt pomocí takzvaných UV souřadnic. Ty nabývají hodnot od 0 do 1 pro obě osy. 0 odpovídá prvnímu pixelu v dané ose a 1 poslednímu. Každý vertex má svoji UV souřadnici. Při renderování trojúhelníku se získávají souřadnice interpolací souřadnic jeho vrcholů.

Mipmapping

Čím dále od objektu jsme, tím méně nás zajímají detaily v textuře. Pokud tedy nějaký model zabírá ve výsledném renderu například 20 pixelů, tak nepotřebuje mít texturu o velikosti 2048x2048px. Můžeme tedy například podle vzdálenosti od objektu využívat jinou texturu a tím ušetřit hodně místa na grafické kartě.

Mipmapping je technika, která využívá tuto ideu. Benjamin Anuworakarn ve článku „Why you really should be using mipmapping in your graphics applications“[8] popisuje mipmapy takto: „Mipmapy jsou menší, předfiltrované verze textury, které reprezentují rozdílné úrovně detailu dané textury. Jsou často uloženy jako sekvence postupně menších textur zvané řetězce mipmap s každou úrovní o polovinu menší než předchozí.“ Takovýto řetězec mipmap zvýší velikost obrázku zhruba o 33%.

Můžeme si představit čtverec, kde rozdělíme červenou, zelenou a modrou složku do čtvrtin daného čtverce. Vznikne nám jedna prázdná čtvrtina, do které můžeme vložit všechny menší mipmapy se vždy poloviční velikostí, než předchozí. Pro ilustraci jsem vytvořil obrázek 2.5.

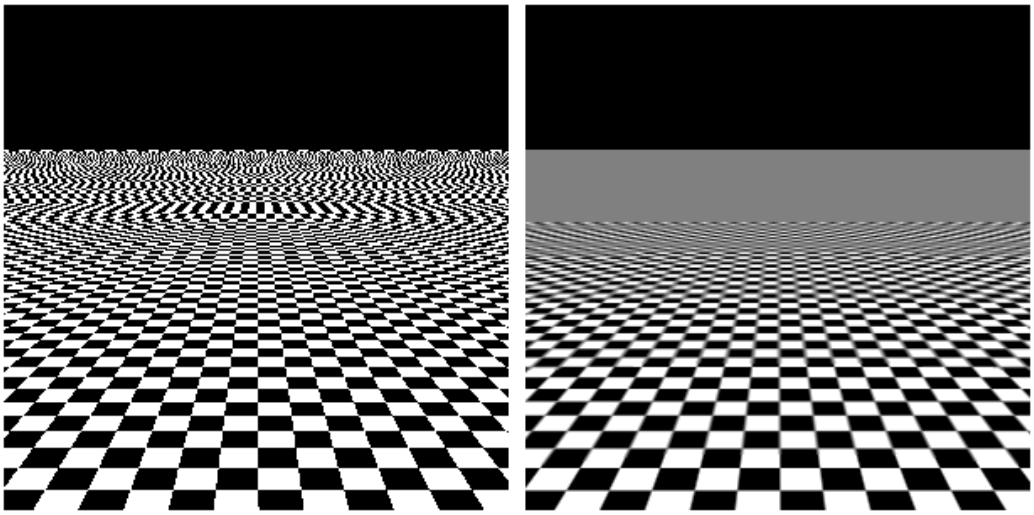


Obrázek 2.5. Ilustrace principu mipmapy.

Ve stejném článku se také dozvíme, že mipmapy mají dvě hlavní výhody:

- vylepšení kvality obrazu – Mipmapping může pomoci vyřešit efekt aliasingu způsobovaného převzorkováním textury. Na obrázku 2.6 lze vidět porovnání stejného obrazu s rozdílem, zda byl využit mipmapping.

- vylepšení výkonu aplikace – Mipmapping zvýší efektivitu mezipaměti, protože textura v plné kvalitě nebude potřeba neustále. Díky tomu bude v mezipaměti více menších textur a tudíž se paměť nepřeplní tak často.



Obrázek 2.6. Vlevo bez použití mipmappingu – vpravo s použitím mipmappingu. Zdroj: `https://textureingraphics.wordpress.com/what-is-texture-mapping/anti-aliasing-problem-and-mipmapping/`

Jsou i případy, kdy se mipmapping nevyplatí používat. Jedním z nich jsou objekty, které se nikdy nevzdálí od hráče dostatečně daleko, aby se využila horší textura. Jasným příkladem je uživatelské rozhraní.

Generaci mipmap lze dělat dvěma způsoby. Buď můžeme zmenšit texturu v průběhu již spuštěné aplikace, nebo si je předvytvoříme. První způsob nám sníží velikost aplikace, ale zvýší nároky při běhu aplikace. Jelikož cílem mojí práce je optimalizace, tak předvytvořené mipmapy jsou lepší volba. Unreal Engine vytváří mipmapy automaticky pro všechny textury, které mají rozměry v mocninách dvou.

Virtuální Textury

Virtuální textury[1] se rozdělují na dva typy: „Runtime Virtual Textures“ a „Streaming Virtual Textures“.

Runtime Virtual Textures[2] slouží k aplikování textury přes jinou při běhu aplikace. Tento typ textur bych využil například na sprejování, které je populární v moderních hrách.

Nás ale zajímá hlavně Streamování Virtuálních Textur(SVT)[3]. To snižuje zaplnění paměti při používání textur velkých rozměrů. Je to alternativa k využívání mipmap.

Klasický mipmapping určuje úroveň textury podle vzdálenosti k objektu. Navíc vždy když je model viditelný, tak načte celou texturu. Tento způsob zbytečně využívá hodně místa. Kdyby byl vidět například pouze komín, tak se načtou textury pro celou budovu i když nebudou nikdy využity.

[1] `https://docs.unrealengine.com/5.0/en-US/virtual-texture-memory-pools-in-unreal-engine/`
[2] `https://docs.unrealengine.com/5.0/en-US/runtime-virtual-texturing-in-unreal-engine/`
[3] `https://docs.unrealengine.com/5.0/en-US/streaming-virtual-texturing-in-unreal-engine/`

Obrázek 2.7. Ukázka ohraničení virtuálních textur.

SVT rozdělí obrázek na dlaždice o různých velikostech, které jsou vždy v mocninách dvou. Následně se do grafické karty posílají pouze dlaždice požadované velikosti a ještě k tomu pouze ty, které jsou vidět ve výsledném renderu. To znamená, že pokud se budeme dívat například na budovu, tak se textury z druhé strany budovy do grafické karty vůbec nedostanou a tudíž se ušetří místo. Na obrázu 2.7 je ukázka scény s ohraničením již zmíněných dlaždic. Různé barvy značí různou velikost dlaždic. Aby Streamování Virtuálních Textur mohlo být použito, tak je zapotřebí, aby daná textura měla oba rozměry v mocninách dvou.

Paralelní rendering

Paralelní rendering[1] využívá vlákna k větší efektivitě renderování. Za běžných okolností se vykreslování provádí ve „vykreslovacím vlákně“. Pokud je paralelní rendering povolen, tak toto „vykreslovací vlákno“ pouze vytváří příkazy a ukládá je do seznamu příkazů. Následně vykreslovací hardwarové rozhraní tyto příkazy vykonává pomocí vhodné API[2] pro danou platformu. Díky tomuto rozdělení není potřeba rozlišovat zda aplikace běží na počítači nebo například na konzoli.

Pixel streaming

Pixel streaming[3] umožňuje ovládání aplikace pomocí webového prohlížeče. Po nainstalování daného rozšíření je nutné aplikaci sestavit a spustit s argumenty, které určují IP adresu a port. Následně lze z nějakého jiného zařízení danou webovou stránku navštívit a ovládat aplikaci na dálku. Uživatel tedy využívá výkon jiného stroje. Navíc také nezáleží na operačním systému, který využívá uživatel. Jelikož konečná aplikace je určena k umístění na jednom místě, tak tato technika není vhodná.

[1] `https://docs.unrealengine.com/5.0/en-US/parallel-rendering-overview-for-unreal-engine/`
[2] Application Programming Interface
[3] `https://docs.unrealengine.com/5.0/en-US/pixel-streaming-in-unreal-engine/`

Dva monitory

Způsobů, jak spustit aplikaci na dva monitory není mnoho. Nabízí se pár možností, které jsou dostupné jako rozšíření. Bohužel ani jedna z nich není zdarma.

Další možností je spuštění aplikace v okně bez okrajů a pomocí konzolového příkazu změnit rozměry okna tak, aby vyplňovalo oba monitory. Tento způsob však zobrazuje jednu kameru přes oba monitory. To lze vyřešit pomocí takzvaného „split sceenu“. Ten se většinou využívá pro rozdělení obrazovky, pokud hraje více hráčů na jednom monitoru, ale každý musí mít jiný pohled. Obrazovku lze rozdělit uprostřed horizontálně i vertikálně. Problém však nastává, když monitory nemají stejné rozlišení. V tu chvíli totiž rozdělení uprostřed není mezi monitory, ale někde v obraze většího monitoru. Jelikož cílem je využití 2K a 4K monitoru, tak toto řešení tedy nelze využít.

Poslední možností na kterou jsem přišel je nDisplay. Ten je sice zbytečně mocný nástroj na můj účel, avšak je zdarma a lze ho využít.

nDisplay

Rozšíření nDisplay[1] je určeno k renderování scény na několik monitorů. Tyto monitory mohou být i připojeny k více počítačům. Každý si následně renderuje na své obrazovky, takže ve výsledku se i rozloží požadavky na výkon mezi více strojů. Umožňuje také vykreslování na několik monitorů připojených k jenomu zařízení, což je přesně vlastnost, kterou využiji.

Po instalaci rozšíření přibydou do enginu dva nové programy: Switchboard a Switchboard Listener. Switchboard Listeren musí běžet na všech zařízeních, kde jsou připojené monitory, na kterých se bude aplikace zobrazovat. Switchboard se využívá k nastavení aplikace, připojení k posluchačům a k následnému spuštění. Pro nastavení je potřeba nDisplay config, který se vytvoří uvnitř Unreal projektu jako běžný blueprint. Lze spustit jak projekt během vývoje, tak sestavenou aplikaci.

Nevýhodou je nutnost znát rozlišení všech monitorů již při tvorbě nDisplay configu. Není to tedy moc flexibilní. Další špatnou vlastností je, že Switchboard ani Switchboard Listener nelze nainstalovat jako samostatný program. Je tedy nutné stáhnout Unreal Engine, vytvořit projekt a přidat do něj rozšíření nDisplay, čímž se do daného počítače stáhne.

2.4 Sběr dat

Abych mohl vyhodnotit dopad jakékoliv optimalizace na výkon, tak ho je potřeba po každé větší změně měřit. Jelikož je sběr dat pořeba dělat již od samého začátku, tak toto je první věc, které se budu muset věnovat.

Způsobů, jak shromažďovat data o výkonu je několik. Po důkladném průzkumu jsem zúžil výběr na 3 hlavní možnosti.

Sledování zevnitř

Jak již bylo řečeno v sekci 2.2, tak Unreal Engine umožňuje psát vlastní scripty v jazyce C++, takže první jasnou možností je napsat si vlastní sledovač uvnitř samotné aplikace. Tím bych získal naprostou kontrolu nad tím, které vlastnosti budu sledovat a jak si je uložím. Vzhledem k tomu, že se sledováním výkonu nemám žádné zkušenosti, tak bych to musel všechno od základu nastudovat. Druhý problém je to, že tento typ

[1] https://docs.unrealengine.com/5.0/en-US/rendering-to-multiple-displays-with-ndisplay-in-unreal-engine/

sledování by běžel uvnitř aplikace, čímž by negativně ovlivňoval její běh a zkresloval by výsledky.

Pozitiva:
- Kontrola nad výběrem sledovaných vlastností.
- Kontrola nad stylem uložení.

Negativa:
- Nutnost napsat celý sledovač sám.
- Zkreslené výsledky.

Sledování zvenčí

Druhou možností je sledování jinou aplikací. Tento způsob má jako hlavní výhodu to, že neovlivňuje běh aplikace. Samozřejmě zde také platí výhoda v kontrole nad volbou sledovaných dat a výsledném uložení. Například by šlo tuto aplikaci napsat v jazyce Python[1], ze kterého by se výsledná aplikace spustila a následně sledovala. To podobně jako v prvním případě vyžaduje umění sledování výkonu aplikace, které by bylo potřeba doplnit.

Pozitiva:
- Kontrola nad výběrem sledovaných vlastností.
- Kontrola nad stylem uložení.
- Neovlivňuje výsledek měření.

Negativa:
- Nutnost napsat celý sledovač sám.

Unreal Insights

Posledním způsobem je program Unreal Insights, který je přímo součástí Unreal Enginu. Tento program se umí připojit ke každé Unreal aplikaci a sledovat její výkon. Daná aplikace, ale musí být vybuilděna v debug režimu. Při spuštění aplikace je potřeba jako argument zadat, které balíčky vlastností chceme pozorovat. Již v průběhu měření se v uživatelském rozhraní vykreslují grafy všeho možného. Po ukončení aplikace se data uloží do souboru, který lze načíst a znovu prohlédnout všechna data na již zmiňovaných grafech. Hlavní výhodou tohoto řešení je, že pokud se v budoucnu rozhodnu, že bych chtěl sledovat ještě nějakou jinou vlastnost, tak už ji budu mít naměřenou. Pro zpětné testování stačí zálohovat všechny verze aplikace. (Není potřeba zdrojový kód.)

Pozitiva:
- Vše už je hotové.
- Sbírá veliké množství dat.

Negativa:
- Nemám kontrolu nad způsobem uložení dat.
- Nutnost buildu v debug módu.

[1] `https://www.python.org/`

2.5 Související práce

3D RECONSTRUCTION DATA SET - THE LANGWEIL MODEL OF PRAGUE

Tato práce se věnuje postupu, jakým byl Langweilův model zrekonstruován. Experti z muzea z důvodu ochrany originálního modelu zakázali využití standardních typů skenerů. Proto byl vytvořen speciální robot, který automaticky pořizoval fotografie částí modelu z různých úhlů. Celý proces trval dva měsíce, během kterých bylo pořízeno téměř 250 000 fotografií ve 4K rozlišení (4096 x 4096 px).

Dále se zde autoři věnovali postupu jak z daných fotografií získali 3D model. Více informací naleznete přímo ve článku [9].

Kapitola 3
Návrh řešení

3.1 Sběr dat

Po poradě s vedoucím Diplomové práce jsem se rozhodl využít program Unreal Insights. Je to profesionální nástroj, který je neustále vylepšován přímo vývojáři Unreal Enginu.

Pro účely porovnání je potřeba, aby byla data naměřena za stejných podmínek. Proto mám v plánu udělat několik průletů kamerou, které se automaticky zapnou po spuštění aplikace.

3.2 Optimalizace

Před tím, než začnu pracovat na samotné aplikaci, je potřeba scénu s Langweilovým modelem optimalizovat. Již v analýze v sekci 2.3 jsem zmínil několik technik, které by mohly pomoci. Zde tedy popíšu jednotlivé kroky, které se chystám vyzkoušet.

Materiály komínů

Jak jsem již zmiňoval v analýze v sekci 2.1, tak modely komínů se opakují. Je jich celkem 10 typů. Momentálně každý komín má svou vlastní texturu, která je naprosto totožná s texturou ostatních komínů stejného typu. Těchto komínů je celkem 8036. Při záběru na celé město se musí načíst do paměti všechny materiály komínů. Grafická karta tedy musí zbytečně načítat o 8026 textur navíc.

Abychom ušetřili paměť v grafické kartě, tak můžeme vytvořit 10 materiálů, které použijeme u odpovídajících komínů. Tím pádem všechny modely dohromady budou využívat jenom 10 textur.

Instancované materiály

Další možnost optimalizace jsou Instancované materiály[1]. Jde o to, že můžeme vytvořit jeden materiál a z něj udělat instanci, která bude mít pouze změněnou nějakou vlastnost. Tato vlastnost může být třeba barva. To lze využít například znovu u komínů, kdy budeme mít jeden materiál typu „komín“ a 10 instancí tohoto materiálu. Abych ušetřil zbytečný krok navíc, tak rovnou při sjednocování materiálů využiji instancované materiály.

Mám v plánu převést na instancované materiály i všechny ostatní, které se ve scéně opakují. V analýze modelu jsem u názvů zmiňoval, že každý materiál má ve jménu typ materiálu. Ten mohu využít k určení materiálu, který lze využít jako rodiče pro daný instancovaný materiál.

Instancované materiály by měly být spíše pro usnadnění práce s velkým množstvím materiálů. Na výkon by měl být dopad minimální, avšak pokud nějaký bude, tak předpokládám, že nebude negativní. To si ale budu muset ověřit až po testování tohoto kroku.

[1] https://docs.unrealengine.com/5.0/en-US/instanced-materials-in-unreal-engine/

Celkem je tedy potřeba 11 materiálů:

- mat_base
- mat_bridge
- mat_chim
- mat_chimney
- mat_nine
- mat_river
- mat_roof
- mat_side
- mat_stat
- mat_wall
- mat_zeme

Jak můžete vidět, tak materiál pro komíny je zde dvakrát. To je způsobeno tím, že občas se objeví i komín, který je specifický pro konkrétní budovu. Rozhodl jsem se tedy oddělit tyto komíny a již zmíněné komíny, které sdílí stejné textury. Specifickým komínům odpovídá `mat_chim`.

Škálování textur

Jak jsem již zmiňoval v analýze v sekci 2.3, tak je potřeba, aby textura měla rozměry v mocninách dvou, pokud ji chceme překonvertovat na virtuální texturu. Je tedy nutné, abychom přeškálovali nebo doplnili obrázky na mocniny dvou.

Doplnění jednotné barvy lze udělat přímo uvnitř Unreal Enginu. V nastavení obrázku nalezneme položku „Power Of Two Mode“, která umožňuje tři režimy: „None“, „Pad to Power Of Two“ a „Pad to Square Power Of Two“. První režim nedělá nic. Zbylé dva doplní chybějící pixely jednotnou barvou, kterou lze zvolit. Výchozí barva je černá. „Pad to Square Power Of Two“ navíc vynucuje i doplnění na čtverec. Tedy jeho rozměry musí být oba stejné. Například obrázek o velikosti 1000x512px by byl doplněn na 1024x1024px, přičemž režim „Pad to Power Of Two“ by ho doplnil pouze na 1024x512px.

Tento způsob však rozhodí UV souřadnice a tudíž ji nelze přímo použít v materiálu. Pokud si všad uložíme u každé textury poměr rozměrů stran před zvětšením vůči novým rozměrům, tak lze UV souřadnice upravit v materiálu. Tento způsob sice funguje, ale na úkor neustálého přepočítávání UV souřadnic během renderování.

Druhou možností je tedy přeškálovat všechny obrázky v externím programu. Tento způsob neovlivní UV souřadnice, protože ty se pohybují v rozmezí 0-1. Dají se tedy představit jako procenta a tudíž je natažení textury neovlivní. Každopádně je zde možné, že při škálování dojde k rozmazání výsledné textury.

Kvůli velkému množství textur nepřipadá v úvahu zvětšovat tyto textury ručně. Je tedy nutné napsat script například v Pythonu. Původně jsem měl v plánu vyexportovat všechny potřebné textury z projektu. To lze ve formátu TGA. Nakonec jsem ale využil originální textury, které byly ve formátech PNG a JPG. Využitím originálních textur dojde k menší ztrátě kvality.

Virtuální textury

Textura je automaticky převedena na virtuální texturu, pokud při importu má oba rozměry alespoň 4096px. To znamená, že v projektu jsou již nějaké překonvertovány.

Pokoušel jsem se někde dohledat doporučenou minimální velikost textur vhodných pro využití virtuálních textur, ale bohužel nikde nic takového není. Z většiny stránek jsem však odešel s dojmem, že čím více, tím lépe. Avšak jelikož nejmenší dlaždice má rozměry 128x128px, tak nemá smysl převádět menší textury na virtuální.

Zkusím tedy převést všechny textury, které mají alespoň jeden rozměr větší než 512px na virtuální. V případě, že se něco pokazí, tak začnu od textur s oběma rozměry alespoň 4096px a následně budu postupně snižovat tuto hranici, dokud se nedostanu na první robitou verzi. V tu chvíli použiji předchozí verzi.

Úroveň detailů

Úroveň detailů, neboli Level of Detail (LOD), lze využít jak u modelů, tak u textur. Jeho aplikování u textur v Unreal Enginu je naprosto triviální. Jediná věc, kterou je nutno udělat, je mít textury s rozměry v mocninách dvou. Unreal Engine následně vytvoří mipmapu automaticky.

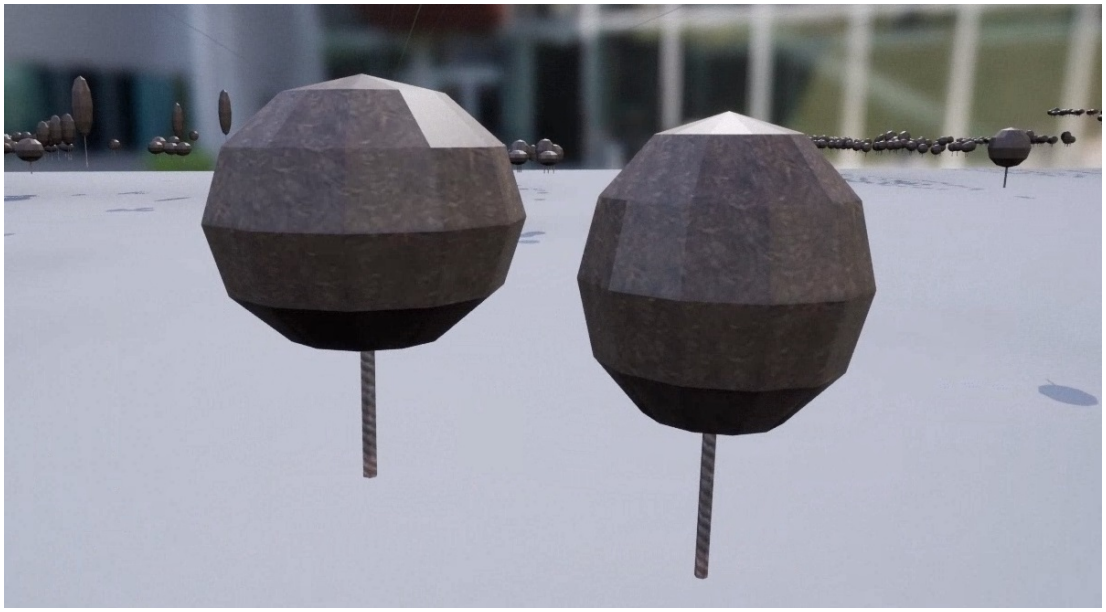
Nanite

Nanite je způsob automatické tvorby LOD pro modely. Jeho aplikace spočívá pouze v zaškrtnutí políčka „Enable Nanite Support“, která se nachází v nastavení modelu v záložce „Nanite Settings“.

Je nutno ověřit, jak bude aplikace běžet s aktivním Nanite na všech objektech a následně porovnat s neaktivním Nanite. Jak jsem již zmiňoval v analýze, tak předpokládám, že na tomto modelu Nanite nebude dobře fungovat.

Stromy

Stromy jsou speciální v tom, že se nebudou optimalizovat, ale spíše naopak. Nynější elipsoidy, které můžete vidět na obrázku 3.1, budou nahrazeny za modely stromů, které více odpovídají originálu. To bohužel přitíží této aplikaci ve vykreslování a o to lepší bude muset být předchozí optimalizace.



Obrázek 3.1. Ukázka původních stromů v Langweilově modelu.

V projektu jsou všechny stromy spojené do jednoho objektu. To je velice nepříjemná situace, protože nemám jak zjistit, kde se jednotlivé stromy nachází. Jejich polohu potřebuji zjistit, abych věděl, kam mám umístit nový model stromů.

Další informace, která je zapotřebí pro náhradu je velikost daného stromu. Jak můžete vidět na obrázku 3.1, tak Pravý strom je více šišatý než ten levý. Pokud se podíváte do levého horního rohu, tak zde zahlédnete ještě více protáhlý strom do výšky. Bohužel stejně jako u pozice nemám způsob jak ji získat.

Od vedoucího práce jsem dostal k dispozici několik souborů s příponou `.wrl`. V nich se nacházely skupiny stromů. Každý strom ve skupině je reprezentován jedním řádkem. Ten obsahuje informace ve tvaru: `translation x y z scale x y z`. Na daném řádku se nachází více věcí, ale pouze tyto nás zajímají.

Po experimentování s těmito daty jsem zjistil, že jednotlivé souřadnice stromů nebyly v globálním souřadnicovém systému, ale v lokálním dané skupiny. Jednotlivé skupiny totiž neměly počátek ve stejném bodě a já neznal jejich posun. Tím pádem tyto soubory nemohu využít a musím vymyslet jiný způsob.

Další soubor, který jsem od vedoucího dostal, byl model ve formátu `.obj`. Ten obsahoval všechny stromy na svých pozicích, ale před spojením do jednoho objektu. Každý strom se skládá ze dvou objektů: koruny a kmenu.

Abychom získali potřebné informace pro umístění nových stromů, tak nám stačí najít a spojit korunu s kmenem každého stromu. Následně už není problém získat posun každého stromu oproti počátku. Pro tuto akci použiji program Blender [1], který stejně jako Unreal Engine podporuje scriptování pomocí Pythonu.

3.3 Aplikace

Po té, co aplikuji kroky ze sekce 3.2 a otestuji výkon aplikace, se mohu uchýlit k tvorbě aplikace pro muzeum. Ta se dá rozdělit na dva kroky: vyřešení zobrazování na dva monitory a uživatelské rozhraní.

Dva monitory

Již v analýze v sekci 2.3 jste se mohli dozvědět, že jsem se rozhodl pro využití nDisplaye pro zobrazovaní na dvou monitorech. Hlavní nevýhodou tohoto řešení je nutnost instalace Unreal Enginu na koncovém počítači. Další nepříjemností je, že spouštění není přímočaré jako zapnutí „.exe“ souboru.

Uživatelské rozhraní

Hlavním cílem je velice uživatelsky přívětivé a intuitivní rozhraní. Při tvorbě uživatelského rozhraní se spoléhám na zpětnou vazbu od vedoucího práce a odborníků z muzea. Nemá tedy smysl na začátku vymyslet návrh a následně se ho držet.

V Unreal Enginu se k tvorbě uživatelského rozhraní využívají takzvané „widgety“. Jejich logika, tedy například akci po stisku tlačítka, se programuje pomocí blueprintů, které jsem již zmiňoval v sekci 2.2. Blueprinty také umožňují naprogramování si svého uzlu pomocí jazyka C++.

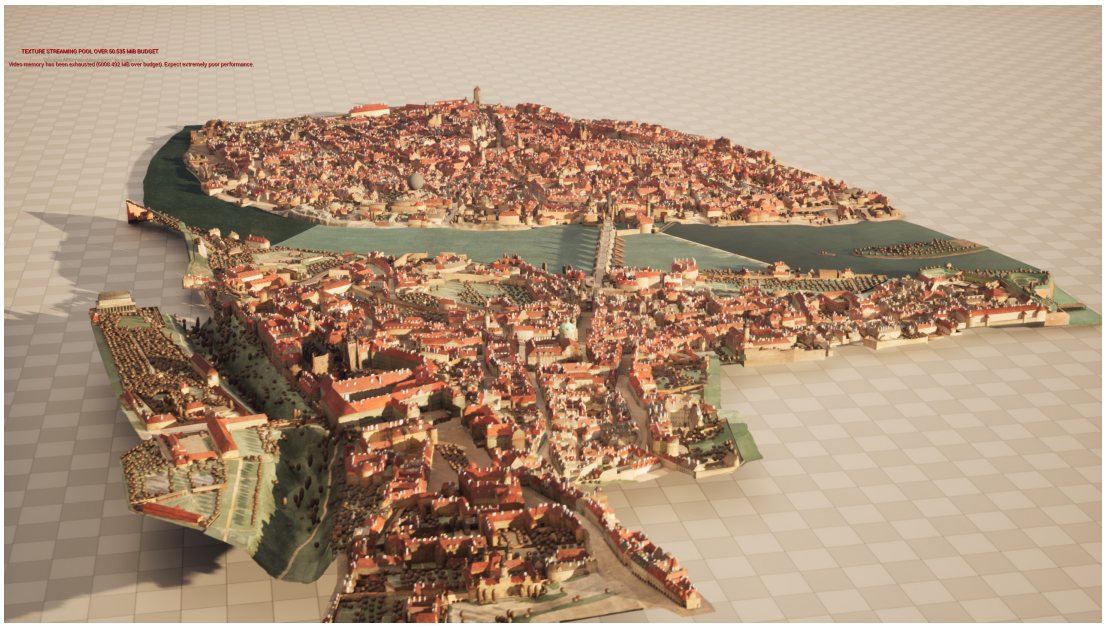
[1] `https://www.blender.org`

Kapitola 4
Implementace

Implementační část lze rozdělit na dvě části. První se věnuje optimalizaci scény s Langweilovým modelem Prahy. Většina práce je spíše o upravování parametrů uvnitř Unreal Enginu pomocí Python scriptů. U každé úpravy jsem také kontroloval její vliv na výkon aplikace. Druhá je o vytváření samotné aplikace na dva monitory s intuitivním ovládáním.

4.1 Průlety

Z důvodu testování jsem si vytvořil několik průletů kamery. Zkoušel jsem dělat takové záběry, aby obsahovaly jak detaily, tak velké množství objektů. Na obrázku 4.1 můžete vidět ukázku z poslední sekvence. Ta byla zaměřena na celý model z dálky. To totiž donutí kameru načíst téměř všechny textury a tudíž je to ideální na testování výkonu.



Obrázek 4.1. Ukázka z jednoho z průletů.

První verze aplikace obsahuje pouze tyto průlety. Slouží jako referenční k porovnávání s ostatními verzemi.

4.2 Sjednocení materiálů komínů

Pro výměnu materiálů komínů je nutné vědět, jakého typu je každý komín. To z názvu materiálu ani textur nelze zjistit, protože dodržují konvenci, kterou jsem popisoval v analýze v sekci 2.1. V projektu je k dispozici soubor `chimneyMapping.txt`. V tomto textovém souboru nalezneme mapování materiálu na typ komínu.

Na každém řádku se nachází vždy název `materiálu;typ--komínu`. Tedy například: `_21_36793c_2_x;mat_chimney_type_102a`.

Při této výměně jsem narazil na jeden problém. Někde bylo ztraceno 255 komínů. Jejich modely byly nahrazeny za čistě bílé kvádry. Tyto komíny můžete vidět na obrázku 4.2. V pravé straně obrázku se nachází normální komíny avšak uprostřed naleznete již zmíněné vadné komíny.

Obrázek 4.2. Ukázka vadných komínů.

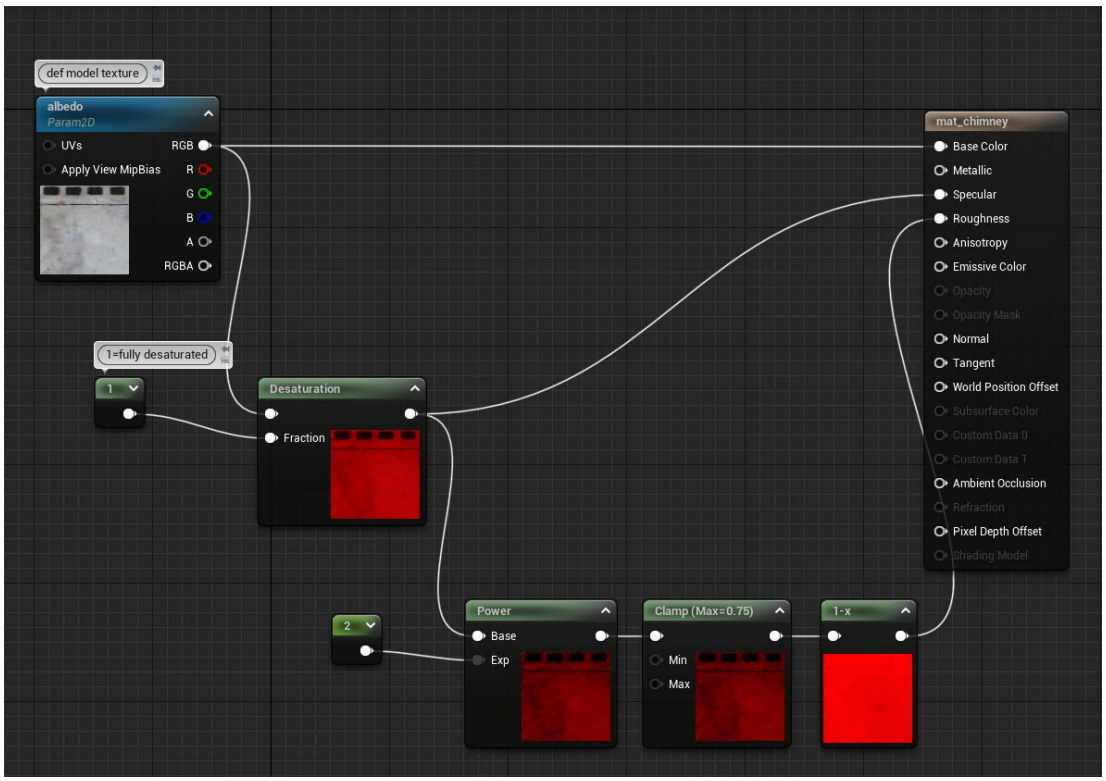
Jelikož 3 typy komínů mají tvar kvádru, tak můžeme předpokládat, že tyto komíny bez textury byly jedním z těchto typů. Po dohodě s vedoucím práce jsme se rozhodli tyto tři materiály náhodně přiřadit těmto vadným komínům. Na obrázku 4.3 je ukázka těchto tří materiálů aplikovaných na bílé kvádry. Můžete je porovnat s komínem nejvíce vpravo a vlevo, které zatím nemají aplikovaný žádný materiál.

Obrázek 4.3. Ukázka nahrazených komínů.

Těchto 255 komínů navíc chybělo i v již zmíněném souboru `chimneyMapping.txt`. Napsal jsem si tedy script v Pythonu, který prošel všechny komíny. Pokud narazil na nějaký materiál, který neexistoval v mapování načteném ze souboru, tak vybral náhodně jeden ze tří již zmíněných materiálů. Toto nové mapování se nakonec uložilo do souboru `chimneyMapping.txt` s tím, že jsem měl vytvořenou kopii originálu. Nakonec soubor obsahoval mapování pro 8036 materiálů.

Dále jsem musel vytvořit 10 materiálů. Rovnou jsem využil instancované materiály. V projektu se již nacházel předvytvořený materiál pro komíny. Tento materiál je nej-

složitější z celého projektu. Můžete ho vidět na obrázku 4.4. Ten jsem tedy využil jako rodiče pro všechny typy komínů.



Obrázek 4.4. Materiál pro komíny.

Celkem jsem tedy vytvořil 10 instancovaných materiálů pro komíny:

- mat_chimney_type_102a
- mat_chimney_type_102b
- mat_chimney_type_102c
- mat_chimney_type_102d
- mat_chimney_type_113a
- mat_chimney_type_113b
- mat_chimney_type_113c
- mat_chimney_type_119a
- mat_chimney_type_119b
- mat_chimney_type_119c

Po vytvoření kompletního mapování a materiálů, se mohlo přejít k výměně. K tomu byl využit další Python script, který prošel všechny vybrané modely. Pokud model obsahoval materiál komínu, tak ho nahradil na základě mappingu, který si načetl ze souboru. Staré materiály zůstaly zachovány. Daný script se jmenuje `chimneyMaterial.py`.

4.3 Náhrada materiálů za instancované

Dalším krokem bylo nahrazení všech zbylých materiálů za instancované. Seznam typů těchto materiálů jsem již zmiňoval v sekci 3.2. V něm naleznete navíc i `mat_chimney`, který jsem již využil při nahrazovaní materiálů u komínů.

Abych mohl začít, tak bylo potřeba vytvořit všech 10 typů materiálů. Ty jsem vytvořil jednoduše tak, že jsem si vždy našel nějakého zástupce každého typu a následně

jsem ho zkopíroval do složky „instancedMaterials“. Tyto kopie jsem přejmenoval podle typu na `mat_typ`, tedy stejně jako v seznamu v návrhu v sekci 3.2. Na rozdíl od komínů zde nepřipadalo v úvahu vytvářet instancované materiály ručně, protože jich bylo mnoho. Vytvořil jsem tedy od každého typu jednu instanci, kterou jsem pojmenoval následovně: `mat_typ_temp`. Tu jsem později pomocí duplikace využil k tvorbě všech ostatních instancí.

Tyto materiály jsou naprosto primitivní. Je to v podstatě pouze výchozí materiál, který má místo barvy nastavenou texturu.

Pro výměnu jsem zase napsal script v Pythonu. Ten procházel všechny vybrané modely a u každého iteroval přes všechny jeho materiály. Pokud se jednalo o materiál pro komín, tak ho přeskočil. U všech ostatních se na základě jména rozhodl, o jaký typ materiálu se jedná. Následně vytvořil kopii instancovaného materiálu `mat_typ_temp` odpovídajícího typu. Tu uložil na stejné místo jako se nacházel původní materiál a přejmenoval ho.

Jak již víme z analýzy Lanweilova modelu v sekci 2.1, tak název materiálu může vypadat například takto: `_19_205850_0_MG`. Nové jméno jsem tedy vytvořil tak, že jsem přidal písmeno „i“ před „MG“. Kopie zmíněného materiálu by se jmenovala tedy takto: `_19_205850_0_iMG`. Písmeno „i“ jsem zvolil jako zkratku pro slovo instancovaný.

U této kopie jsem nakonec ještě nastavil parametr „Texture“ na stejnou texturu jakou využíval původní materiál.

Kvůli rozlišení vzorů „base“ a „9“, které mají rozdílný význam pokud patří k řece, jsem vytvořil pro řeku script zvlášť. Ten byl téměř totožný s původním, ale měl upravenou detekci vzorů. Tyto dva scripty se jmenují `materials.py` a `materialsRiver.py`.

Samozřejmě se vyskytly i materiály, které byly jedinečné a nebo nedodržovaly názvovou konvenci. Bylo jich celkem 8. Musel jsem ručně projít, rozhodnout co s nimi a následně nastavit vyjímky. Čtyři jedinečné materiály jsem ponechal takové, jaké byly. Dva jsem přiřadil do střech a poslední dva odpovídaly stěnám.

Jedinečné materiály:

- _48_fontana_MG
- _71_00_base1_MG
- _71_00_base2_MG
- _95_all

Materiály se špatnou konvencí:

- _16_149243_04_MG – střecha
- _16_150881_04_MG – stěna
- _02_56180_0189_MG – střecha
- _02_56180_1182_MG – stěna

Po spuštění scriptu bohužel polovina materiálů nefungovala. Problém spočíval ve virtuálních texturách. Jak jsem již zmiňoval v návrhu v sekci 3.2, tak některé textury se nastavily jako virtuální již při importu.

Když totiž v materiálu vyměníte normální texturu za virtuální, tak se automaticky změní parametr „Sample Type“ z „Color“ na „Virtual Color“. To se ovšem u instancovaných materiálů neděje. Pokud jsem tedy vytvořil materiál `mat_typ` s normální texturou, tak všechny instance, které ji nahradily virtuální texturou, nefungovaly a naopak. Musel jsem tedy vytvořit dalších 10 materiálů. Ty byly určené pro všechny materiály s virtuální texturou. Pro rozlišení těchto materiálů jsem přidal na konec `_vt`, takže nová jména vypadají takto: `mat_typ_vt`. Jejich instance jsou pojmenovány obdobně: `mat_typ_vt_temp`.

Dále bylo potřeba u již vytvořených materiálů, které měly nastavenou virtuální texturu v parametru „Texture“, vyměnit virtuální textury za normální.

Celkem je tedy potřeba 10 materiálů pro virtuální textury:
- mat_base_vt
- mat_bridge_vt
- mat_chim_vt
- mat_nine_vt
- mat_river_vt
- mat_roof_vt
- mat_side_vt
- mat_stat_vt
- mat_wall_vt
- mat_zeme_vt

Musel jsem tedy upravit scripty `materials.py` a `materialsRiver.py`, aby rozpoznávaly virtuální textury a podle toho určovaly typ materiálu. Než jsem ale mohl tyto upravené scripty spustit, tak jsem vytvořil ještě dva scripty. `materialsRevert.py` má za úkol navrátit původní materiály k modelům. `deleteUnusedSelected.py` smaže všechny vybrané assety, které nejsou ničím využívány. Tento script je spíše upravený script od mamoniem[1].

Nakonec stačilo spustit upravené scripty a vše již fungovalo, tak jak mělo.

4.4 Virtuální textury

Pro využití virtuálních textur je potřeba, aby dané textury měly rozměry v mocninách dvou neboli anglicky „Power Of Two“(POT). Jak jsem již zmiňoval v návrhu v sekci 3.2, tak máme dvě možnosti, jak toho dosáhnout.

Porovnání metod

Prvním je doplnění například černou barvou. To lze udělat přímo v Unreal Enginu pomocí parametru „Power Of Two“ Mode. Druhý způsob spočívá v přeškálování v nějakém externím programu a následném importu nové textury. Oba způsoby mají své pro i proti.

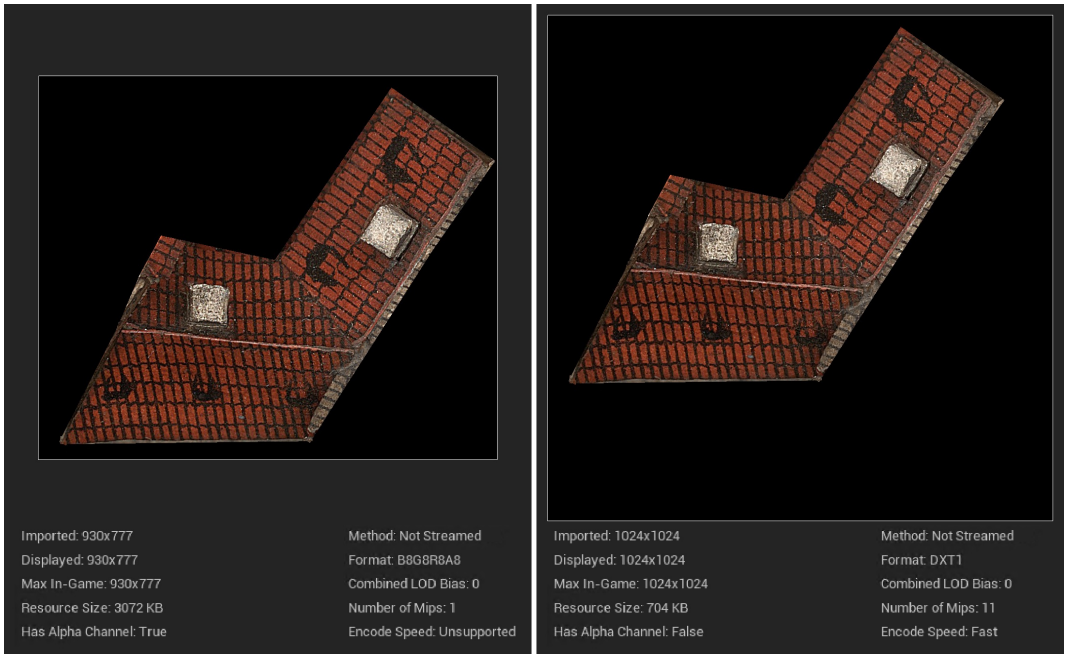
Udělal jsem porovnání stejného obrázku s rozdílným nastavením. Ve spodní části vždy naleznete informace k dané textuře.

Zajímají nás hlavně tyto parametry:
- Imported – rozměry textury
- Resource Size – velikost na disku v KB
- Method – sděluje, zda je povolen Streaming Virtuálních Textur
- Format – typ komprese obrázku
- Number of Mips – počet mipmap, které byly automaticky vytvořené

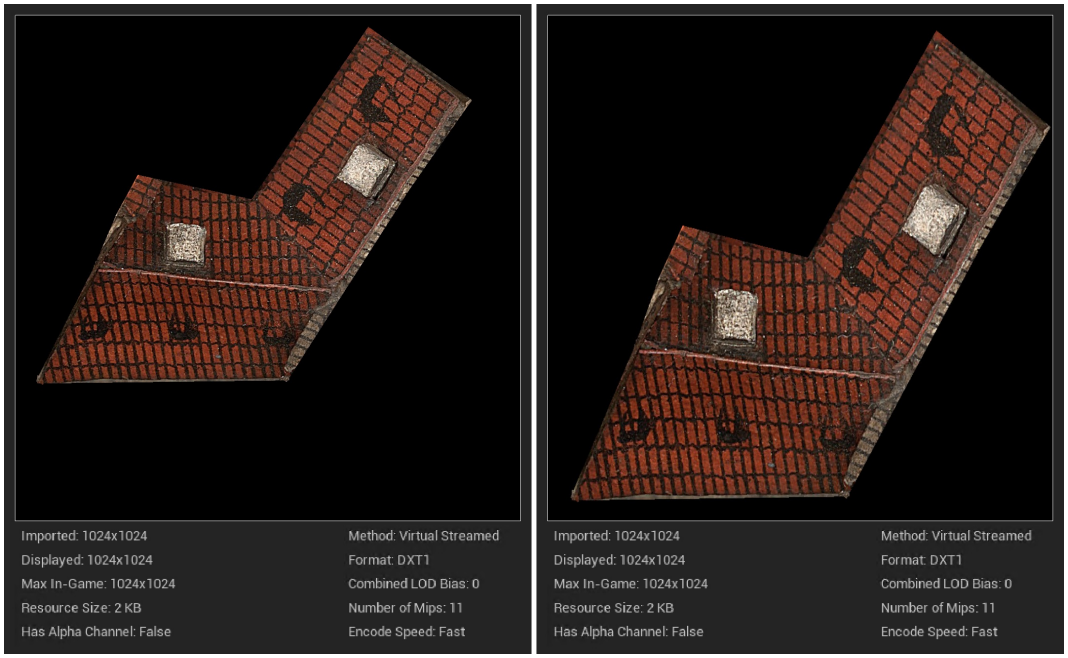
Na obrázku 4.5 vlevo je originální obrázek bez žádného speciálního nastavení. Povšimněte si, že jeho rozměry jsou 930x777px. Tedy nejsou v mocninách dvou, což je důvod, proč textura nemůže být streamována ani mít mipmapy. Také je to důvod, proč jako jediný využívá formát B8G8R8A8. Velikost tohoto obrázku je 3072 KB.

[1] `https://github.com/mamoniem/UnrealEditorPythonScripts/blob/master/Assets/DeleteUnusedAssets.py`

Na pravé straně nalezneme ten samý obrázek s tím, že má „Power Of Two“ Mode nastaven na „Pad to Power Of Two“. Ve výchozím nastavení byl doplněn o černou barvu. Nový rozměr po doplnění je tedy 1024x1024px. Díky tomu mohla být využita komprese DXT1, která vyžaduje rozměry v mocninách dvou. Dále bylo vytvořeno celkem 11 mipmap. Pozoruhodné je také to, že jsme zvětšili obrázek, vytvořili 10 mipmap navíc a velikost se i přes to změnšila na 704 KB. To je více než 4 krát menší než originální obrázek. Důvodem je již zmíněná komprese DXT1.



Obrázek 4.5. Porovnání stejného obrázku s jiným nastavením. Vlevo – originální obrázek, vpravo – POT Mode nastaven na Pad to POT.



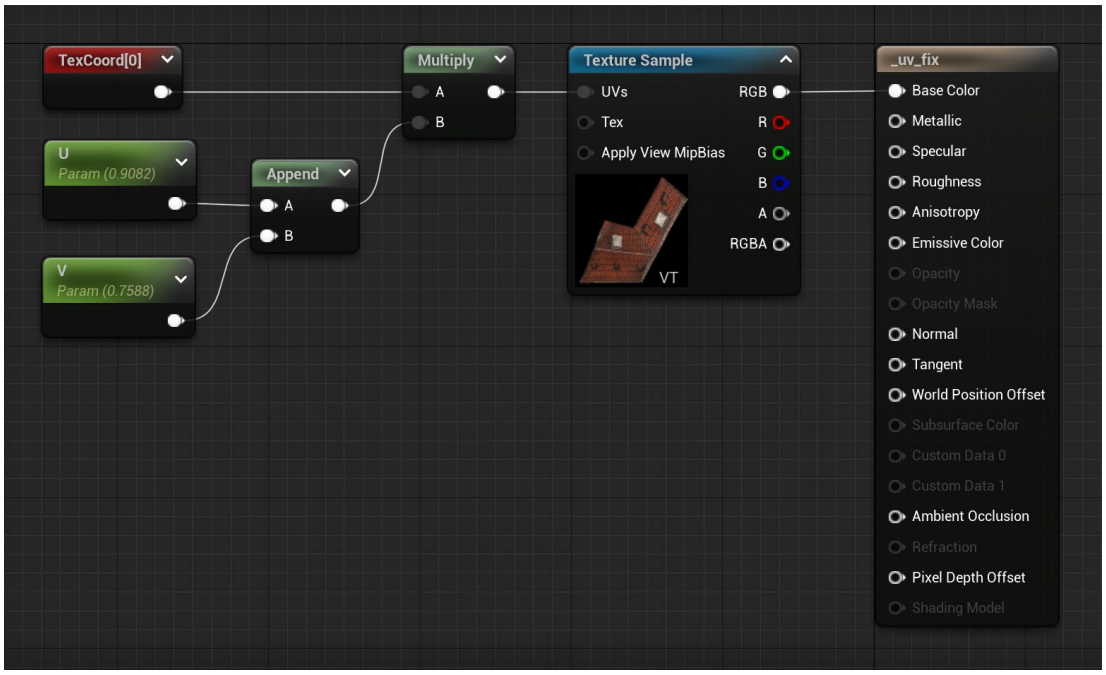
Obrázek 4.6. Porovnání stejného obrázku s jiným nastavením. Vlevo – POT Mode nastaven na Pad to POT a povolen Streaming Virtuálních Textur, vpravo – Obrázek externě škálovaný na mocniny dvou a povolen Streaming Virtuálních Textur.

Na levé straně obrázku 4.6 se nachází téměř totožný obrázek jako na pravé straně předchozího. Jediný rozdíl je, že zde je povolen Streaming Virtuálních textur. To zapříčinilo další snížení velikosti textury na 2 KB.

Na pravé straně se vyskytuje obrázek přeškálovaný v externím programu. Také má aktivní Streaming Virtuálních textur. Jak lze vidět, tak jeho vlastnosti jsou totožné s texturou na straně levé. Jeho hlavní výhodou je to, že není potřeba upravovat UV souřadnice.

Doplnění na mocniny dvou

Pokud upravíme texturu doplněním na mocniny dvou, tak nastane problém s UV souřadnicemi. Ten lze vyřešit pomocí nového materiálu 4.7. K jeho vytvoření jsou potřeba rozměry původní a nové textury. Poměr jejich šířek, který získáme jako *stará*/*nová*, uložíme do parametru „U“. Obdobně vypočteme i parametr „V“ jako poměr výšek.



Obrázek 4.7. Materiál, který opraví UV souřadnice.

Parametry „U“ a „V“ se následně spojí do 2D vektoru, kterým se vynásobý UV souřadnice aktuálně vykreslovaného pixelu. To zapříčiní přeškálování UV souřadnic zpět na původní velikost. V bloku „Texture Sample“ se nachází náhled textury. V ní lze vidět, že neobsahuje černé okraje, protože byly oříznuty úpravou UV souřadnic.

Tento způsob je sice funkční, ale při renderování každého pixelu daného materiálu by se neustále muselo vypočítávat škálování UV souřadnic. To by mělo velký dopad na dobu renderování. Sice je to jednodušší způsob než škálování v externím programu, ale je nevhodný pokud se pokoušíme optimalizovat scénu.

Škálování textury

Jak jsem již zmiňoval v návrhu v sekci 3.2, tak kvůli velkému množství textur, je nelze zvětšovat ručně. Rozhodl jsem se tedy využít zase Python. Jelikož v tuto dobu se všude řešil ChatGPT od firmy OpenAI[1], tak jsem toužil po vyzkoušení jeho možností. Tento script byl vhodný pro zkoušku jeho schopností.

[1] `https://openai.com`

Z počátku jsem chtěl textury vyexportovat z Unreal Enginu do formátu TGA. Požádal jsem tedy ChatGPT o script, který načte TGA obrázek a přeškáluje ho na mocniny dvou. On se rozhodl využít knihovnu PIL[1]. První verze nefungovala, protože načítal obrázek v módu „RGBA“, ale on byl pouze „RGB“. Požádal jsem ho, ať to upraví a kód již fungoval.

Výsledný obrázek byl ale modrý, i když původní byl červený. Zeptal jsem se ho tedy v čem je problém a ať ho opraví. Vysvětlil mi, že TGA soubor ukládá pixely v pořadí „BGR“ a tudíž musím tyto kanály prohodit. Rovnou mi i poslal upravený script.

Nakonec jsem si ještě všiml, že obrázek byl zezrcadlený podle osy Y. Požádal jsem ho tedy o úpravu, kterou udělal. Při ní bohužel rozbil prohození pořadí kanálů. To jsem tedy překopíroval z předchozí verze a vše fungovalo jak má.

Kód každopádně hezky rozděloval do funkcí a občas i psal komentáře. Celkově hodnotím spolupráci s ChatGPT za velice kladnou. Naštěstí není dokonalý a navíc člověk musí rozumět tomu, co po něm chce. Z toho důvodu se neobávám, že by programátoři byli nahrazeni. Spíše jim umělá inteligence jenom ulehčí práci, ale nanahradí je.

Nakonec jsem našel originální textury ve formátech PNG a JPG. Upravil jsem tedy script na tyto formáty sám. Dále jsem také připsal průchod všech složek s originálními soubory. Takto jsem přeškáloval všechny obrázky, které měly alespoň jeden rozměr větší než 512px. Nový soubor jsem pojmenoval stejně jako originál s tím, že jsem na konec přidal `_s`. Tedy přeškálovaný obrázek `_01_661_0_tx.png` se jmenuje `_01_661_0_tx_s.png`. Tento script jsem pojmenoval `script_pil.py`.

Výměna textur

Tyto soubory jsem naimportoval do projektu do složky `scaled_textures`. Všechny tyto textury jsem následně převedl na virtuální. Nakonec jsem pak vytvořil scripty `materialsScaledTextures.py` a `materialsRiverScaledTextures.py`, které projdou všechny vybrané modely. V nich pak kontrolují, zda existuje daná textura i ve složce `scaled_textures`. Pokud ano, tak ji tomuto materiálu přiřadí. Řeka má oddělený script ze stejného důvodu, jako při instancování materiálů.

Po té, co jsem spustil tento script, jsem viděl ve scéně rozmazané textury. Zjistil jsem, že když jsou virtuální textury povoleny u velkého množství obrázků, tak se jim začne zhoršovat kvalita. Nastávaly dvě chyby. Tyto chyby lze vidět na obrázku 4.8. Vlevo se nachází originální obrázek, uprostřed lze vidět jemné rozmazání a vpravo úplné rozmazání do jednolité barvy. Pokud jsem u některého obrázku vypl a znovu zapl streamování virtuálních textur, tak vždy přešla do úplného rozmazání. I přesto jsem zkusil aplikaci sestavit, ale výsledek vypadal stejně.

Obrázek 4.8. Porovnání obrázku s jeho chybnými stavy.

[1] `https://pillow.readthedocs.io/en/stable/`

Udělal jsem tedy kopii tohoto scriptu a pojmenoval ji `materialsScaledTexturesBig.py`. Zde jsem přidal podmínku na velikost obrázku. Nejdříve jsem zkusil použít pouze textury, s velikostí obou rozměrů alespoň 4096px.

Dále jsem tuto hranici zmenšoval. Nejdříve na 2048px a následně na 1024px. U verze s hranicí 1024px se textury zase rozbily, tak jsem ponechal tu s virtuálními texturami, které měly oba rozměry alespoň 2048px.

4.5 Úroveň detailů

Měnit úroveň detailů lze jak u modelu, tak i u textur. Jak jsem již zmiňoval v analýze v sekci 2.1, tak Langweilův model nemá moc detailní modely. Jde o to, že většina zdí jsou prostě rovné plochy, které jsou reprezenovány dvěmi trojúhelníky. To znamená, že model moc zjednodušovat nelze. Proto jsem se rozhodl nejdříve experimentovat s texturami.

Mipmapy textur

Abychom mohli využít Unreal Engine k automatickému vytvoření mipmap, tak je potřeba, aby všechny textury měly rozměry v mocninách dvou. Jelikož jsem si toto neuvědomil dříve, tak jsem nepřeškáloval všechny textury rovnou. Upravil jsem tedy script na škálování a pojmenoval ho `script_pil_small.py`. Ten zvětšil všechny menší obrázky než ten původní. Ty jsem naimportoval do složky `scaled_textures_small`.

Dalším krokem bylo vytvoření scriptu `materialsScaledTexturesSmall.py`. Jeho úkolem bylo projít všechny materiály vybraných modelů a pokud obsahoval texturu, která neměla rozměry v mocninách dvou, tak ji nahradil. Nové textury vybíral buď ve složce `scaled_textures` nebo `scaled_textures_small`. V původní složce se totiž stále nacházely textury, které nebyly využity jako virtuální.

Unreal Engine už automaticky vytvořil mipmapy při importu těchto obrázků, takže není potřeba nic jiného dělat.

Nanite

Pro automatickou tvorbu LOD modelů lze využít Nanite. O tom, k čemu slouží, jsem již psal v analýze v sekci 2.3.

Když jsem začal experimentovat s Nanitem, tak jsem zjistil, že některé modely ho již měly aktivní. Nebylo jich mnoho, ale překvapilo mě to. Nanite je totiž nutno aktivovat. Při importu ve výchozím stavu zůstane vypnutý.

Jak jsem již v analýze v sekci 2.3 zmiňoval, tak jsem nepředpokládal, že Nanite bude fungovat dobře. Proto jsem script, který jsem pojmenoval `naniteChange.py`, napsal s myšlenkou na následnou deaktivaci. Na začátku scriptu tedy najdeme proměnnou `USE_NANITES`, která je typu „bool“. Tedy nabývá hodnot pravda/nepravda. Změnou hodnoty této proměnné tedy změníme účel scriptu. Ten po spuštění projde všechny vybrané modely a upraví jejich nastavení Nanitu podle proměnné `USE_NANITES`.

Nejdříve jsem tedy aktivoval Nanite u všech modelů. Vše fungovalo jak má. Očekával jsem, že budou vidět přechody mezi jednotlivými úrovněmi detailů. Nejvíce jsem čekal problémy s věžičkami u kostelů. Nic takového se však nedělo. Vše vypadalo dobře, dokud jsem nenaměřil sestavenou verzi. Jak se ale dozvíte v kapitole testování 5, tak nastal veliký úbytek FPS[1].

[1] frames per second - snímků za sekundu

Z tohoto důvodu jsem tedy využil script se změněným nastavením proměnné `USE_NANITES` na nepravdu. Výsledkem tedy je vypnutý Nanite i u modelů, kde byl Nanite aktivní, když jsem dostal projekt do rukou.

Z mě neznámého důvodu se razantně snížila načítací doba aplikace. Ještě zvláštnější je, že se snížila v obou verzích. Tedy jak ve verzi s aktivním Nanitem u všeho, tak i u verze s deaktivovaným Nanitem.

4.6 Výměna stromů

Po dokončení veškeré optimalizace přišel čas na výměnu stromů. Jak jsem již zmiňoval v návrhu, tak je nutné získat souřadnice a velikost jednotlivých stromů. Následně je potřeba buď najít na internetu a nebo vytvořit nějaký strom, který by odpovídal reálnému modelu.

Získání souřadnic a výšky

K získání souřadnic a výšky využiji model `stromy_separate.obj` a program Blender. Jelikož Blender umožňuje také scriptování pomocí Pythonu, tak jsem zkusil vše udělat přímo v něm. Musel jsem se tedy naučit pracovat s Blender Python API[1]. S tím jsem neměl žádné problémy, protože je velice dobře zdokumentované. Navíc je Blender hojně používaný, takže se na internetu dala dohledat spousta dotazů na již zmíněnou API i s odpověďmi.

Při prozkoumávání modelu jsem zjistil, že všechny objekty, až na jeden, dodržují názvovou konvenci `stromy.ID_Material__28.ID`. Navíc každé liché ID je koruna stromu a jí odpovídající kmen je předchozí sudé ID. Problematická je tedy koruna s ID 001, protože neexistuje žádný kmen s ID 000. Zde přichází na řadu jediný objekt, který nedodržuje konvenci. Ten se jmenuje pouze `stromy_Material__28`. Tento objekt je kmen, který odpovídá koruně 001. Proto jsem daný strom přejměnoval na `stromy.000_Material__28.000`.

Napsal jsem script, který si uloží všechny vybrané objekty do pole. To se následně seřadí s využitím mnou vytvořené funkce pro získání klíče z názvu objektu. Ta pouze získala ID za tečkou a převedla ho na číslo. Takto seřazené pole následně prošel jako frontu a vždy dvě po sobě jdoucí položky spojil v jednu.

V tuto chvíli byly jednotlivé stromy spojené v jeden objekt, ale všechny měly stále počátek ve středu globálního souřadnicového systému. Další script tedy sloužil k převedení počátku do středu takzvaného „bounding boxu“. To je kvádr, který přesně opisuje daný objekt a jeho hrany jsou rovnoběžné s osami.

Poslední script vytvořil pole, do kterého přidával objekty. Ty obsahovaly naše informace, tedy „location“ a „scale“. Po troše experimentování jsem zjistil, že je potřeba pozici vynásobit 1000 a velikost 10. Pozici jsem ukládal klasicky jako `x y z`, ale pro velikost stačila pouze osa `Y`. Toto pole jsem následně uložil ve formátu json[2].

Model stromu a náhrada

Ačkoli jsem prohledal několik stránek, tak se mi nepodařilo najít žádný strom, který by odpovídal tomu, co jsem potřeboval. Nakonec jsem se rozhodl zkusit jehličnan ze stránky Free3D[3]. Neměl jsem v plánu ho v projektu použít, ale spíše vyzkoušet, zda

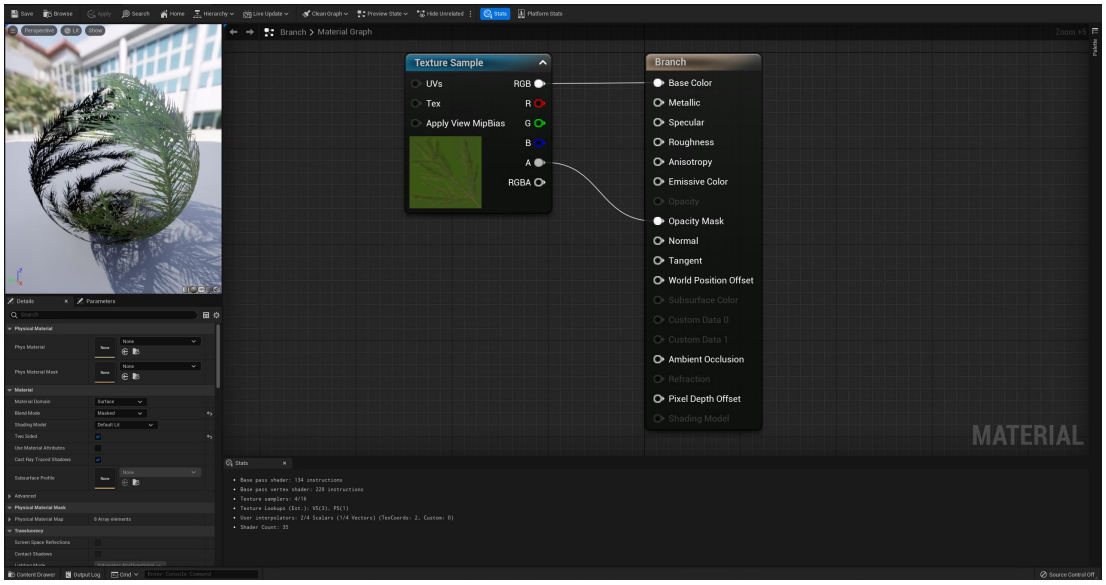
[1] `https://docs.blender.org/api/current/index.html`
[2] `https://www.json.org/json-en.html`
[3] `https://free3d.com/3d-model/fir-low-poly-11569.html`

tento typ modelu je vhodný po výkonostní stránce. Tento strom můžete vidět na obrázku 4.9.

Obrázek 4.9. Stažený model stromu ze stránky Free3D.

U tohoto modelu jsem poprvé řešil průhlednost materiálu. Nejprve jsem musel vytvořit blok „Texture Sample“, kam jsem zvolil texturu větvičky, kterou jsem stáhl s modelem. Ten jsem následně připojil na parametr „Base Color“. Dále jsem musel vybrat blok výsledného materiálu. V jeho nastavení vlevo bylo potřeba změnit parametr „Blend Mode“ z „Opaque“ na „Masked“. Po tomto kroku se v bloku materiálu odemkl parametr „Opacity Mask“. Do té stačilo připojit alfa kanál z bloku „Texture Sample“.

Jelikož byl výsledný strom řídký, tak jsem si uvědomil, že bych mohl povolit renderování z obou stran. Toho jsem docílil tak, že jsem v nastavení bloku materiálu zaškrtnul políčko „Two Sided“. Výsledný materiál můžete vidět na obrázku 4.10.



Obrázek 4.10. Materiál jehličí pro stažený model stromu.

Když už jsem měl model, tak jsem mohl začít rozmisťovat. Napsal jsem tedy script `trees.py`. Ten načetl mnou vytvořený json soubor a podle něj následně rozmístil modely stromů do scény. Aby ve scéně nebyly přímo v kořenu hierarchie, tak jsem využil funkci `set_folder_path(folder_path)`. Jako parametr jsem zvolil cestu `'/model/stromy/'`.

Na obrázcích 4.11 a 4.12 můžete vidět stejný pohled kamery před a po výměně. Důvod, proč některé stromy jsou zapuštěné do země je ten, že i v původním modelu byly tyto stromy zapuštěné. Jelikož u nového modelu začíná koruna mnohem dříve, tak je to více očividné. Původně totiž byl zapuštěn jenom kmen, což nepůsobilo divně.

Obrázek 4.11. Originální rozmístění stromů.

Obrázek 4.12. Rozmístěné testovací stromy.

Když jsem se druhý den pokoušel pokračovat, tak nastal problém. Projekt nešel zapnout. Zkoušel jsem ho spustit několikrát, ale vždy hned na začátku načítání spadl. Chybová hláška značila, že prý dochází ke kruhové závislosti assetů, což ale nebyl můj případ.

Po dlouhém testování jsem přišel na to, že pokud v souboru `DefaultEngine.ini` změním parametr `GameDefaultMap` na jinou scénu, tak se projekt bez problémů načte.

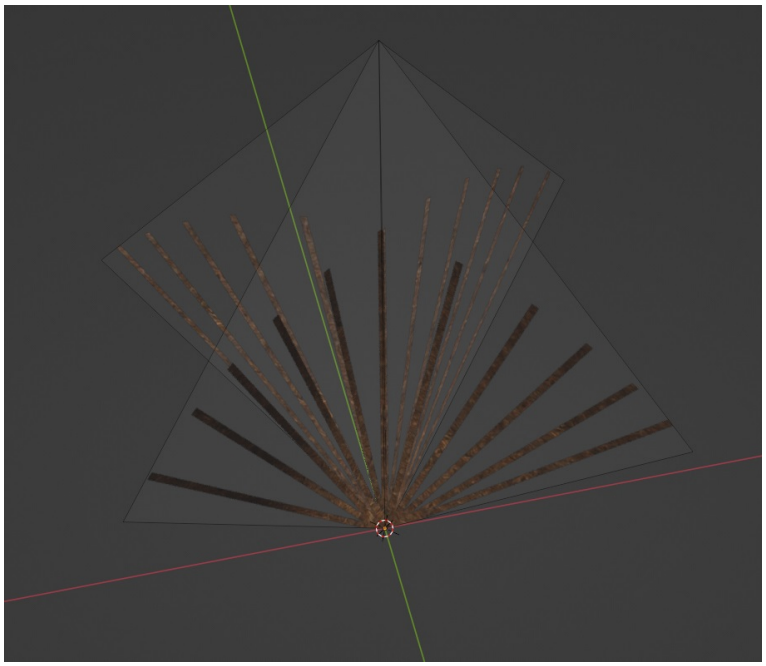
Avšak při pokusu o načtení hlavní scény znovu engine spadne. Sice jsem nevěděl co chybu způsobuje, ale věděl jsem, že je problém v hlavní scéně.

Naštěstí jsem měl spoustu záloh. Načetl jsem tedy nejnovější zálohu a mohl jsem pokračovat.

Tvorba nového modelu stromu

K tvorbě nového modelu jsem využil stejný princip, který používal stažený model. Ten měl jednu texturu větviček. Jeden shluk jehličí byl tvořen čtvercem, který byl ohnutý v úhlopříčce. Ten využíval zmíněnou texturu. Daný shluk byl následně rozkopírován několikrát po celém stromu tak, že jeden roh byl vždy přidělaný ke stromu. Každý měl různou velikost. Dolní byly mnohem větší než horní.

Já jsem se tedy inspiroval tak, že jsem si také udělal shluk, ale ten byl ze dvou čtverců, které tvořily kříž. Můžete ho vidět na obrázku 4.13. Pro texturu kmene jsem využil stejnou texturu, jakou využívaly elipsoidové stromy, ale přeškáloval jsem ji, aby měla rozměry v mocninách dvou.

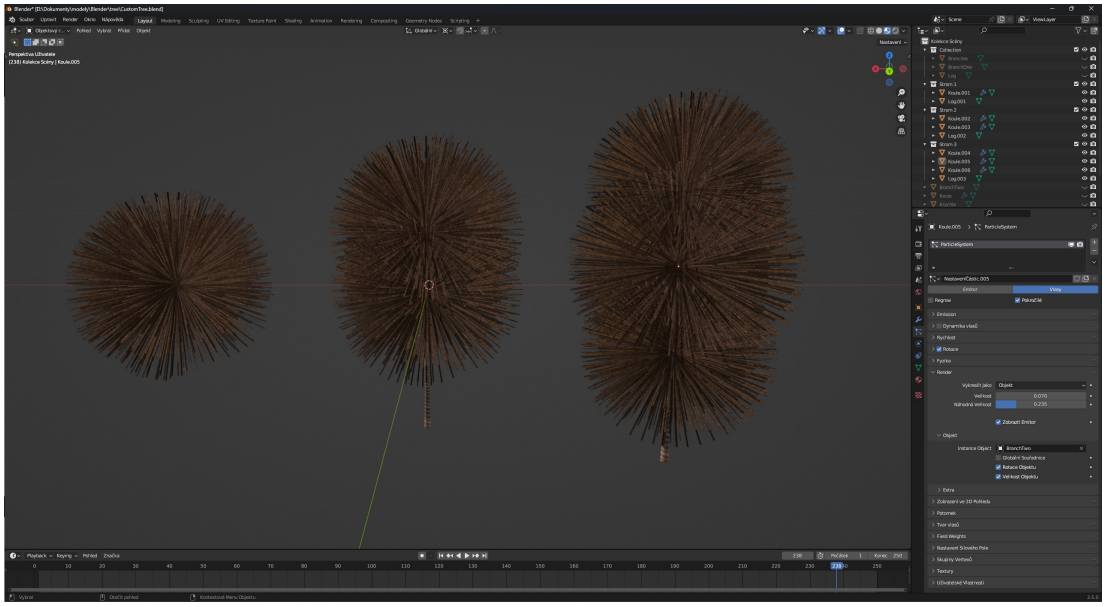


Obrázek 4.13. Shluk větviček pro nový model stromu.

Dále jsem vytvořil kouli, kterou jsem přeškáloval na 0.00005 ve všech osách. Na tuto kouli jsem pomocí systému částic nanesl křížový shluk štětinek. Zvolil jsem jako typ systému vlasy. Celkem jsem na jednu kouli dal 50 částic. Pak jsem upravoval parametry velikosti, rotace a náhodnosti dokud se mi výsledek nezamlouval.

Celkem jsem udělal 3 velikosti stromů. To umožní větší variaci stromů ve scéně. Díky tomu jsem mohl udělat více specifické proporce pro vysoké stromy a nízké keře. Tyto 3 velikosti můžete vidět na obrázku 4.14.

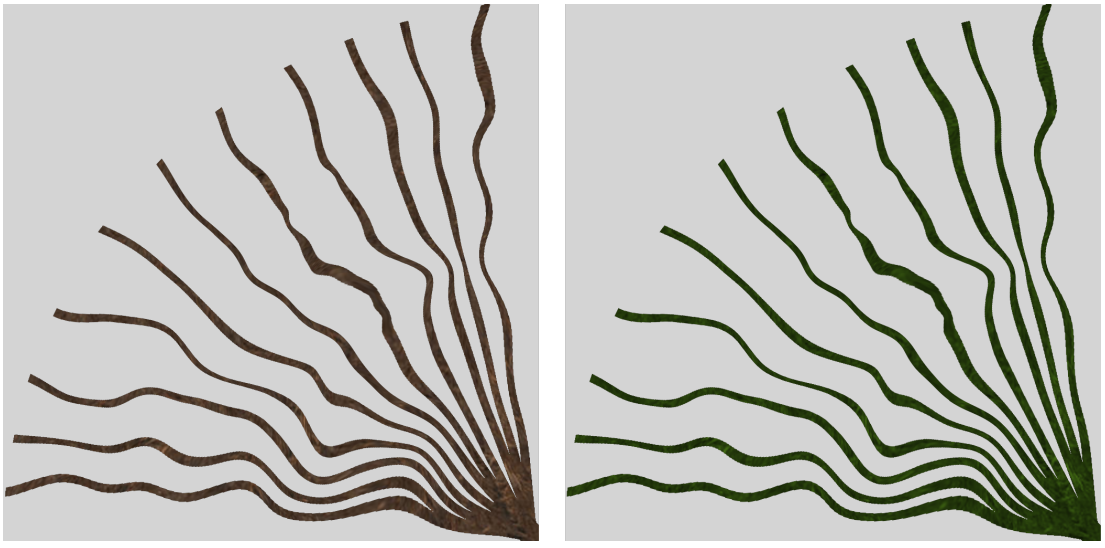
Prostřední velikost obsahuje dvě částicové koule, které jsou nastaveny stejně jako ta původní na nejmenším stromu. Poslední strom má celkem 3 částicové shluky. Pouze prostřední má upravené nastavení. Jak můžete vidět, tak ta má o trochu delší štětiny. Později uvidíte, že jsem od tohoto rozhodnutí ustoupil.



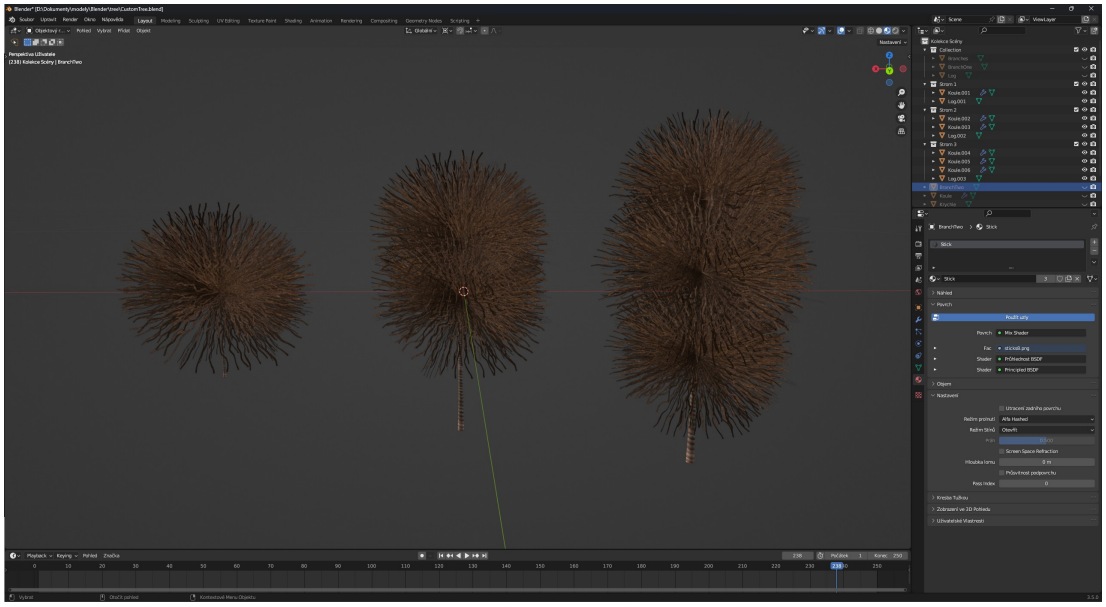
Obrázek 4.14. Ukázka rovné verze stromů.

Jelikož stromy působily celkem jednolitě, tak jsem upravil texturu na vlnitou. Novou texturu můžete vidět na obrázku 4.15 vlevo. Pozadí je šedivé pouze zde, aby obrázek vynikl. Textura má pozadí plně průhledné. Výsledek stromů s novou texturou se nachází na obrázku 4.16. Řekl bych, že to byl krok správným směrem. Přestože rozložení trojúhelníků je stejné, tak stromy nyní působí více přirozeně a hustší.

Aktuální textura byla vytvořena jako výřez z textury pro elipsoidové stromy. Přestože byla použita v modelu, tak reálný model má spíše zelené štětiny. Z toho důvodu jsem upravil odstín mé textury do zelena. Finální texturu tedy můžete vidět na obrázku 4.15 vpravo.



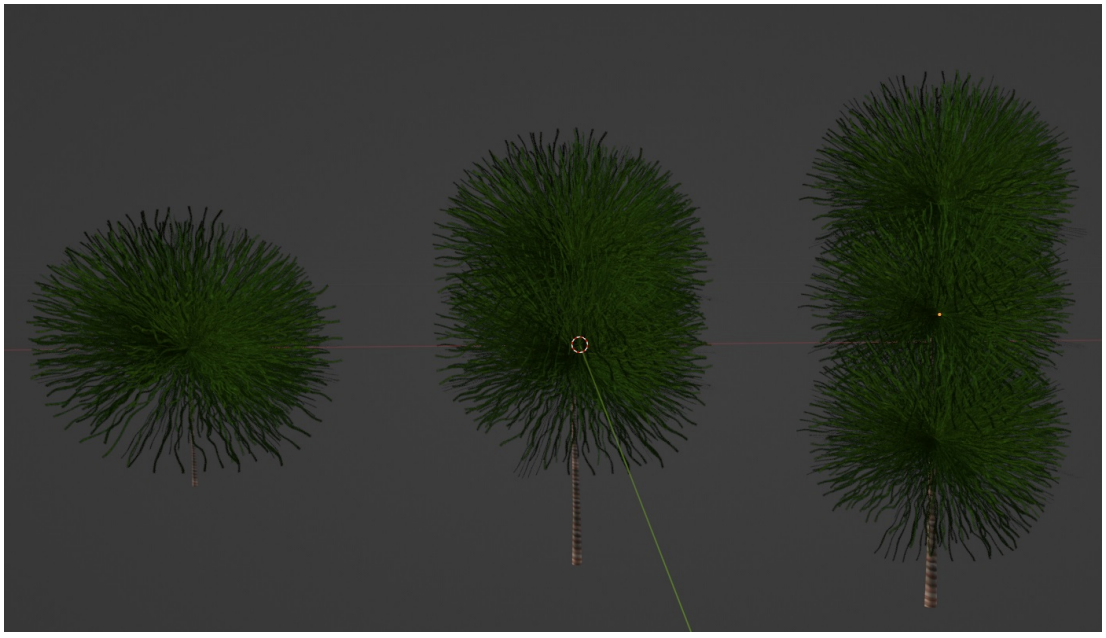
Obrázek 4.15. Porovnání textur štětinek. Vlevo - vlnitá, vpravo - finální.



Obrázek 4.16. Ukázka vlnité verze stromů.

Při aplikování finální textury jsem ještě upravil nastavení prostředního shluku u největšího stromu, aby měl stejně velké částice jako ostatní. Na obrázku 4.17 se nachází první verze mých stromů.

V tuto chvíli jsem v záložce „Vlastnosti modifikátoru“ použil tlačítko „Vytvořit Instance Reálně“. To udělalo z jednotlivých částic normální trojúhelníkové objekty. Ty jsem následně spojil do jednoho objektu i s kmenem. Jejich velikost jsem nastavil tak, aby všechny stromy byly vysoké 1m. To je z důvodu, aby se následně dala jednoduše aplikovat velikost stromů z json souboru co jsem vytvořil v jednom z předchozích kroků. Změnil jsem počátek na střed „bounding boxu“ a všechny je přesunul do počátku. Výsledný model tedy obsahuje 3 objekty. To jsem vyexportoval v jednom „.fbx“ souboru.



Obrázek 4.17. Ukázka zelené verze stromů.

Při importu do Unreal Enginu stačí nemít zaškrtnuté políčko „Combine Meshes“ a každý strom se načte zvlášť. Materiál pro štětinky jsem nastavil stejně jako na obrázku 4.10, který jsem vytvořil u staženého stromu.

Když už jsem měl hotové modely, tak stačilo upravit script na rozmisťování stromů tak, aby podle velikosti zvolil odpovídající model. Teď stačilo jenom zjistit hraniční velikosti pro změnu modelu.

Po zkoušení počtu stromů v každé skupině jsem se rozhodl využít tyto hodnoty:

- malé – < 0.13
- střední – $>= 0.13$ a zároveň $<= 0.25$
- velké – > 0.25

Při tomto rozložení je nejvíce středně velkých stromů. Připadalo mi to nejvhodnější. Po rozmístění stromů jsem byl na první pohled spokojen s výsledkem. Na druhý pohled jsem ale zjistil, že čím je strom dál, tím je více vidět rozdělení do koulí. Tento efekt můžete vidět na obrázku 4.18.

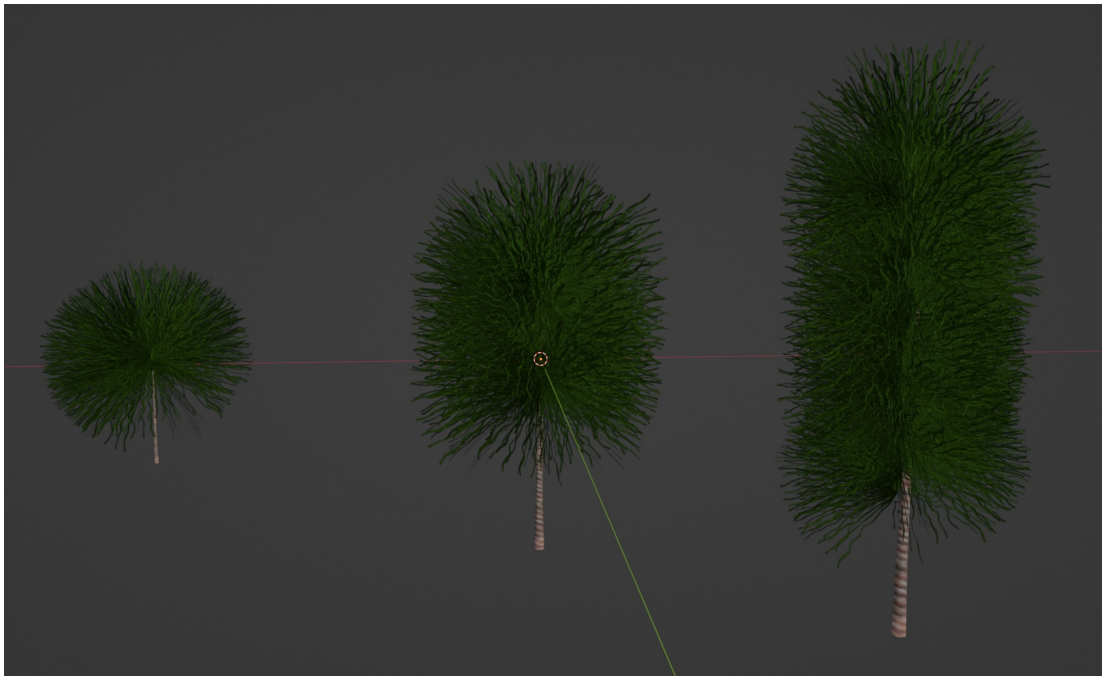
Z tohoto důvodu jsem musel předělat model stromu ještě jednou. Nejmenší strom nebylo potřeba měnit. Nejdříve jsem experimentoval s nahrazením za polokouli, kterou jsem natáhl na poloviční elipsoid. Samozřejmě zase přeškálovanou na minimum v osách X a Y. Zde ale blbnul vršek, takže by potřeboval mít nahoře zase kouli. Nakonec jsem ale použil válec, který fungoval skvěle i sám o sobě. Počet částic je nyní vždy dvojnásobný oproti předchozí verzi. Nejmenší má tedy 50, prostřední 100 a největší 200 částic. S přibývající velikostí stromu jsem také zvětšoval velikost štětinek.

Obrázek 4.18. Rozmístěné stromy první verze.

Tato verze stromů byla již finální. Můžete ji vidět na obrázku 4.19. Provedl jsem tedy stejné úkony, abych je vyexportoval jako v předchozí verzi.

Když jsem je chtěl vložit do projektu, tak ten znovu při zapínání spadl. Nahrál jsem tedy znovu zálohu, znovu vytvořil průhledný materiál pro štětinky a nahrál finální verzi stromů. Následně jsem hledal chybu.

Zkusil jsem rozmístit 1 strom pomocí mého scriptu a následně restartovat Unreal Engine. Scéna fungovala, ale v hierarchii scény nyní kromě složky `model` existovala i složka `model1`. Složka `model` obsahovala stejné podsložky jako před tím, ale všechny až na složku `/model/stromy/` byly všechny prázdné. Původní objekty z této složky se nyní nacházely ve složce `model1`.



Obrázek 4.19. Ukázka finální verze stromů.

Když jsem se pokusil složku `model` odstranit, tak se zase celá scéna rozbila. Po nahrání zálohy jsem zkusil ještě rozmístit 5 stromů pomocí mého scriptu. Nyní se po restartování objevily složky: `model1`, `model2`, `model3`, `model4` a `model5`.

Rovnou jsem nahrál zálohu. Problém byl ve funkci `set_folder_path(folder_path)`. Proto jsem ji zakomentoval. Všechny stromy se tedy vložily do kořene hierarchie. Mě pak stačilo je všechny označit a přesunout do složky `/model/stromy/`. Když jsem to udělal takto, tak vše fungovalo i po restartování Unreal Enginu. Jak můžete vidět na obrázku 4.20, tak stromy v dálce již nejsou dělené. Myslím si, že chyba v dané funkci je v tom, že pokud daná složka existuje, tak místo aby položku do ní přidala, tak vytvoří její kopii. Ve chvíli, kdy jsem přidal tisíce stromů na ráz, tak to vytvořilo spoustu složek a Unreal Engine to nezvládl.

Obrázek 4.20. Finální rozmístění stromů.

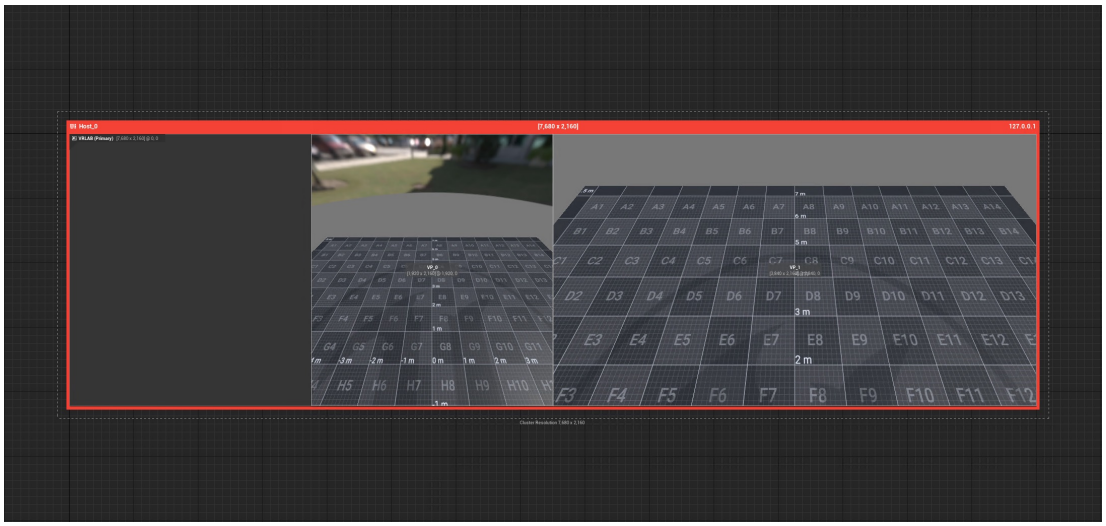
4.7 Dva monitory

Pro zobrazení na dva monitory jsem využil nDisplay, který je součástí Unreal Enginu. Aby mohl být využit, tak je nutné ho nainstalovat do projektu přes správce doplňků. Jak jsem již zmiňoval v sekci 2.3, tak je potřeba vytvořit nDisplay config. Ten se chová velice podobně jako blueprint. Má tedy vlastní hierarchii do které lze přidat jakýkoliv komponent. Já jsem do ní dal dvě kamery a dva komponenty typu: „Scene Capture Component 2D“. Funkce těchto komponentů je velice podobná. Kamera zachycuje scénu a výsledný obraz posílá do viewportu. „Scene Capture Component 2D“ také zachycuje scénu, avšak výstup ukládá do textury. To lze využít například na zobrazování dané textury v uživatelském rozhraní nebo na uložení vyrenderovaného obrázku.

Na rozdíl od klasického blueprintu nDisplay config obsahuje ještě nastavení samotného zobrazení na monitory. Aktuální nastavení můžete vidět na obrázku 4.21. Hlavní červené ohraničení značí uzel, který odpovídá jednomu stroji. Uzlů může být několik. K nim se následně lze připojit a na dálku využívat jeho monitory k zobrazování. V mém případě ale stačí pouze jeden, protože nám stačí jeden počítač. Uzel musí mít nastaveno rozlišení a umožňuje také nastavit posun okna. To se hodí v případě, že máte v systému Windows jako hlavní monitor nastaven ten vpravo. V tu chvíli lze okno posunout o velikost levého monitoru doleva a aplikace se tedy zobrazí správně.

V obrázku také můžete vidět, že uzel je rozdělen na tři části. První část zleva je prázdná. Toto je místo, kde se bude zobrazovat mapa, takže zde není potřeba nic renderovat. Druhá část je viewport s náhledem. V pravé polovině se nachází druhý viewport, který je určen pro zobrazování na televizi. O přesném účelu těchto částí se dozvíte v sekci 4.8.

Dále v něm nalezneme také jeden vizuální script. Jeho účelem je nastavit transformaci kamery pro televizi na stejnou jako má kamera náhledu s tím, že se přesune postupně s určitou rychlostí, kterou lze nastavit proměnnou „Speed“. Je to z důvodu, že člověk, který aplikaci ovládá chce mít rychlou odezvu, avšak lidi co se pouze dívají na televizi by ztratili přehled, kdyby se jim kamera neustále teleportovala. Tímto řešením docílíme rychlé odezvy i toho, že ostatní návštěvníci neztratí orientaci. Tato funkce je zakázaná, pokud jsou kamery ovládány předpřipravenou sekvecí průletu.



Obrázek 4.21. Rozložení nDisplaye.

Obou viewportům odpovídá jedna kamera z hierarchie. Každá kamera je spojena s jedním „Scene Capture Componentem 2D“. To mi dává možnost testovat aplikaci

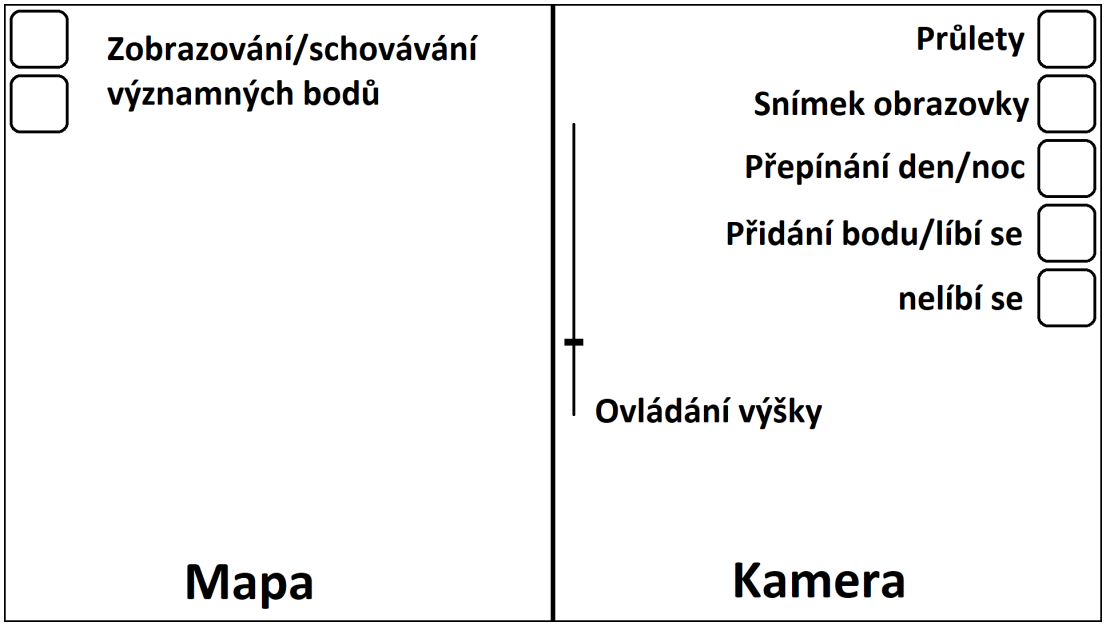
pomocí uživatelského rozhraní, když nemám k dispozici dva monitory správných rozměrů. Rozměry výsledné aplikace jsou totiž nutné zadat již při vytváření tohoto configu. Aplikaci tedy nelze spustit na monitorech s nižším rozlišením. To je jedním z problémů nDisplaye.

4.8 Grafické rozhraní

Požadavky na aplikaci byly tyto:

- Aplikace pro prohlížení modelu na dvou obrazovkách, kde jedna je dotyková a je určena k ovládání. Druhá je pouze televize k prohlížení pro ostatní návštěvníky, kteří neovládají aplikaci.
- Pohyb po modelu
- Zapnutí předpřipravených průletů
- Znázornění významných míst
- Přepnutí do pohledu na významná místa
- Vytvoření snímku modelu a následné zaslání na email
- Změna osvětlení
- Zobrazení mapy
- Možnost skrytí/zobrazení významných míst
- Znázornění významných míst na mapě i ve 3D pohledu

První prezentovaná verze grafického rozhraní měla podobu, kterou můžete vidět na obrázku 4.22. Tu si vyzkoušela odbornice z muzea PhDr. Kateřina Bečková.



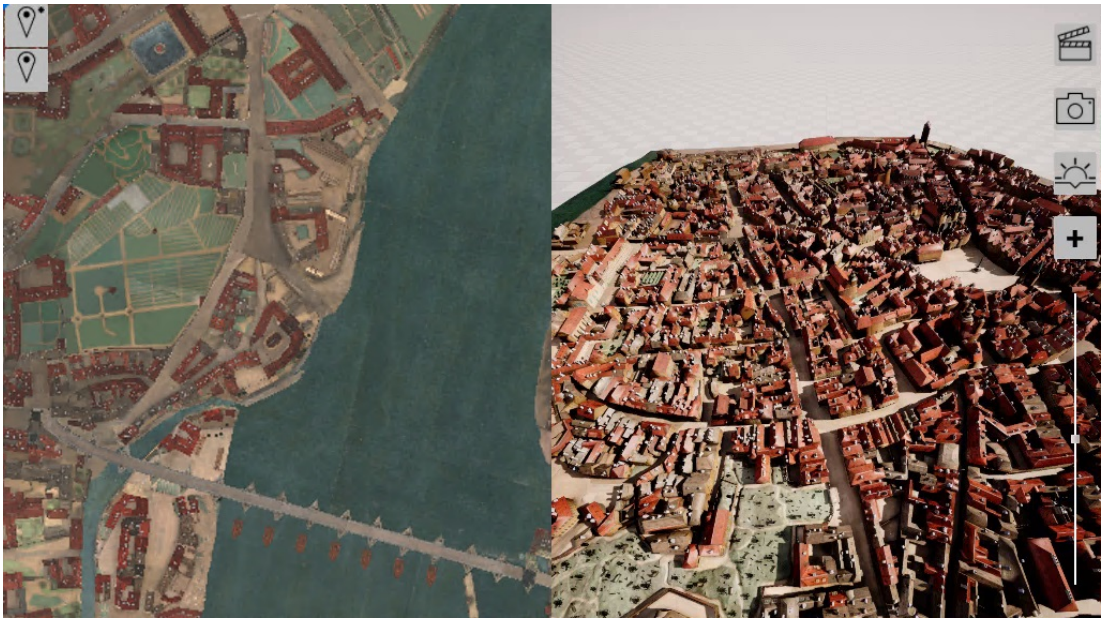
Obrázek 4.22. Prvotní návrh aplikace.

Na základě její zpětné vazby jsem rozhraní upravil. Přepínání mezi dnem a nocí bylo nahrazeno za tři různé úhly slunce. Slider na úpravu výšky byl přesunut doprava dolů. Přidal jsem také zobrazení názvu významného místa, na které směřuje kamera. Přesunul jsem také tlačítka pro hlasování k názvu místa.

Poslední věc, kterou jsem upravoval bylo ovládání. Prezentovaná verze umožňovala na pravé polovině pouze otáčení kamery. Díky zpětné vazbě jsem zjistil, která gesta jsou ještě potřeba. Přidal jsem tedy 3 gesta pro pohyb po modelu.

Pro lepší představu, jak výsledná aplikace vypadá, přikládám i fotku 4.23 sestavy, která byla využívána při testování s uživateli. Snímky obrazovek finální aplikace můžete vidět na obrázcích 4.24 a 4.25.

Obrázek 4.23. Sestava pro testování.



Obrázek 4.24. Náhled dotykové části grafického rozhraní.

Obrázek 4.25. Náhled televizní části grafického rozhraní.

Celé grafické rozhraní se skládá z těchto widgetů:

- UserInterface
- CameraPreviewWidget
- Map
- POI
- AdminWidget
- POIRow

Map

Mapu můžete vidět v levé polovině obrázku 4.24. Slouží k lepší orientaci uživatele a také k jednoduššímu přesunu na delší vzdálenost. Samotná textura mapy je předrenderovaná pomocí příkazu `HighResShot` a má rozměry 7955x8950px.

Ťuknutím na mapu, se náhled přesune na danou pozici se zachovanou rotací. Pro posun mapy je zapotřebí ťuknutí s tažením. Kvůli rozpoznání zda chce uživatel přesunout náhled nebo posunout mapu, je vytvořena proměnná „StartDraggingOffset“. Ta má v základním nastavení hodnotu 10. Při pohybu prstu se porovnává, zda vzdálenost mezi začáteční a aktuální pozicí prstu je větší než daná proměnná. Pokud ano, tak se přejde do režimu posunu mapy. Při zvednutí prstu se pak testuje, zda bylo hnuto s mapou a pokud nebylo, tak se přesune náhled.

Pomocí dvou prstů naráz lze mapu přiblížit či oddálit. Pro tuto akci je potřeba provést standardní gesto, kdy upravujeme vzdálenost meziprsty. Uvnitř funkce `Zoom_Touch` se nachází uzel `Clamp (Float)` na který je napojený vstup `Scale`. Ten zabraňuje moc velkému přiblížení a oddálení. Kdybychom mapu moc přiblížili, tak bychom ji viděli až moc rozpixelovanou. Při velkém oddálení bychom zase viděli za hranice textury, což je nežádoucí. Jelikož maximální a minimální hodnota měřítka je daná velikostí textury, tak tyto hodnoty lze upravit pouze přímo ve funkci.

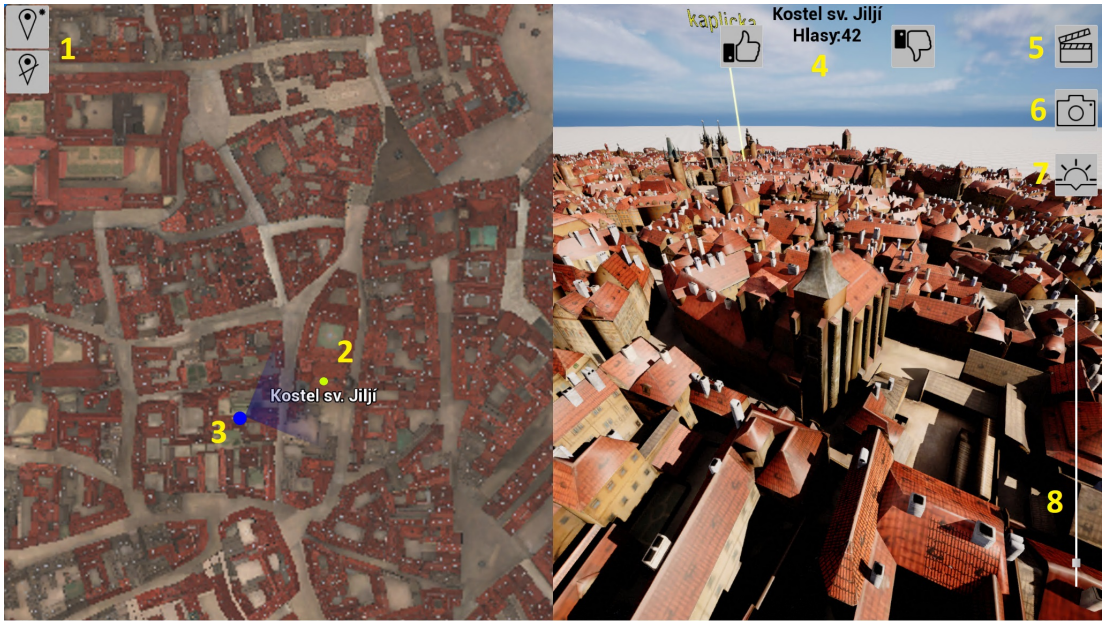
Další funkcí, kterou mapa nabízí je zobrazování významných míst. Ta se rozdělují do dvou skupin: originální a uživatelská. Originální jsou předpřipravená a určí si je odborníci z muzea. Zobrazují se modře. Uživatelská jsou místa, která navrli uživatelé. O této možnosti se dozvíte více později v této sekci. Hlavní rozdíl mezi těmito skupinami

je možnost hlasování u uživatelských významných míst. Pokud má místo 0 hlasů, tak se zobrazuje žlutě. Zeleně se znázorňují body se 100 a více hlasy. Prostor mezi těmito hodnotami je vyplněn lineární interpolací mezi žlutou a zelenou barvou.

Mapa také spravuje hlasování významných míst. Pokud hlasy klesnou pod 0, tak je dané místo odstraněno.

Při tvorbě mapy jsem se inspiroval z projektu, který sdílel uživatel Everynone v diskuzi[1] na unreal engine foru. Musel jsem ho upravit tak, aby jako vstupy využíval dotyk místo myši, ale hlavní myšlenka je totožná.

Na mapě také nalezneme bod, který vyobrazuje pozici kamery. Můžete ho vidět na obrázku 4.26 pod číslem 3.



Obrázek 4.26. Očíslované grafické rozhraní na dotykovém monitoru.

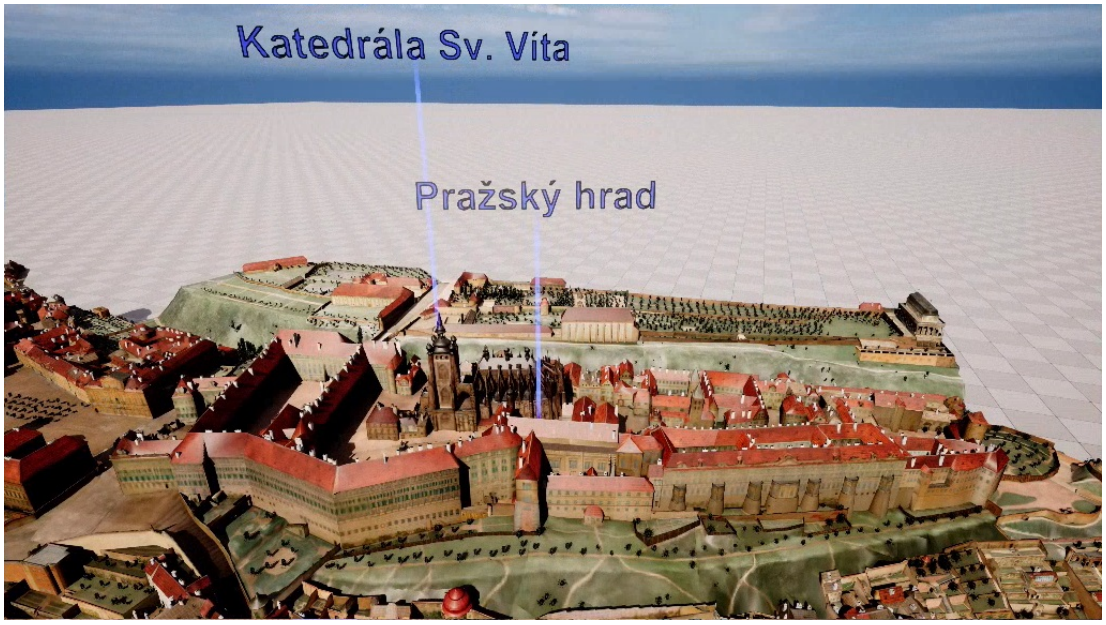
Při spuštění aplikace mapa načte významná místa z JSON souborů ve složce `./Data/`. Z těchto dat následně vytvoří ve scéně paprsky, které znázorňují pozice významných míst. Můžete je vidět na obrázku 4.27. Také umístí na mapu widgety typu POI a předá jim data, která potřebují k funkčnosti.

Paprsky

Při vytvoření paprsku se podle výšky základny určí jeho velikost. Ta zaručuje, že text bude umístěn v jedné ze tří úrovní. Hranice pro určení úrovně jsou 150 a 300. Na obrázku 4.27 se nachází dva paprsky a oba jsou v jiné úrovni.

Název významného bodu se neustále natáčí tak, aby vždy směřoval ke kameře. Barevné označení odpovídá již zmiňované konvenci. Pokud je paprsek méně než 500 m od kamery, tak se nezobrazuje.

[1] `https://forums.unrealengine.com/t/how-to-scale-zoom-a-widget-with-blueprints/430983/25`



Obrázek 4.27. Paprsky znázorňující významné místo.

POI

POI je zkratka pro „point of interest“, což lze přeložit jako významné místo. Je to widget, který se zobrazuje na mapě na pozici, která odpovídá danému významnému místu. Ve 3D prostoru se na tomto místě nachází paprsek. Můžeto ho vidět na obrázku 4.26 pod číslem 2.

Při ťuknutí na tento widget se kamera náhledu přesune na místo, kde se nacházela při vytváření bodu. Aby někdo nemohl neustále hlasovat na jedno místo pořád dokola, tak jsem zavedl čekací dobu 5 minut mezi hlasováním. Tu obstarává tento widget.

CameraPreviewWidget

CameraPreviewWidget je hlavním způsobem na ovládání ve 3D prostoru. Původně obsahoval obrázek, který jako zdroj využíval texturu, do které renderoval „Scene Capture Component 2D“. Ve finální verzi je však naprosto průhledný.

Celé jeho ovládání je založeno na gestech. Ťuknutí a tažení ovládá rotaci kamery. Při ní zůstává kamera na místě.

Pomocí dvojího poklepání na jedno místo, se kamera přesune nad toto místo. Výška nad zemí zůstává zachována. Zjištění na jaké místo bylo ťuknuto, se z počátku zdálo velice obtížné. Řešení bylo nakonec celkem jednoduché. Nejdříve jsem si zjistil vektory z kamery směřující do středů hran pohledu v lokálních souřadnicích. Následně ze souřadnic doteku lze pomocí lineární interpolace mezi již zmíněnými vektory vypočítat vektor daného směru v lokálních souřadnicích. Ten pak již stačí pouzepřevést do globálních souřadnic a vypočítat průnik paprsku.

Zbylá dvě gesta jsou pomocí dvou prstů. První je totožné s přiblížením na mapě. V tomto případě se náhled posune dopředu či dozadu směrem, který se vypočítá stejně jako při dvojím poklepání. Jediný rozdíl je v tom, že se využije bod přesně uprostřed mezi prsty.

Poslední gesto se vyvolá posunem dvou prstů po obrazovce. To způsobí posun v rovině kamery, tedy nahoru, dolů nebo do stran. Pro rozlišení těchto dvou gest je zavedena proměnná `ZoomTreshold`, která se porovnává s absolutní hodnotou vzdálenosti mezi

prsty od posledního aktualizace. Ve chvíli, kdy vzdálenost přesáhne prahovou hodnotu, tak se přepne do režimu približování.

UserInterface

Widget se jménem UserInterface je nejdůležitější ze všech. Ten totiž všechny ostatní spojuje do jednoho celku. Také přidává několik tlačítek a obstarává jejich logiku.

Na obrázku 4.26 u čísla 1 se nachází dvě tlačítka s téměř totožnou ikonkou. Tato tlačítka se využívají k zobrazení a schování významných míst. Reagují na ně jak widgety na mapě, tak paprsky ve scéně. Horní tlačítko s hvězdičkou ovládá originální místa. Uživateli vytvořená místa jsou ovládána spodním tlačítkem.

Pod číslem 4 na obrázku 4.26 najdeme popisek významného místa, které se nachází uprostřed náhledu. Pokud náhled nesměřuje na žádné významné místo, tak je popisek prázdný. V případě, že je pozorované místo originální, tak se zde zobrazuje pouze název. U uživateli vytvořených míst se zobrazuje i počet hlasů. Palec nahoru a palec dolů zde nalezneme pouze pokud na daný bod lze hlasovat. To znamená, že zmízí na 5 minut po hlasování, což jak již víte obstarává POI widget. Kliknutím na palec se přidá nebo ubere jeden hlas.

Obrázek 4.28. Grafické rozhraní s rozbalenou volbou průletů a jiným nasvícením.

Tlačíto u čísla 5 na obrázku 4.26 slouží k přehrávání předpřipravených průletů. Po stisku se rozbalí číselná volba průletů, kterou můžete vidět na obrázku 4.28 v pravém horním rohu. První možnost spustí sekvenci, která postupně přepíná pohledy na všechna významná místa. Tato sekvence je jediná, která se dá zrušit dotykem kamkoli na obrazovku. Pokud se obrazovky nikdo nedotkl po dobu 5 minut, tak se tato sekvence spustí sama. Zbylé možnosti přehrají předpřipravený průlet. Tyto průlety jsou provizorní a jejich konečnou podobu si vymyslí odborníci z muzea. Tlačítko se znakem > zabalí tuto nabídku do původní podoby.

Tlačítko s ikonkou fotoaparátu u čísla 6 na obrázku 4.26 je určeno k vytvoření snímku obrazovky. Po jeho stisknutí se otevře přes celou obrazovku rozhraní pro zadání emailu. To můžete vidět na obrázku 4.29. Zde může uživatel zadat svoji emailovou adresu na kterou se zašle snímek obrazovky. K pořízení snímku obrazovky se využívá „Scene Capture Component 2D“. Ten je připojen ke kameře, která se zobrazuje na televizi.

Obrázek 4.29. Rozhraní pro zaslání emailu.

Kvůli této akci jsem musel vytvořit 3 vlastní blueprint uzly. Ty se vytváří pomocí jazyka C++.

`KeyboardControl.cpp` se stará o otevření a zavření virtuální klávesnice, která je potřeba na dotykovém monitoru. Přesněji spustí soubor `./Data/Keyboard/Keyboard.exe` s parametrem `show` pro zobrazení nebo `hide` pro schování.

`SendEmail.cpp` má za úkol odeslání emailu. Stejně jako předchozí script pouze spustí soubor `./Data/Email/Email.exe` s pěti parametry. Ty určují jak příjemce, tak i email odesilatele s potřebnými údaji pro odeslání.

`EmailValidation.cpp` ověří platnost emailu pomocí regexu.

Python scripty, které jsem využil pro vytvoření `.exe` souborů popíši podrobněji později v této sekci.

Číslo 7 na obrázku 4.26 přísluší tlačítku, které upravuje osvětlení. Nabízí tři možnosti: východ slunce, západ slunce a poledne. Ikona tlačítka se mění na obrázek značící příští stav. Například na obrázku 4.28 znázorňuje přepnutí na poledne.

Pod číslem 8 na obrázku 4.26 nalezneme slider. Jeho funkcí je nastavení výšky. Minimální hodnota se nechází 10 jednotek nad objektem přímo pod kamerou. Maximální hodnota přesune kameru 1000 jednotek nad minimum.

Poslední tlačíto nalezneme na obrázku 4.28 vpravo. Jeho ikona je znak + a nachází se mezi sliderem a ovládáním osvětlení. Toto tlačítko je vidět pouze pokud se uprostřed náhledu nenachází žádné významné místo. Po jeho stisknutí se zobrazí rozhraní podobné při vytváření snímku obrazovky. Můžete ho vidět na obrázku 4.30.

Uživatel zde může zadat název jím navrhovaného významného místa. Po potvrzení se vytvoří místo s 10 hlasy. Jeho pozice se vypočítá jako střed osově zarovnaného kvádru, který obepisuje objekt nacházející se uprostřed náhledu. Také se uloží transformace kamery, která se použije v budoucnu pro zobrazení významného bodu dalšími uživateli.

Obrázek 4.30. Rozhraní pro návrh významného místa.

Ovládání virtuální klávesnice

Jako virtuální klávesnici využívám tu, která je zabudovaná přímo v systému Windows[1]. Jelikož se mi neporařilo zprovoznit její automatické zobrazování, tak jsem zvolil jiný přístup. Klávesnici lze vyvolat pomocí klávesové zkratky Ctrl + Win + O. Napsal jsem tedy Python script, který využívá knihovnu `pywinauto` k tomu, aby nasimulovala zmáčknutí těchto kláves. Jelikož jedna klávesová zkratka klávesnici zobrazuje i schovává, tak bylo jednoduché invertovat zobrazení klávesnice. Abych tomu předešel, tak pomocí knihovny subprocess nejdříve zjistím, zda běží program `osk.exe`. Dále jsem přidal nutnost parametru `show` nebo `hide`. Díky těmto dvěma informacím mohu ověřit zda je nutné klávesovou zkratku vyvolat. Výsledný python script jsem sestavil pomocí pyinstalleru s argumentem `--noconsole`, který zakáže zobrazení konzole při spuštění.

Odeslání emailu

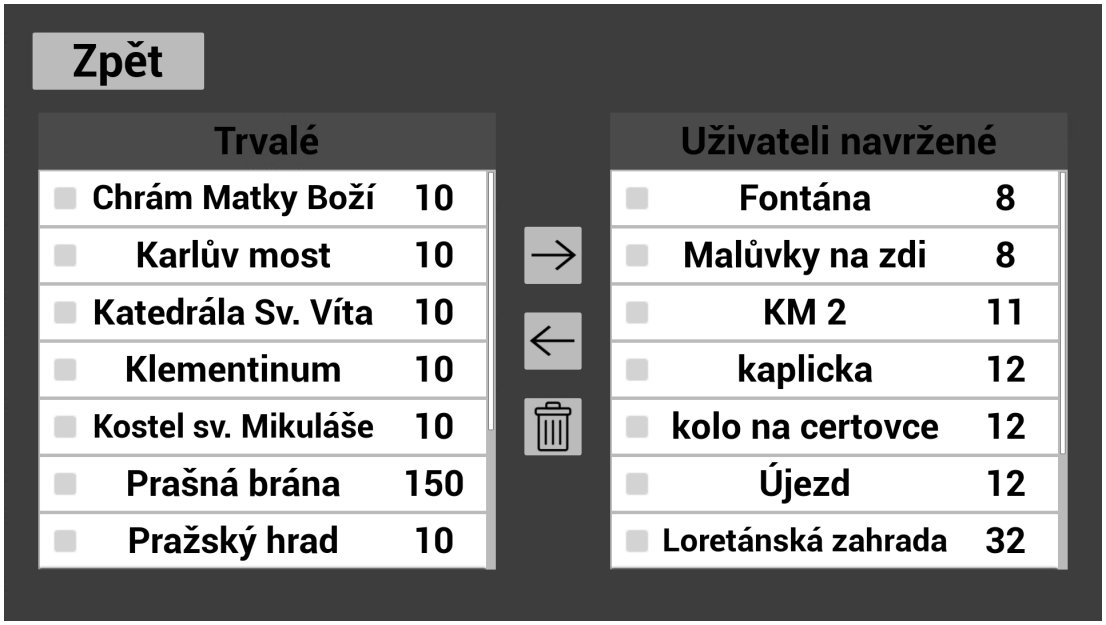
Pro odeslání emailu jsem také napsal Python script. Ten využívá knihovny `email` a `smtplib`. SMTP, neboli „Simple Mail Transfer Protocol“, je internetový protokol, který lze využít na posílání emailů. Script je nutno spustit s argumenty ve tvaru: `odesílající příjemce server heslo [cesta k příloze]`. Příloha je volitelná. Script jsem testoval pouze s gmailem. Výsledek jsem sestavil stejně jako script pro ovládání virtuální klávesnice.

AdminWidget

Pokud 5x rychle za sebou rozbalíme a sbalíme menu na spušetění průletů, tak se nám spustí panel pro správce. Můžete ho vidět na obrázku 4.31. V něm nalezneme dva sloupečky, které se skládají z widgetů POIRow. Každý řádek odpovídá jedomu významnému místu a zobrazuje jeho název a počet hlasů. Uživatel může vybrat místa pomocí zaškrtávajících políček a následně prosvést akci pomocí jednoho ze tří tlačítek, která se nachází mezi sloupečky. Šipka vpravo přesune vybraná originální místa mezi uži-

[1] `https://www.microsoft.com/cs-cz/windows`

vateli navržená. Šipka vlevo přesune vybraná uživateli navržená místa mezi originální. Tlačítko s košem po potvrzení varovné hlášky nevratně odstraní všechna vybraná místa.



Obrázek 4.31. Panel pro správu významných míst.

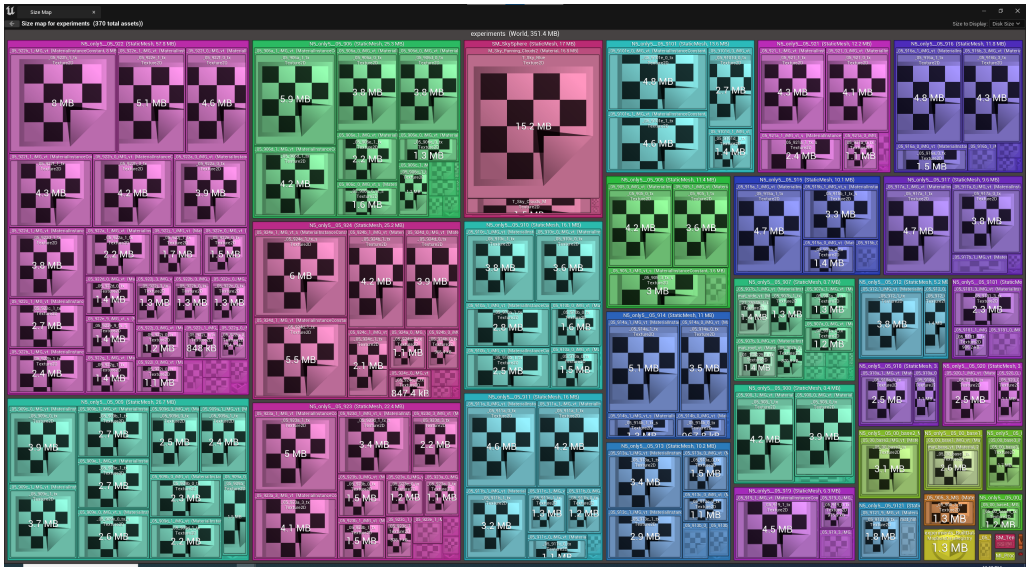
Ukázková videa

Pro lepší představu, jak přesně aplikace funguje jsem natočil dvě videa a nahrál je na YouTube[1]. Na prvním videu[2] naleznete pohledy obou monitorů. Druhé video[3] obsahuje pouze obraz z dotykového monitoru. V obou případech jsem předvedl všechny funkce, které aplikace nabízí.

[1] https://www.youtube.com
[2] https://youtu.be/Z5EHjgnjNOA
[3] https://youtu.be/kLXLvRltytc

Kapitola 5
Testování

Než začnu s analýzou testování, tak je potřeba zmínit ještě jedna věc. Unreal Engine umožňuje zobrazení takzvané „Size map“ pro každou scénu. V té se nám vyobrazí všechny modely v dané scéně i s odpovídajícími materiály a texturami v sobě. Tyto objekty se vykreslí do mřížky, kde jejich rozměry vyjadřují velikost v paměti nebo na disku (podle zvoleného módu). To můžete názorně vidět na obrázku 5.1 z vedlejší scény.



Obrázek 5.1. Size mapa z vedlejší scény.

Obrázek 5.2. Size mapa z hlavní scény.

Z mě neznámého důvodu bohužel tato funkce v hlavní scéně nefunguje. Na obrázku 5.2 lze vidět Size mapu pro hlavní scénu. Zde se vykresluje pouze 7 assetů.

Hlavní problém nastává až v pozdější fázi, kdy při měření se téměř všechny textury nerozpoznávají a tudíž se nezobrazují v části „LLM Textures“. Zkratka „LLM“ znamená „Low Level Memory“, neboli paměť na grafické kartě. Bude nás tedy zajímat spíše hodnoty „LLM Total“ nebo „LLM Untracked“. Rozhodl jsem se využívat „LLM Total“, která sice obsahuje položky navíc, ale k účelu zjištění dopadu provedené optimalizace stačí.

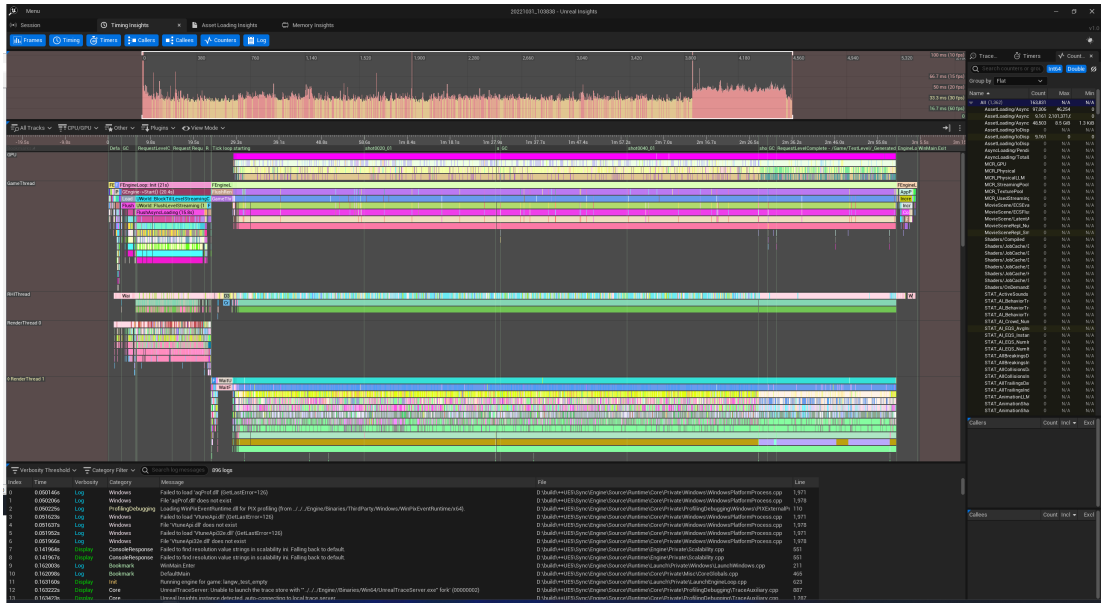
5.1 Testování optimalizace

Data jsem měřil na svém stolním počítači s parametry:

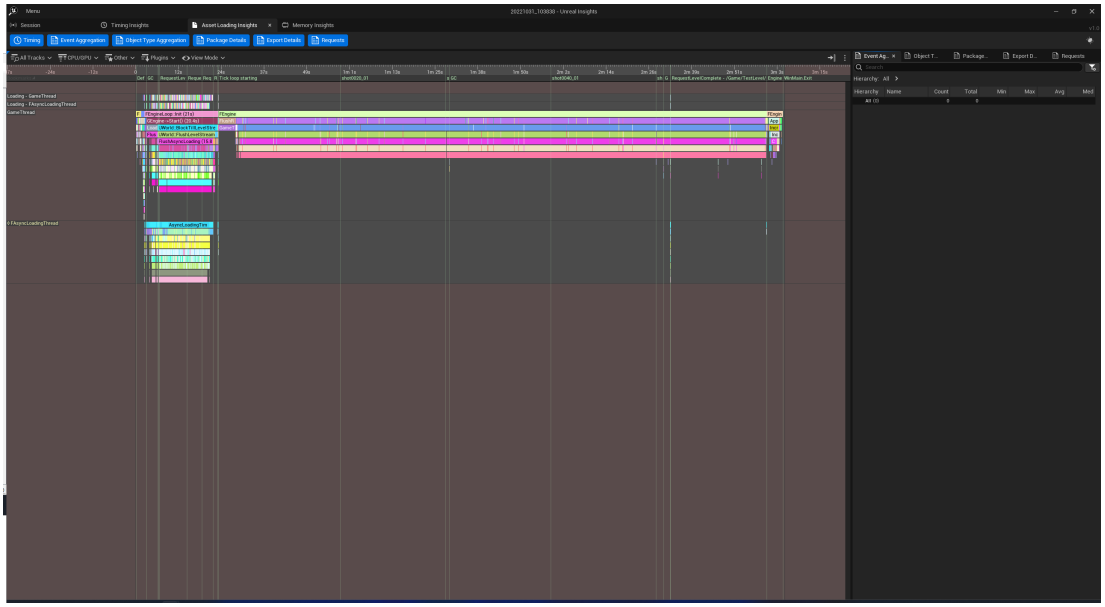
- ASUS ROG STRIX RTX 3070 Ti 8 GB
- 11th Gen Intel Core i7-11700 2,5 GHz
- obrazovka: 2560x1440 164,83 Hz
- 32 GB RAM
- Windows 11

Data z Unreal Insights

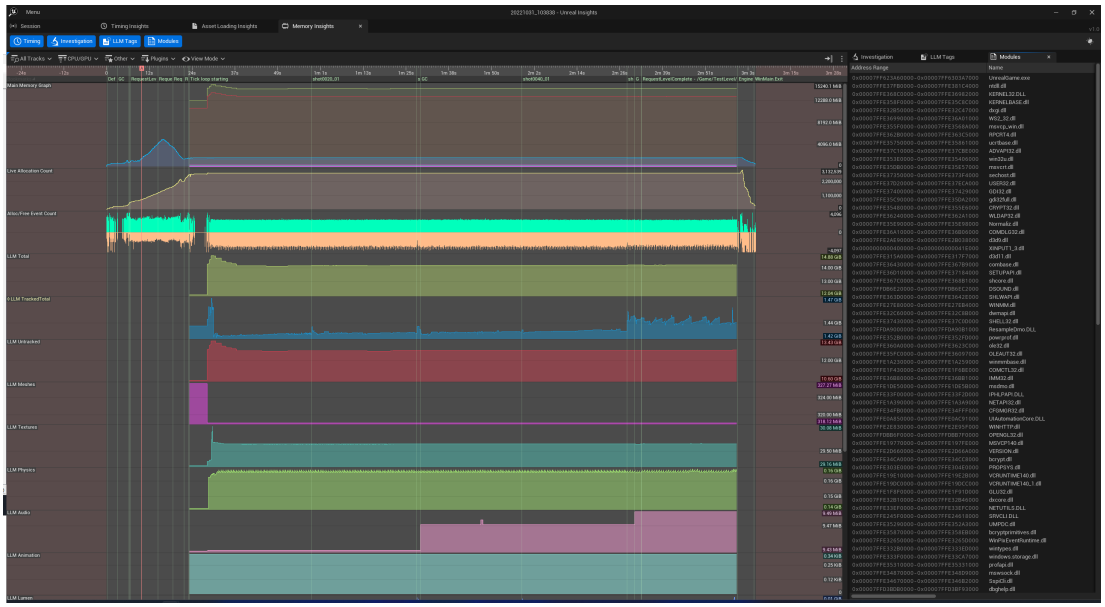
Na následujících obrázcích 5.3 5.4 5.5 jsou vyobrazena naměřená data z prvního buildu aplikace, který obsahuje pouze průlety bez jakýchkoliv optimalizací. Slouží tedy jako referenční bod. Jelikož je celá aplikace Unreal Insights spíše určena k interakci, tak jsou tyto obrázky spíše ilustrační pro představu, jak vypadá výsledek měření. Vytvořil jsem tedy tabulky s daty, které jsou dle mého názoru nejdůležitější. Těmi jsou průměrné snímky za sekundu a hodnota zaplnění paměti na grafické kartě. Jelikož při načítání vždy zaplnění stouplo vysoko a následně se hodnota snížila a ustálila s již malými výkyvy, tak jsem se rozhodl udávat tuto hodnotu jako maximální a ustálenou. Každý build jsem měřil 3x a následně vypočítal průměr těchto hodnot.



Obrázek 5.3. Časovače - build 1.



Obrázek 5.4. Načítání assetů - build 1.



Obrázek 5.5. Paměť - build 1.

Build 1

Tento build neobsahuje žádné optimalizace. Slouží jakožto referenční bod. V této a několika další verzích můžeme v levém horním rohu vidět chybovou hlášku: „Video memory has been exhausted“. Můžete ji také vidět i na obrázku 5.6.

Obrázek 5.6. Chybová hláška při nedostatku paměti.

Build 1			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	37.69	14.58	14.16
2	36.03	14.8	14.14
3	36.79	14.98	14.17
průměr	36.84	14.79	14.16

Tabulka 5.1. Naměřené hodnoty 1. buildu.

Naměřená data z této verze můžete vidět v tabulce 5.1. Byl jsem celkem překvapen, že průměrný počet snímků za sekundu byl 36,84, což je mnohem více, než bych čekal. Také v ní můžeme vidět, že ustálené vytížení paměti grafické karty je průměrně 14,16 GB. Což je více než má grafická karta na testovacím počítači. Proto se také zobrazovala již zmíněná chybová hláška.

Build 2

První změnou, kterou jsem provedl bylo vyměnění materiálů komínů. Naměřená data naleznete v tabulce 5.2. Průměrné zaplnění paměti se snížilo o 0,28 GB. Textury komínů byly sice malých rozměrů, ale bylo jich mnoho. Dále také nejspíše ubyla data o materiálech, které byly nahrazeny za instancované.

Můžeme také pozorovat snížení snímků za sekundu. Jelikož je paměť stále přeplněná, tak je nucena neustále načítat a zapomínat textury. Proto z toho nemůžeme nic vyvozovat.

Build 2			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	36.88	14.15	13.86
2	36.12	14.14	13.85
3	35.93	14.62	13.94
průměr	36.31	14.3	13.88

Tabulka 5.2. Naměřené hodnoty 2. buildu.

Build 3

Druhá změna byla nahrazení všech materiálů za instancované. Jak můžete vidět na tabulce 5.3, tak tento krok neměl nijak zvášť velký dopad na výslednou aplikaci. Hlavním cílem instancovaných materiálů je usnadnění práce při vývoji. Každopádně mají i malý pozitivní dopad na paměť grafické karty.

Build 3			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	37.41	14.08	13.81
2	36.08	14.08	13.81
3	35.25	14.52	13.84
průměr	36.25	14.23	13.82

Tabulka 5.3. Naměřené hodnoty 3. buildu.

Build 4

Další úprava spočívala ve využití virtuálních textur. Jak jsem již zmiňoval v implementaci, tak první verze, která nahrazovala texturu u všech s oběma rozmery většími než 512px, byla po vizuální stránce nepoužitelná. Proto jsem ji ani neměřil. Přeškálování textur mělo však i veliký dopad na velikost sestavené aplikace. Přičemž 3. verze zabírala 7,83 GB, tak nynější zabírá 11,3 GB.

Data následujících verzí se nachází v tabulkách 5.4 a 5.5. Verze 4.1 využívá virtuální textury pouze u obrázků s oběma rozměry většími než 4096px. Již zde je znatelný pokles průměrného zaplnění paměti o 0,26 GB.

Mnohem znatelnější vylepšení poskytla verze 4.2, která měla virtuální textury u všech obrázků s oběma rozměry alespoň 2048px. Celkové snížení je v tomto případě o dalších 6,25 GB. Jedná se také o první verzi, ve které se nenachází již zmíněná chybová hláška v levém horním rohu, kterou jste mohli vidět na obrázku 5.6. Tato změna také způsobila pozitivní dopad i na průměrné snímky za sekundu, které stouply na 37,92.

Build 4.1			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	37.74	13.87	13.56
2	35.47	13.85	13.57
3	35.46	14.41	13.55
průměr	36.22	14.04	13.56

Tabulka 5.4. Naměřené hodnoty 4.1. buildu.

Build 4.2			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	38.48	7.28	6.99
2	36.96	7.28	7.01
3	38.32	7.34	7.03
průměr	37.92	7.3	7.01

Tabulka 5.5. Naměřené hodnoty 4.2. buildu.

Build 5

V páté úpravě se jednalo o přeškálování všech textur na mocniny dvou. To umožnilo automatickou generaci mipmap. V tabulce 5.6 můžete vidět, že průměrné zaplnění paměti se snížilo o dalších 1,16 GB na 5,85 GB. Snímky za sekundu zase stouply na 38,65. Tato verze zvýšila velikost aplikace o 0,1 GB na 11,4 GB, což je zanedbatelné.

Build 5			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	38.32	6.02	5.81
2	38.31	5.96	5.85
3	39.33	6.04	5.88
průměr	38.65	6.01	5.85

Tabulka 5.6. Naměřené hodnoty 5. buildu.

Build 6

V šesté verzi jsem aktivoval Nanite u všech modelů. Jak již víte z implementační části v sekci 4.5, tak to mělo negativní dopad. Naměřená data naleznete v tabulce 5.7. Celkové zaplnění paměti stouplo o 0,66 GB a průměrný počet snímků za sekundu klesl na 24,75.

Předpokládám, že za to může správa všech Nanite objektů. Při ní totiž musí neustále zjišťovat vzdálenosti a rozhodovat přepínání mezi úrovněmi detailů.

Build 6			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	24.33	6.62	6.5
2	24.65	6.59	6.49
3	25.27	6.65	6.54
průměr	24.75	6.62	6.51

Tabulka 5.7. Naměřené hodnoty 6. buildu.

Build 7

V sedmé verzi jsem tedy deaktivoval Nanite u všech objektů. Některé objekty měly Nanite aktivovaný již od samého začátku. Jedná se v podstatě o úpravu páté verze a proto budu tyto dvě verze srovnávat.

Jak můžete vidět v tabulce 5.8, tak zaplnění paměti je v podstatě totožné. Změna nastává ve snímkách za sekundu. Ty průměrně vzrostli o 0,4. To se však dalo očekávat, protože v minulé verzi jsme zjistili, že Nanite má negativní dopad na fps.

Build 7			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	38.84	6.0	5.84
2	39.01	5.99	5.81
3	39.29	6.05	5.87
průměr	39.05	6.01	5.84

Tabulka 5.8. Naměřené hodnoty 7. buildu.

Build 8

V předchozích verzích jsem hodnotil, zda měla změna pozitivní vliv. U této však budu hodnotit, jak moc negativně se úprava projevila. Jedná se totiž o výměnu stromů za detailnější model.

Zde jsem nahrazoval za stažený model jehličnanu. Cílem bylo zjistit, zda daný typ modelu bude mít malý dopad a tudíž ho budu moci využít k tvorbě nového modelu. V tabulce 5.9 zjistíme, že průměrné snímky za sekundu se snížili o 2,05. Jelikož se stále držíme nad 30 snímky za sekundu, což byl mnou určený limit, tak můžeme tento typ modelu využít.

Celkem mě překvapilo, že průměrné zaplnění paměti grafické karty se snížilo o dalších 0,12 GB. Když jsem se nad tím pozastavil, tak to začalo dávat smysl. Předchozí stromy byly spojeny do jednoho modelu, ale aktuální stromy jsou ve scéně rozmístěny jako

Build 8			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	36.84	5.9	5.69
2	37.29	5.84	5.66
3	36.86	5.97	5.8
průměr	37.0	5.9	5.72

Tabulka 5.9. Naměřené hodnoty 8. buildu.

jednotlivé objekty. Před tím totiž musela grafická karta načíst celý model všech stromů i když byl vidět pouze jeden.

Build 9

Devátá úprava již využívala k výměně stromů můj model. Naměřená data se nachází v tabulce 5.10. Došlo zde k dalšímu poklesu zaplnění paměti o 0,37 GB. Očekával jsem spíše nárůst, protože stažený strom měl vyšší počet trojúhelníků pouze oproti nejmenšímu stromu. Na druhou stranu jeho textura kmene byla větších rozměrů. Snímky za sekundu jsou v podstatě totožné.

Build 9			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	37.03	5.47	5.35
2	36.94	5.52	5.3
3	37.18	5.58	5.41
průměr	37.05	5.52	5.35

Tabulka 5.10. Naměřené hodnoty 9. buildu.

Build 10

Poslední verze nepřináší žádnou úpravu uvnitř projektu. Rozdíl je v sestavení, které probíhalo v režimu „Shipping“. V dokumentaci Unreal Insights na stránce zaměřené na měření paměti na grafické kartě[1] se nachází v červeném ohraničení varování, že je potřeba, aby sestavená verze byla v režimu „Development“. Já jsem však zkusil změřit i tuto verzi a vše fungovalo. Jak můžete vidět v tabulce 5.11, tak dopad byl pozitivní, ale minimální.

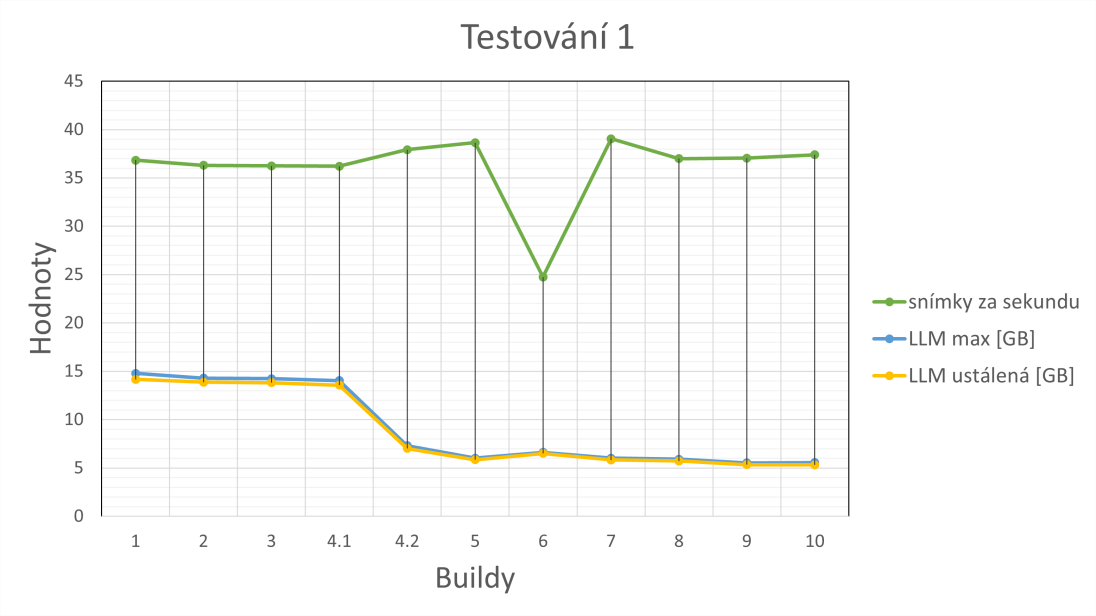
Build 10			
číslo měření	snímky za sekundu	LLM total maximum [GB]	LLM total ustálená [GB]
1	38.02	5.53	5.32
2	37.18	5.56	5.33
3	37.01	5.59	5.33
průměr	37.4	5.56	5.33

Tabulka 5.11. Naměřené hodnoty 10. buildu.

[1] `https://docs.unrealengine.com/5.0/en-US/memory-insights-in-unreal-engine/`

Pro lepší představu o dopadu optimalizačních kroků na aplikaci jsem vytvořil graf 5.7. Můžeme v něm vidět postupný nárůst snímků za sekundu, který velmi klesl při využití technologie Nanite. Následně ale klesl i při výměně modelu stromů. Ten však zdaleka nebyl tak moc razantní.

Dále v něm lze vyčíst také zaplnění paměti na grafické kartě. U něho můžeme pozorovat, že nejvíce kleslo při využití virtuálních textur. Také je zde menší nárůst u 6. verze, tedy při využití technologie Nanite.



Obrázek 5.7. Graf z prvního testování.

Provedl jsem také měření i na jiném počítači. Zde jsem však měřil pouze verze, které byly využité ve výsledné aplikaci a měly nějaký dopad na výkon. Zvolil jsem build 1, 4.2, 5, 7 a 10.

Tato data jsem měřil na školním počítači s parametry:
- NVIDIA Titan
- 11th Gen Intel Core i9-11700 2,5 GHz
- obrazovka: 2560x1440 164,83 Hz
- 64 GB RAM
- Windows 10

Naměřené hodnoty můžete vidět v tabulkách: 5.12, 5.13, 5.14, 5.15 a 5.16. Po prozkoumání dat dojdeme ke stejnému závěru jako při prvním testování. Pro lepší přehled jsem také vytvořil graf 5.8. V něm lze znovu pozorovat nárust snímků za sekundu a následný pokles při výměně modelu stromu.

Build 1			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	26.75	14.76	14.53
2	25.25	15.13	14.57
3	26.09	15.19	14.58
průměr	26.03	15.03	14.56

Tabulka 5.12. Naměřené hodnoty 1. buildu.

Build 4.2			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	27.75	7.78	7.42
2	26.75	7.85	7.51
3	27.29	7.84	7.48
průměr	27.26	7.82	7.47

Tabulka 5.13. Naměřené hodnoty 4.2. buildu.

Build 5			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	28.06	6.42	6.24
2	27.79	6.56	6.36
3	27.81	6.55	6.32
průměr	27.89	6.51	6.31

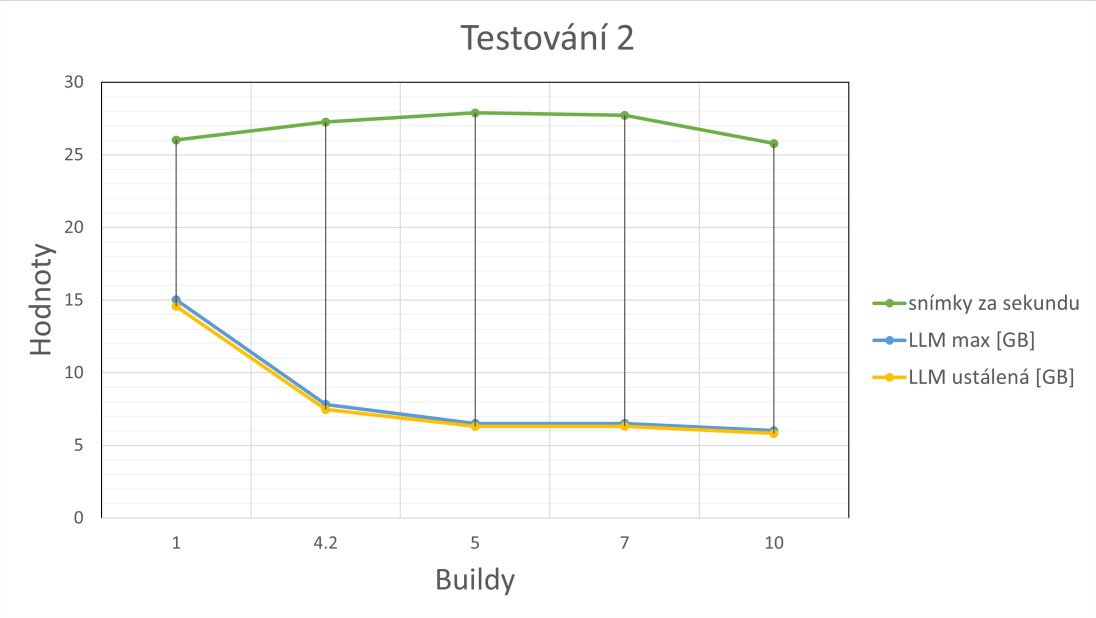
Tabulka 5.14. Naměřené hodnoty 5. buildu.

Build 7			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	27.69	6.43	6.25
2	27.81	6.55	6.36
3	27.67	6.54	6.35
průměr	27.72	6.51	6.32

Tabulka 5.15. Naměřené hodnoty 7. buildu.

Build 10			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	25.69	5.95	5.76
2	25.82	6.05	5.83
3	25.84	6.02	5.86
průměr	25.78	6.01	5.82

Tabulka 5.16. Naměřené hodnoty 10. buildu.



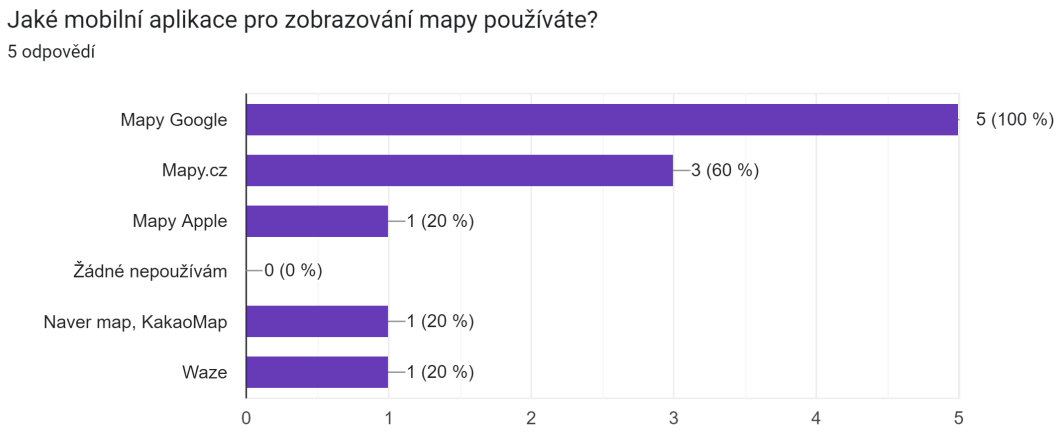
Obrázek 5.8. Graf z druhého testování.

5.2 Testování grafického rozhraní

Pro testování grafického rozhraní jsem si pozval 2 ženy a 3 muže. Po seznámení s účelem této aplikace byli vyzváni k jejímu vyzkoušení. Během testování říkali své poznatky nahlas. Ty jsem si průběžně zapisoval. Po té, co si aplikaci vyzkoušeli, jsem jim ještě předložil dotazník[1].

Výsledky dotazníku

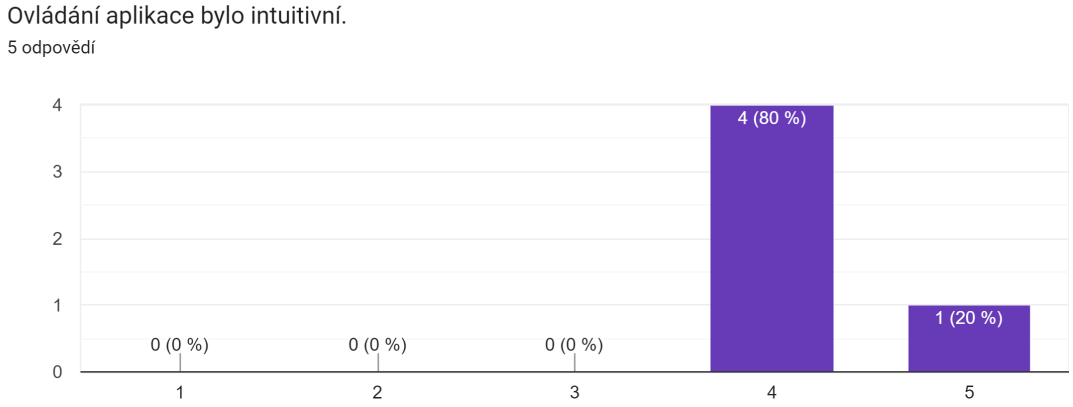
První otázka byla směřována na mobilní aplikace pro zobrazování map. Jak můžete vidět v grafu 5.9, tak všichni účastníci využívají Mapy Google. Tato skutečnost také ovlivnila testování, což se dozvíte v jedné z následujících otázek.



Obrázek 5.9. Graf z odpovědí na otázku: „Jaké mobilní aplikace pro zobrazování mapy používáte?“.

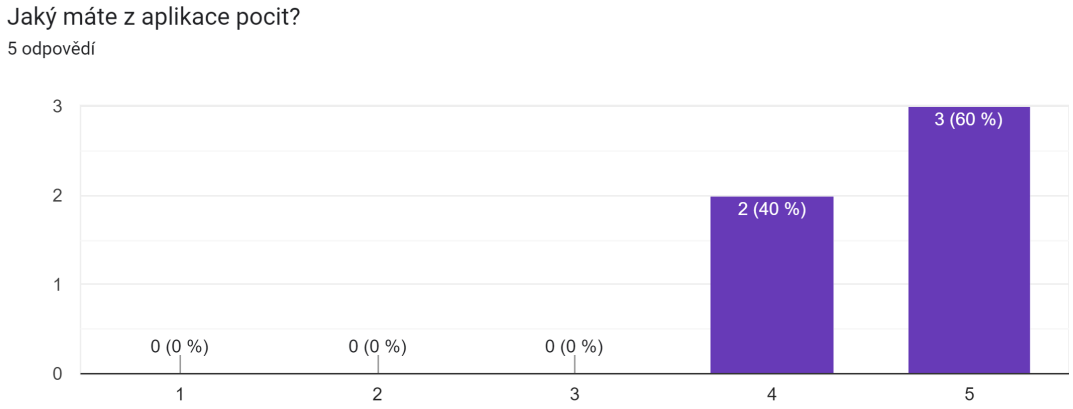
[1] `https://forms.gle/8X7zFGgXkkrbzEL59`

V následujících dvou otázkách byla možnost volby ze stupnice od 1 do 5. První tvrzení znělo: „Ovládání aplikace bylo intuitivní.“. Zde na stupnici možnost 1 znamenala nesouhlasím a 5 souhlasím. Účastníci se téměř shodli na odpovědi 4. Jediná rozdílná odpověď byla pro volbu 5. Graf z tohoto tvrzení naleznete na obrázku 5.10. Důvodem sníženého hodnocení u 4 lidí byla funkce, na kterou byli všichni zvyklí z aplikace Mapy Google.



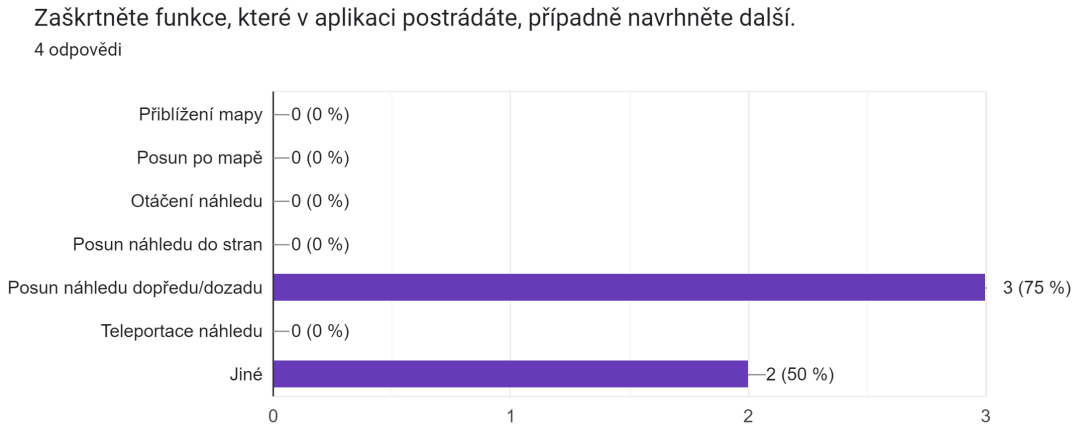
Obrázek 5.10. Graf z odpovědí na tvrzení: „Ovládání aplikace bylo intuitivní.“.

U otázky „Jaký máte z aplikace pocit?“ znamenala možnost 1 negativní a 5 pozitivní. Z grafu 5.11 vyčteme, že aplikace zanechávala pozitivní dojem.



Obrázek 5.11. Graf z odpovědí na otázku: „Jaký máte z aplikace pocit?“.

Poslední otázka, kde si účastník vybíral z nabídky, umožňovala zaškrtnutí více možností. Otázka zněla: „Zaškrtněte funkce, které v aplikaci postrádáte, případně navrhněte další.“. Aplikace obsahovala všechny funkce v nabídce, avšak jak můžete vidět v grafu 5.12, tak odpověď „Posun náhledu dopředu/dozadu“ byla zvolena třikrát. Tato odpověď odkazovala na možnost přiblížení a oddálení náhledu, kterou jsem již popisoval v sekci 4.8. Všichni, kteří zvolili tuto možnost, však zamýšleli gesto, které znali z aplikace Google Mapy. V ní totiž ve „3D pohledu“ můžete dotykem jednoho prstu posunut pohled v globální rovině XY. Na možnost „Jiné“ navazuje další otázka.



Obrázek 5.12. Graf z odpovědí na otázku: „Zaškrtněte funkce, které v aplikaci postrádáte, případně navrhněte další.“.

Následující otázky již nenabízely možnosti, avšak vyžadovaly textový vstup. První takováto otázka byla navazující na předchozí a zněla: „Pokud jste v předchozí otázce zvolil/a odpověď „Jiné“, zde je prosím popište.“. Jelikož možnost „Jiné“ zvolili dva lidé, tak i tato otázka má pouze dvě odpovědi.

První je „Pohyb XY“, což je odkaz na funkci z aplikace Google Mapy, avšak dotazovaný pochopil možnost „Posun náhledu dopředu/dozadu“ tak, jak byla zamýšlena.

Druhá odpověď zněla „Přelet ve 3D mapě z jednoho místa na druhý na monitoru“. Toto odkazuje na postupný posun kamery na novou pozici i u náhledu na dotykovém monitoru. Ten jsme s vedoucím zvažovali již při tvorbě aplikace. Pro instantní teleportaci jsme se rozhodli z důvodu rychlejší odezvy. Avšak uživatelé si posun kamery na televizi užívali a myslím si, že by ocenili průlet vidět i na dotykovém monitoru.

Na otázku „Proč jsou podle vás významná místa rozdělena na dvě skupiny?“ až na jednu z žen odpověděli všichni v podstatě správně. Dotyčná odpověděla: „Na to jsem nepřišla“.

Další otázka se také týkala významných míst a zněla: „Jaký je podle vás význam barev významných míst?“. Zde stejně jako v minulé otázce 4 lidé odpověděli správně, že se týká počtu hlasů. Poslední účastnice však odpověděla: „Nevím“.

Když jsem se všech účastníků dotazoval na rozdělení významných míst, tak z jejich odpovědí vyplynulo, že pro uživatele jako takového je bod jako bod. Každopádně po předložení situace, kdy například nějaké dítě v muzeu přidá spoustu významných míst, která budou naprosto nesmyslná, se jim možnost zobrazení pouze schválených míst jevila jako velice užitečná.

Otázka „Narazili jste na nějakou chybu?“ dotala pouze jednu kladnou odpověď, která byla „Rychlý posun v 3D zobrazení“. Na vysokou citlivost otáčení a přiblížení si stěžoval ještě jeden účastník, který to však nezmínil v dotazníku. Každopádně se jedná spíše o špatnou vlastnost, nežli o chybu.

Předposlední dotaz zněl „Bylo nějaké gesto, které dělalo něco jiného, než jste očekával/a?“. Dva lidé odpověděli pouze „Ne“ a jeden „Ne, všechna gesta jsou v rámci možností intuitivní.“. Další dvě odpovědi již popisovaly neočekávené chování.

První zmiňovala „Invertovaný pohyb otáčení.“. Týká se to rotace kamery v náhledu, který by měl podle účastníka testování mít invertované otáčení ve směru nahoru a dolů.

Úplně první dotyková verze měla přesně toto ovládání a bylo to naprosto matoucí. Tuto změnu navrhoval hlavně proto, že to byl hráč zvyklý na ovládání myší. Ve hrách je totiž standardní mít při pohybu myší k sobě otočit kameru k zemi. Jsem ale přesvědčen, že kdyby vyzkoušel invertované otáčení, tak by hned změnil svůj názor.

Druhá odpověď poukazovala na očíslování tlačítek průletů, která jsou naprosto nic neříkající a matoucí. S touto kritikou souhlasím, avšak mě nenapadá žádné jiné řešení.

Poslední otázka zněla: „Byla nějaká akce, kterou jste chtěl/a provést, ale nešlo to?“. Odpovědi na tento dotaz byly:

- Ano, pohyb po mapě pomocí jiných gest než přiblížení.
- Rychlý posun “scrollování” dopředu
- Ne
- Nevím
- Ano, nešla rotace kolem bodu a dvěma prsty pohyb XY

První, druhá a polovina poslední odpovědi jsou v podstatě totožné a odkazují na již zmiňovanou funkci z Google Map. Poslední odpověď ještě zmiňuje otáčení okolo bodu uprostřed obrazovky, které by mělo být prováděno točivým pohybem dvou prstů na obrazovce.

Zapsané poznatky

V této části vypíšu poznatky, které jsem si zapsal při pozorování účastníků. Abych neopakoval věci, které zmiňovali již v dotazníku, tak zmíním pouze ty navíc.

Padl návrh na vypnutí takzvaného „motion blur“, což je rozmazání obrazu při pohybu kamery. Je to věc, kterou většina počítačových hráčů nemá v oblibě, avšak lidé, kteří aktivně nehrají hry ji vnímají pozitivně. Jelikož v muzeu budou hráči menšinou, tak jsem se rozhodl „motion blur“ ponechat.

Další účastník by rád měl možnost vytvořit si vlastní průlet. I když by tato funkce mohla být zajímavá, tak si nemyslím, že by mnoho návštěvníků strávilo například hodinu vytvářením klíčových snímků pěkného průletu.

Ohledně průletů byl ještě jeden požadavek. Všichni se na ně vydrželi koukat zhruba 1 vteřinu a už chtěli pouštět další. Chtělo by to tedy umožnit ukončení průletu pomocí dotyku na obrazovku.

Podle jedné z žen bylo neintuitivní to, že pro teleportaci na mapě je potřeba ťuknout jednou, ale v náhledu dvakrát. Jelikož nikdo jiný tento problém nezmiňoval, tak jsem se rozhodl ponechat nynější ovládání. Navíc při testování s odborníky z muzea se na mapě teleportovalo pomocí „dvouťuku“ a na základě zpětné vazby to bylo změněno na jeden ťuk.

Stejná osoba ještě zmiňovala několik dalších návrhů pro úpravu. První z nich byla možnost zvětšit náhled či mapu na celou obrazovku. Tento návrh se mi celkem zalíbil a zvažuji jeho přidání do konečné verze.

Druhý se týkal předpřipravených bodů. Ty by podle ní mohly obsahovat i nějaké historické informace.

Další požadavek byl velice rozumný a nejspíše bych ho také zakomponoval do další verze. Významné body by podle ní měly ukládat i nastavení polohy slunce a při zobrazení bodů by se mělo osvětlení také upravit. Při určitém úhlu světla se totiž mohou lesknout střechy a proto by měly významné body ukazovat pohled s totožným osvětlením jaké měl uživatel při vytváření.

Poslení návrh se týkal mapy. Zde by chtěla okolí modelu doplnit o aktuální mapu Prahy. To by mohlo dodat lepší kontext o poloze.

Další účastník zmínil ještě jednu matoucí věc. Významný bod má totiž 10 hlasů hned po tom, co se vytvoří a pokud mu klesnou pod 0 hlasů, tak se smaže. Podle něj by se měl vytvářet s 0 hlasy a mazat se například na -5 hlasech. S tímto nápadem souhlasím a také ho zakomponuji do další verze.

5.3 Testování výkonu s grafickým rozhraním

Provedl jsem ještě jedno testování na třech počítačích, abych ověřil dopad jak grafického rozhraní, tak hlavně dvou monitorů. Využil jsem znovu program Unreal Insights. U každého jsem zase provedl 3 testování a následně vypočítal průměr. Zde jsem ale neměl předpřipravené průlety, takže jsem prostě provedl většinu akcí, které aplikace umí. Parametry počítačů byly tyto:

Parametry 1. počítače:
- NVIDIA Titan
- 11th Gen Intel Core i9-11700 2,5 GHz
- obě obrazovky: 3840x2160 30 Hz
- 64 GB RAM
- Windows 10

Parametry 2. počítače:
- NVIDIA GeForce RTX 4080
- 10th Gen Intel Core i9-10900X 3,7 GHz
- obě obrazovky: 3840x2160 30 Hz
- 64 GB RAM
- Windows 11

Parametry 3. počítače:
- NVIDIA Quadro RTX 5000
- 10th Gen Intel Core i9-10900X 3,7 GHz
- obě obrazovky: 3840x2160 30 Hz
- 64 GB RAM
- Windows 10

Naměřené hodnoty jsem zapsal do tabulek: 5.17, 5.18 a 5.19. Pro lepší čitelnost jsem také vytvořil graf 5.13.

1. počítač			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	6.55	9.42	8.81
2	7.24	9.34	8.88
3	6.64	9.39	8.86
průměr	6.81	9.38	8.85

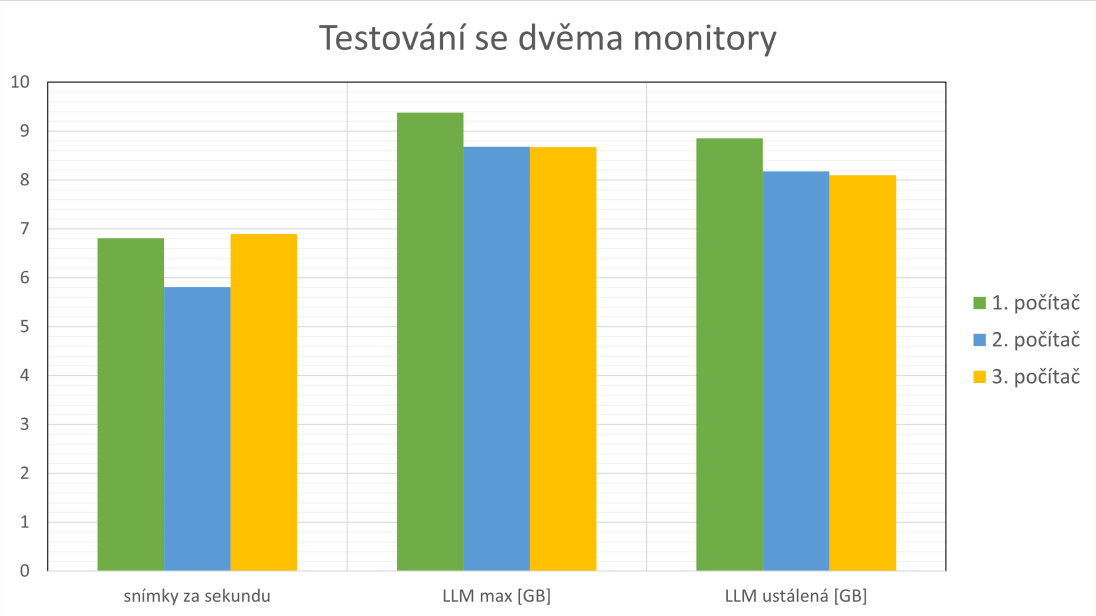
Tabulka 5.17. Naměřené hodnoty na 1. počítači.

2. počítač			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	5.52	8.7	8.2
2	6.29	8.66	8.21
3	5.63	8.68	8.12
průměr	5.81	8.68	8.18

Tabulka 5.18. Naměřené hodnoty na 2. počítači.

3. počítač			
číslo měření	průměrné fps	LLM total maximum [GB]	LLM total ustálená [GB]
1	6.81	8.68	8.11
2	7.04	8.66	8.09
3	6.84	8.67	8.09
průměr	6.9	8.67	8.1

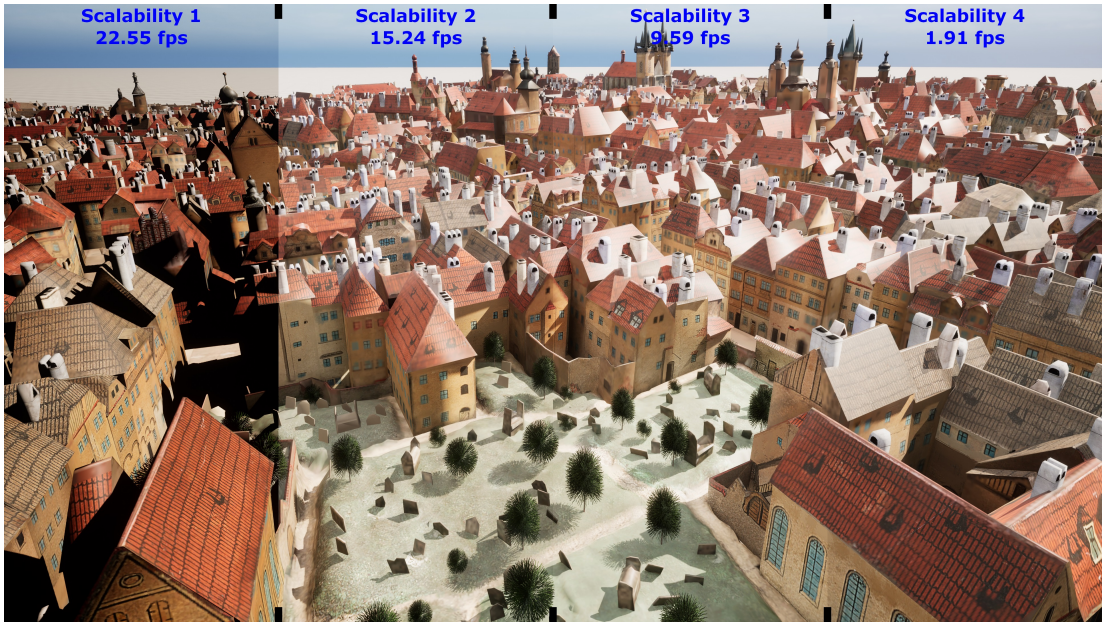
Tabulka 5.19. Naměřené hodnoty na 3. počítači.



Obrázek 5.13. Graf z testování výkonu konečné aplikace.

Jak můžete vidět, tak aplikace má průměrně zhruba 6,5 snímků za sekundu, což je velice málo. Její účel však spočívá v nastavení nějakého pohledu, který se po chvíli stabilizuje. Vysoký počet snímků za sekundu tedy není nutností. Pro nastavení pozice kamery byly dostačující.

Každopádně u LLM total můžeme vypozorovat, že oproti poslednímu buildu aplikace na jeden monitor se hodnota zvýšila zhruba o 1 GB. Tím pádem máme ověřeno, že mezi jednotlivými kamerami dochází ke sdílení textur. Rozdíl spočívá v texturách potřebných pro grafické rozhraní a také v tom, že ne vždy obě kamery vidí stejnou část modelu, tak je potřeba načíst více textur.



Obrázek 5.14. Porovnání různých nastavení kvality.

Celé testování probíhalo s nastavením „Scalability 3“, což je výchozí nastavení kvality obrazu. Hodnotu tohoto parametru lze nastavit od 0 do 4. Jednotlivé úrovně pak znamenají: Low, Medium, High, Epic a Cinematic. Pro lepší představu vlivu tohoto nastavení jsem vytvořil obrázek 5.14. Na něm se nachází 4 pruhy stejného pohledu s rozdílným nastavení „Scalability“. Vynechal jsem „Scalability 0“, protože již „Scalability 1“ projevuje veliký pokles kvality výsledného obrazu. Jak můžete vidět, tak rozdíl mezi „Scalability 2“ a „Scalability 4“ je po vizuální stránce naprosto minimální, avšak s velikým rozdílem ve snímcích za sekundu.

Obrázek 5.15. Nastavení kvality obrazu.

Na obrázku 5.15 je vyobrazeno co konkrétně upravuje parametr „Scalability“. Před konečným odevzdáním aplikace muzeu je potřeba vyzkoušet nastavení všech těchto parametrů zvlášť a vytvořit vlastní nastavení, které zachová kvalitu obrazu, ale zlepší výkon.

Kapitola 6
Závěr

Tuto diplomovou práci lze rozdělit na dvě části. V první jsem se věnoval optimalizaci scény. Druhá popisovala tvorbu grafického rozhraní k jejímu zobrazení a následnému testování. Proto tedy i závěr rozdělím na dvě části.

Optimalizace

V průběhu optimalizace jsem vyzkoušel několik technik. Některé byly vhodné a jiné nikoliv. Zde tedy shrnu co jsem se dověděl.

Pozitivní dopad:

- Sjednocení materiálů, které jsou totožné snížilo zaplnění paměti u grafické karty.
- Instancované materiály mají dopad naprosto minimální. Jsou vhodné spíše pro usnadnění vývoje.
- Textury přeškálované na mocniny dvou sníží velikost na disku díky lepší kompresi.
- Virutální textury jsou vhodné pouze u velikých textur. Jejich využitím se zvýší počet snímků za sekundu a sníží nároky na paměť u grafické karty.
- Mipmapy sice zvýší velikost textury na disku o 33,3%, ale velice pozitivně ovlivní jak počet snímků za sekundu, tak nároky na pameť.
- Pokud se ve scéně nějaký objekt objevuje vícekrát, tak je vhodné ho mít odděleně. Jako příklad mě napadá třeba shodný nábytek v několika budovách. Toto sníží zaplnění paměti na grafické kartě.

Negativní dopad:

- Přestože je Nanite velice užitečným nástrojem, tak v mém případě byl jeho dopad negativní. To bylo nejspíše způsobené typem mého modelu, který byl na počet trojúhelníků méně náročný.
- Použití složitějšího modelu samozřejmě zhoršilo počet snímků za sekundu. I když to je jasné, tak to zde pro jistotu uvádím.

Po úpravě aplikace na dva monitory nároky velice stouply. Oba displeje také měly větší rozlišení, než monitor, který jsem využíval k testování při optimalizaci. Sice byla aplikace použitelná a svému účelu posloužila dobře, avšak nižší snímky za sekundu byly viditelné.

Při rozhovoru s lidmi během testování jsme přišli ještě na pár možností, jak by šla aplikace ještě optimalizovat. První možnost byla zmíněna i v analýze, avšak tehdy jsem se rozhodl ji pro jednoduchost nevyužít. Kdybych totiž upravil nastavení nDisplaye na dva počítače, kde by každý renderoval pouze jeden monitor, tak by se nároky na počítač snížily zhruba na polovinu.

Druhý návrh spočíval v osvětlení pomocí Lumenu. Údajně je velice náročný a jeho vypnutím bych mohl dosáhnout velikého zvýšení počtu snímků za sekundu. Vzhledem k tomu, že hlavním smyslem aplikace je zobrazení papírového modelu Prahy, tak nahrazení osvětlení za méně realistické je dle mého názoru naprosto v pořádku.

Poslední věc, která mě napadá, je převedení projektu na novější verzi Unreal Enginu, kde došlo k vylepšení mnoha funkcí včetně nDisplaye a Unreal Insights.

Teoreticky by se aplikováním těchto úprav mohly snížit nároky aplikace, ale to je potřeba ověřit. Bohužel mi ale nezbyl čas na vyzkoušení těchto zlepšení.

Grafické rozhraní

Na základě zpětné vazby od uživatelů je potřeba ještě trochu upravit ovládání aplikace. Přidání možnosti pohybu v globální rovině XY byla téměř jednohlasně žádaná funkce. Podobně na tom bylo umožnění vypnutí průletů. I když si stěžoval pouze jeden člověk na citlivost ovládání, tak ji i přes to plánuji snížit a to hlavně u přiblížení. Tyto funkce mám v plánu určitě dodělat.

Další dvě úpravy požadovali pouze jednotlivci, ale také je dodělám do další verze. První se týká počtu hlasů u významných míst. Chtěl bych je předělat tak, aby se při vytvoření nastavili na 0 a mazali třeba na -5 hlasech. Druhá změna je také u významných míst, ale týká se osvětlení. Při vytváření by se mělo ukládat i aktuální nastavení polohy slunce. To by se automaticky přepínalo při zobrazení daného místa.

Přidání gesta na otáčení kamery okolo bodu uprostřed náhledu musím důkladně zvážit. V mojí aplikaci jsou již dvě gesta na dva prsty naráz a tudíž by mohlo docházet k omylnému vyhodnocení akce.

Poslední návrhy jsou nutné probrat s odborníky z muzea. Jedná se o možnost zobrazení informací u předpřipravených míst. Další úprava by přidala aktuální mapu Prahy místo čtverečkové sítě. Třetí funkce nejvíce zasahuje do aktuálního rozložení. Ta by měla umožnit přepnutí náhledu či mapy do zobrazení na celou obrazovku.

Po konzultaci s odborník z muzea a aplikování těchto změn bych možná ještě provedl jedno testování s uživateli. Po té by měla být aplikace vhodná pro vystavení v muzeu vedle originálního modelu.

Literatura

[1] Kateřina Bečková. *Svědectví Langweilova modelu Prahy*. první vydání. Praha: Schola ludus-Pragensia, 96. ISBN 80-853-9409-X.

[2] *Langweil*. 2022.
https://www.langweil.cz/.

[3] *Game engine*. 2001-2023. datum přístupu: 2023-08-08.
https://en.wikipedia.org/wiki/Game_engine.

[4] *TOP 5 Herních enginů pro indie vývojáře*. 2019.
https://gamesdev.cz/top-5-hernich-enginu-pro-indie-vyvojare/.

[5] *Nácvik každodenních činností ve VR*. Jan Tošner. Bakalářská práce. ČVUT FEL. Roudnice nad Labem, 2021.
https://dspace.cvut.cz/handle/10467/94658.

[6] *Understanding Nanite - Unreal Engine 5's new virtualized geometry system*. 2021. datum přístupu: 2023-08-12.
https://www.unrealengine.com/en-US/blog/understanding-nanite---unreal-engine-5-s-new-virtualized-geometry-system.

[7] *Exploring Lumen — Unreal Engine 5's dynamic global illumination and reflections system*. 2021. datum přístupu: 2023-08-12.
https://www.unrealengine.com/en-US/blog/exploring-lumen-unreal-engine-5-s-dynamic-global-illumination-and-reflections-system.

[8] *Why you really should be using mipmapping in your graphics applications*. 2019.
https://blog.imaginationtech.com/why-you-really-should-be-using-mipmapping-in-your-graphics-applications/.

[9] David Sedlacek, Jan Burianek a Jiri Zara. 3D Reconstruction Data Set -The Langweil Model of Prague. *International Journal of Heritage in the Digital Era*. 2013, 2 (2), 195-220. DOI 10.1260/2047-4970.2.2.195.

Příloha A
Zadání práce

- 1) Seznamte se s daty Langweilova modelu Prahy. Jmenná konvence, struktura a rozsah dat, použité formáty. 3D Langweilův model dodá vedoucí práce.
- 2) Seznamte se s prostředím pro vývoj her Unreal 5. Soustřeďte se na optimalizační techniky pro zobrazení rozsáhlých a paměťově náročných modelů (např. virtuální textury, instance materiálů, LOD systém, progresivní načítání, paralelní rendering, pixel streaming, nDisplay, ...).
- 3) Navrhněte měření náročnosti zobrazování scény (fps, draw calls, paměť, ...).
- 4) Optimalizujte scénu s modelem s cílem snížit HW náročnost při zachování detailu a s předpokladem pro rendering na dvě obrazovky současně, jednu s 4K rozlišením a druhou FullHD (cca 2K). Postupně zaznamenávejte optimalizační kroky a zhodnoťte jejich důležitost a příspěvek ke snížení náročnosti scény.
- 5) Navrhněte a implementujte aplikaci pro prohlížení modelu na dvou obrazovkách. Jedna bude dotyková (horizontální stůl, 2K rozlišení), druhá bude pouze zobrazovací (4K rozlišení). Dotyková obrazovka bude umožňovat interakci s modelem, není nutné aby se na obou obrazovkách zobrazoval stejný pohled na model. Minimální implementovaná funkcionalita: pohyb po modelu (létání/chození), zapnutí předpřipravených průletů, znázornění významných míst a přepnutí do pohledu na ně, návrh nového významného místa, vytvoření si screenshotu z modelu, změna osvětlení.
- 6) Návrh a implementaci uživatelského rozhraní aplikace konzultujte průběžně s vedoucím práce a vybranými odborníky z Muzea města Prahy.
- 7) Proveďte uživatelské a výkonostní testování výsledné aplikace (s alespoň pěti uživateli).

Příloha B
Zkratky

B.1 Zkratky

LOD	Level of Detail – Úroveň detailu.
KB	kilobyte – Jednotka kapacity počítačové paměti.
MB	megabyte – Jednotka kapacity počítačové paměti.
GB	gigabyte – Jednotka kapacity počítačové paměti.
API	Application Programming Interface – Rozhraní pro interakci většinou ve formě knihovny.
LLM	Low-Level Memory – Paměť přímo na grafické kartě.
POT	Power Of Two – Mocnina dvou.
FPS	Frames per second – Snímky za sekundu.