VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERAKTIVNÍ ZOBRAZENÍ DATOVĚ NÁROČNÉ 3D SCÉNY VE WEBOVÉM PROHLÍŽEČI

INTERACTIVE VISUALIZATION OF LARGE 3D GRAPHICS IN WEB BROWSER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN LUDVÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL ŠPANĚL, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Ludvík Martin**
Program: Informační technologie
Název: **Interaktivní zobrazení datově náročné 3D scény ve webovém prohlížeči**
Interactive Visualization of Large 3D Graphics in Web Browser
Kategorie: Počítačová grafika

Zadání:

1. Seznamte se s možnostmi zobrazení 3D grafiky ve webovém prohlížeči (např. technologie WebGL nebo služby typu game-streaming).
2. Připravte vhodné testovací 3D scény, které budou datově náročné (rozsáhlá 3D scéna s množstvím polygonů nebo velké mračno bodů).
3. Proveďte základní experimenty s vybranými technologiemi a identifikujte problémy při interaktivním zobrazení takových scén v prohlížeči.
4. Navrhněte aplikaci se zobrazením datově náročné 3D scény v okně webového prohlížeče.
5. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručný plakát prezentující vaši práci, její cíle a výsledky.

Literatura:

- Nagaraja and Rocha: Rendering Large Models in the Browser in Real-Time, QCon, 2019 (https://qconnewyork.com/ny2019/presentation/rendering-large-models-browser-real-time).
- Ponchio and Dellepiane: Fast decompression for web-based view-dependent 3D rendering, International Conference on 3D Web Technology, 2015 (http://vcg.isti.cnr.it/Publications/2015/PD15/Ponchio_Compressed.pdf)
- Schuetz, M.: Potree: rendering large point clouds in web browsers, Diploma thesis, 2015 (https://github.com/potree/potree/).
- WebGL specification and resources, Khronos Group, https://www.khronos.org/webgl/

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních tří bodů zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Španěl Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 2. listopadu 2021

Abstrakt

Tato bakalářská práce se zabývá vykreslováním datově náročných scén ve webovém prohlížeči. Jaké možnosti postupu řešení pro vytvoření funkční aplikace jsou dostupné a popis existujících řešení. Konečné vytvoření point cloudu webového prohlížeče pomocí JavascriptAPI WebGPU.

Abstract

This bachelor thesis deals with rendering data-intensive scenes in a web browser. What solution are available for creating a functional application and a description of existing solutions. Final creation of a web browser point cloud using JavascriptAPI WebGPU.

Klíčová slova

Vykreslování, interaktivní zobrazení, Point cloud, WebGPU, webový prohlížeč, Grafika, Type script

Keywords

Rendering, interactive visualization, Point cloud, WebGPU, web browser, Graphics, Type script

Citace

LUDVÍK, Martin. *Interaktivní zobrazení datově náročné 3D scény ve webovém prohlížeči.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Španěl, Ph.D.

Interaktivní zobrazení datově náročné 3D scény ve webovém prohlížeči

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Ludvík
11. května 2022

Poděkování

Děkuji svému vedoucímu Ing. Michalu Španělovy za trpělivost a odbornou pomoc při výběru a vypracování této práce.

Obsah

Kapitola 1

Úvod

Cílem teto práce je seznámení s interaktivním zobrazením datově náročných scén přímo v okně prohlížeče. V dnešní době je spousta různých aplikací přístupných ve webovém prohlížeči. Výjimkou nejsou ani 3D aplikace. Práce přiblíží dostupné možnosti řešení zobrazovaní náročných scén v okně prohlížeče.

Blíže se také seznámíme s pojmem poit cloud, jak probíhá jejich tvorba, možnosti využiti a způsoby vykreslování.

Téma práce jsem si vybral především kvůli osobnímu zájmu o 3D počítačovou grafiku. Zaujala mě problematika vykreslování point cloudů a chtěl jsem zkusit implementovat interaktivní prohlížeč za pomocí nového programovacího rozhraní WebGPU.

V kapitole 2. jsou představeny řešení pro zobrazení 3D scény ve webovém prohlížecí. 3. kapitola se zabývá problematikou point cloudů, jak probíhá jejich tvorba, možnosti využiti a způsoby vykreslování. Ve 4. kapitole je popis návrhu řešení a popis vybraných technologií. 5. kapitola popisuje implementaci aplikace a řešení vzniklých problému. A konečně poslední 6. kapitola hodnotí vzniklé řešení a porovnává ho s existujícími.

Kapitola 2

Interaktivní zobrazení 3D scény ve webovém prohlížeči

V této kapitole budou přiblíženy klíčové pojmy a technologie. Představeny budou možnosti řešení zobrazování scén v prohlížeči a i jejich existující implementace.

V dnešní době se všechny aplikace přesouvají do tzv. cloudových aplikací. V prohlížeči je možno vytvořit dokumenty, tabulky, prezentace nebo editovat videa. Uživatelé očekávají stejnou možnost pro 3D aplikace. Proto je zapotřebí vývoje aplikací schopných vykreslovat 3D scénu v okně prohlížeče.

Existuje více metod pro dosažení požadovaného výsledku. Vykreslování 3D scény muže probíhat bud přímo na zařízením nebo na vzdáleném serveru. Každé řešení má svoje klady a zápory. Tato práce se bude dále zaměřovat na vykreslování scény přímo na zařízením ve webovém prohlížeči. Zde se pokusím přiblížit hlavní rozdíly mezi jednotlivými přístupy.

2.1 Vykreslování na zařízení

Díky vzniku WebGL v roce 2011 bylo umožněno standardizované vykreslování složitých 3D scén ve webovém prohlížeči. WebGL definuje spojení mezi programy v prohlížeči a základními funkcemi grafické karty počítače, aniž by bylo zapotřebí zásuvného modulu [10].

Jelikož je scéna vykreslovaná přímo na zařízení, je třeba přenést data ze serveru na stranu webového klienta. To představuje dvě kritická omezení. Pokud model, který chceme zobrazit má být soukromý, je zde problém s zabezpečením proti odcizení modelů.[3] Druhý problém nastává při vykreslování datově náročných scén. Protože se celá scéna nemůže přenést do zařízení pro svou velikost, vyžaduje řešení implementovat sadu algoritmu a procesu pro výběr části scény která se má právě zobrazit.

Výhody:

- Nízké náklady na provoz aplikace
- Latence pří interakci s prostředím je téměř nulová

Nevýhody:

- Data jsou posílána na zařízení, problém se zabezpečení proti odcizení.
- Výkonost řešení závisí na hardwaru uživatele na straně klienta - nemůžeme zaručit dokonalý zážitek

WebGL Frameworky

Pro urychlení a usnadnění vývoje 3D aplikací ve webovém prohlížeči vzniklo několik javascriptových frameworků. Zde si popíšeme některé z těch nejpoužívanějších.

three.js

Three.js je volně dostupný framwork pro tvorbu různých 3D aplikací v okně prohlížeče. Framework umožňuje vytvářet animované 3D aplikace ve webovém prohlížeči bez nutnosti dokonalého ovládání samotného WebGL. Obsahuje řadu funkcí usnadňující vývoj. Zde je seznam vybraních:

- Scény: přidávat a odebírat objekty za běhu
- Animace: armatury, dopředná kinematika, inverzní kinematika
- Shadery: přístup k plným možnostem OpenGL Shading Language (GLSL)
- Nástroje: kompletní sada 3D matematických funkcí včetně frustum, matrix, quaternion, UVs a dalších
- Export a import: Blender, openCTM, FBX, Max a OBJ
- Podpora: Excelentní dokumentace a spoustu navodů (Knihy Discover three.js, Learn Three.js - Third Edition)

babylon.js

Babylon.js je podobný framework jako three.js Stejně jako three.js umožňuje tvorbu 3D aplikací a videohry pro web [4]. Framework je vyvinutý pomocí jazyku TypeScript, lze jej použít k vytváření 3D aplikací a videoher. Hlavním cílem je usnadnění práce s grafikou v prohlížeči. Babylon je rozdělen do několika modulů:

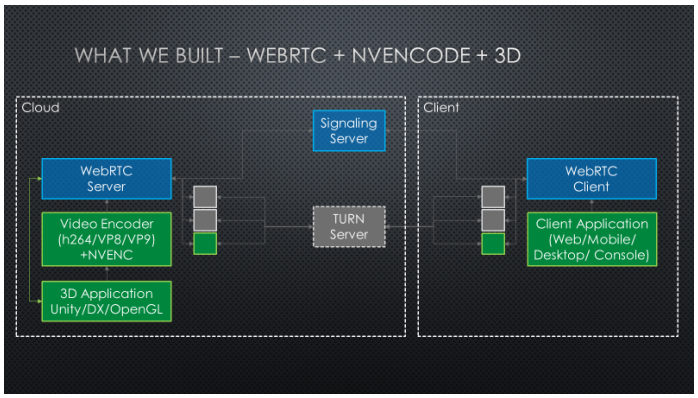
- Playground: Editor pro experimentování s kódem. Kód může být psán v JavaScriptu nebo Typescriptu. Výsledek se okamžitě zobrazí v okně vedle kódu.
- Sandbox: prohlížeč umožňující importovat podporované soubory a zobrazovat je v přímo v prohlížeči.
- Node Material Editor: Vytváření shaderů je komplikovaný proces, který často vyžaduje znalost daného jazyka a shaderů obecně. Tento nástroj je vizuální editor, který vám umožňuje spojovat bloky dohromady a vytvářet shadery. Dá se také použít pro vytvářet procedurální textury a post-proces efekty.
- Gui Editor je vizuální editor pro vytváření grafických uživatelských rozhraní. GUI se tradičně musí konstruovat pomocí kódu, tento editor usnadňuje manipulaci se všemi parametry každého ovládacího prvku. Další výhodou je okamžité zobrazení rozhraní po provedení sestavení .

Existuje mnoho dalších modulů jako Particle Editor, Sprite Editor, Skeleton Viewer a Performance Profiler. Díky větší míře abstrakce oproti three.js, muže při rozsáhlejších scénách klesnout výkonost.

2.2 Pixel Streaming

V dnešní době se stává velmi populární cloud streaming. Zde jsou 3D scény jsou vykresleny na vzdálených serverech, zpracovány a jednotlivé snímky jsou poslány ke klientovy. Architektura tohoto řešení je mnohonásobně složitější než u prvního řešení, pouhý server odesílající data na vyžádání.

Příkladem řečení je přenos přes technologii WebRTC První částí implementace je tzv. signaling server. Ten slouží k navázaní spojení mezi oběma stranami. Následně je navázáno peer-to-peer spojení za pomocí TURN (Traversal Using Relay NAT) serveru. Ten umožňuje spojení zařízení na jiných sítích i přes NAT (Network address translation) . Komunikační rámec WebRTC zajišťuje nejnižší možnou latenci mezi klientem a serverem.



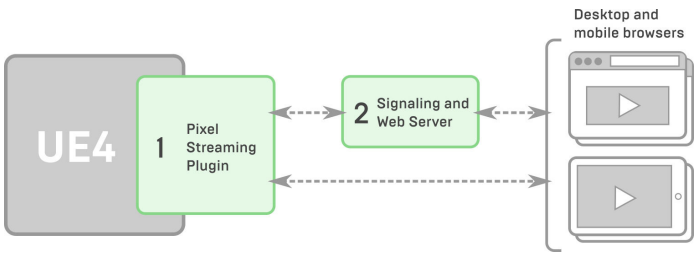
Obrázek 2.1: Diagram příkladu implementace pixel streamingu [1]

Obrázek popisu architekturu open source nástroje 3DSteamingToolkitu. Účelem projektu 3DStreamingToolkit je poskytnout přístup k vývoji 3D serverových aplikací, které streamují snímky v reálném čase do jiných zařízení po síti [1]. Využívá k tomu:

- Zásuvný modul C++ na straně serveru pro vzdálené vykreslování a streamování 3D scén
- Cloudové architektury a infrastruktury umožňující škálovatelné nasazení
- Komprese videa s nulovou latencí pomocí NvPipe/NVEncode
- WebRTC pro zasílání snímků a popřípadě dalších dat jako zvuků

[1] Převzato z: `https://raw.githubusercontent.com/3DStreamingToolkit/3DStreamingToolkit/master/Docs/Images/cloud3dwebrtc.jpg`.

Existují více řešení tohoto problému a jedním z nich je implementace pixel streamingu přímo do herního enginu Unreal. Zde architektura vypadá následovně:



Obrázek 2.2: Diagram implementace pixel streamingu za pomocí Unreal Engine pluginu. [3]

Pixel Streaming Plugin – Tento plugin běží uvnitř Unreal Engine. Zakóduje konečné výsledky každého vykresleného snímku pomocí komprese videa H.264, zabalí tyto snímky videa spolu se zvukem hry do mediálního streamu a odešle tento proud do jednoho nebo více připojených prohlížečů prostřednictvím přímého připojení peer-to-peer.

Signalizační a webový server je zodpovědný za vyjednávání spojení mezi prohlížeči a Pixel Streaming Pluginem. Webový server poskytuje funkcionalitu prohlížeče v prostředím HTML a JavaScript, které přehrává mediální stream.

Existují i další implementace, všechny jsou ale primárně stejné. Vždy jde o přenos snímků od serveru k webovému klientovy nebo aplikaci.

Výhody:

- Zdrojová data zůstávají během přenosu vykreslených snímků stále v bezpečí na serveru
- náročná scéna může být zobrazena i na mobilních nízko výkonových zařízeních
- Uživatel muže okamžitě vidět scénu/ model bez čekání na přenos dat ze serveru

Nevýhody:

- Vyšší náklady na provoz
- Latence muže být vysoká pokud uživatel nemá dobré připojení, nebo je vzdálený od serveru.
- Velký a nepřetržitý datoví tok, nevhodné pro dlouhodobé používání.

2.3 Vykreslování na straně serveru

Hlavní rozdíl oproti pixel streamingu je, že klientovy nejsou posílaný přímo vykreslené snímky. Jedna z možných implementací je například zasílání krychlových map (angl. Cube maps).

Při použití zmíněného přístupu je na stranu klienta přenášena tzv. G-Buffer Cube. G-Buffer je název pro sadu textur vykresleních pomocí grafiky (barevná, hloubková, normálová atd.). V tomto případě je tato textura určená pro mapování na krychli. Klient umístí krychli s texturou do prostoru, tak aby kamera byla uprostřed a provádí vykreslování. Díky vrstvám map krychle G-buffer umožňují klientům implementovat pokročilé 3D operace a integrovat další 3D modely. Data 3D modelu jsou navíc plně uchovávána na serveru, což poskytuje

[3] Převzato z: `https://docs.unrealengine.com/5.0/Images/sharing-and-releasing-projects/pixelstreaming/pixel-streaming-overview/cloud-architecture-424.webp`

vysoký stupeň ochrany. Vykreslování 3D scény je v prováděno na straně serveru, a proto lze implementovat aplikaci i pro nízko výkoná zařízení. Usnadňuje implementaci vykreslovacích služeb na různých platformách.

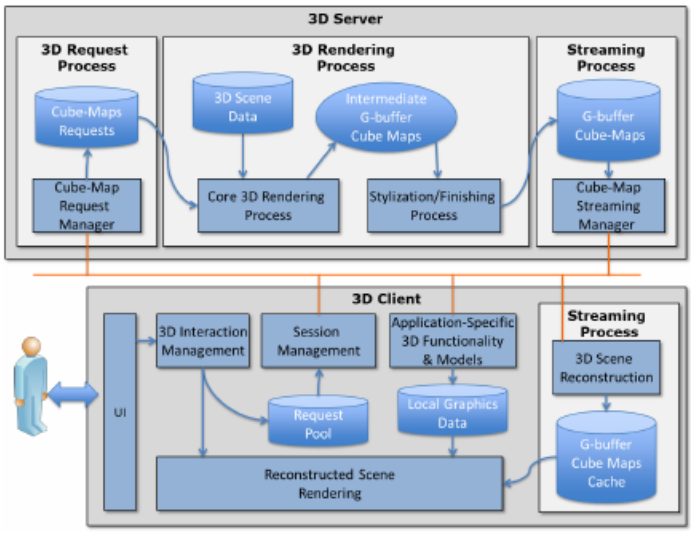
Uživatel interaguje s nejnovější mapou krychle, která je konzistentní. Pokud je změněna pozice kamery, je zaslán požadavek serveru pro vygenerování nové textury krychle a aktuální je označena jako zastaralá. Díky tomu se mohou začít vyskytovat artefakty při změně krychle. Mezi artefakty patří rozmazání obrazu a nesprávné zobrazení 3D objektu.[8].

Výhody:

- Zdrojová data zůstávají během přenosu vykreslených krychlí v bezpečí na serveru
- Náročná scéna může být zobrazena i na mobilních nízko výkonových zařízeních

Nevýhody:

- Vyšší náklady na provoz
- Latence muže být vysoká pokud uživatel nemá dobré připojení, nebo je vzdálený od serveru.



Obrázek 2.3: Architektura aplikace vykreslování cube maps na straně serveru [8].

Kapitola 3

Point cloud

Point cloud neboli mračno bodů, je označení pro 3D modely, které se skládají z mnoha nezávislých bodů. Každý bod má svoji pozici v prostoru a další případné vlastnosti.

3.1 Využiti

Využiti point cloudů je poměrně široký. Primárním účelem point cloudů je vytvořit 3D model reprezentující reálný objekt. Body lze zobrazovat pomocí prohlížečů jako 3D model, ale často jsou data bodů nejprve převedena na polygonovou síť, protože většina 3D softwarových programů pracuje s polygony. Zde si přiblížíme jen některé užitečné využití.

Plán prostorového uspořádání

Slouží k vytvoření plánu budovy nebo prostoru bez složitého měření. Existují dva způsoby zpracovaní vytvořeného modelu. Point cloud si můžete prohlížet z horního ortografického pohledu a plánovat půdorys ručně pomocí CAD nebo softwaru pro plánování podlaží. Nebo můžete použít software s umělou inteligencí k automatickému vygenerování půdorysu.

Informační model budovy

Vzhledem k tomu, že mračno bodů přesně a komplexně odráží aktuální stav budovy , znamená to, že nabízí všechna nezpracovaná prostorová data, která jsou potřeba k vytvoření nového informačního modelu budovy nebo aktualizaci stávajícího modelu. To umožní porovnat stav budovy v reálném čase, abyste mohli zkontrolovat chyby, konflikty, a střety[11].

Sledování postupu výstavby

Pokud během stavby pravidelně pořídíte point cloud, můžete tato data použít ke sledování toho, jaká práce je dokončena, kdy je dokončena a kde. Existují automatické nástroje k poskytování podrobnějších aktualizací, například software, který automaticky analyzuje point cloudu za účelem rozpoznání konkrétních objektů. Lze dokonce sledovat práce s plány a rozpočty ze kterých software dokáže vypočítat dosavadní náklady.

Správa dopravní infrastruktury

Pomocí point cloudů lze snadno spravovat rozsáhlé sítě infrastruktury. Okamžitý přehled o požadavcích na zlepšení sítě. Používá se pro automatickou, přesnou a prediktivní údržbu infrastrukturních sítí jako nezbytná součást digitalizace.

Klasifikace

Pro zlepšení přehlednosti point cloudových dat lze použít klasifikaci. Některé systém umí automaticky přidělit body kategoriím, jako jsou budovy, vegetace a terén. Získáme tak nové poznatky o datech, jako jsou topologie střech, vegetace, chování terénu a konfigurace měst.

Skenovaní objektů

Mezi využiti patří i dokumentace objektů a skenování objektů za účelem tzv. triangulace. Triangulace je proces převedení seznamu znáních bodů na sadu trojúhelníků. Využívá se teto techniky za účelem vytvoření realistického modelu. Ten muže být použit pro vytvoření CGI (Computer-generated imagery) pro film nebo statickou scénu. V dnešní době se tento proces začíná využívat i pro tvorbu her. Proces triangulace probíhá za pomocí specializovaného programu. Populární je například aplikace Pointfuse nebo open sorce program Meshroom.

3.2 Vytváření poit cloudů

Existují dvě možností pro vytváření point cloudů. Pro rychlé a přesné skenování objektů je požívaná technologie LiDAR[1]. LiDAR měří vzdálenost objektu za pomocí měření času za který se paprsek laseru odrazí od objektu. Samotný LiDAR nemá schopnost zaznamenat barvu od odraženého povrchu . Pokročilé skenery mohou být vybavené navíc třemi laserovými diodami a foto-diodami (RGB), které zachycují intenzitu barev vracející se od cíle.

Zařízení LiDAR mohou být umístěny na letadlo, dron a existují i řešení určená pro ruční ovládání. Firma GeoSlam nabízí právě tyto ruční skanery. Některé point cloudy určené pro testování výkonu aplikace v kapitole 6.1 jsou pořízené těmito zařízeními.

Další metodou pro vytváření point cloudů je tzv. fotogrammetrie. Zde jsou body vytvářeny ze sady fotografií pořízených z různých úhlů. Metoda využívá fotografie pořízené z různých úhlů objektu za účelem generování 3d modelu. Pro převedení fotografií do 3d modelu, je zapotřebí specializovaného softwaru. Tato metoda je relativně nízko nákladová a výsledky jsou poměrně přesné. Na druhou stranu je tato metoda vytváření bodů časově mnohem náročnější na sběr a následné zpracování dat.

Nástroje na zpracování point cloudů

Point cloudy nemusí být hned po vytvoření použitelné. Je nutné provést úpravy nebo vytvořit klasifikaci bodů. Při pořizování můžou vzniknout nevyžádané body, kterým se říká šum. Takové body je možno odstranit nebo je klasifikovat jako zmíněný šum. Existuje hromada programů pro zpracování bodů, některé se specializují na specifickou funkcionalitu.

[1] LiDAR - Light Detection and Ranging (Detekce a měření světla)

Mezi běžné operace patří:

- Převedení point cloudů na jiné datové typy. - Můžete ho například převést na digitální výškový model terénu a sdílet jej jako rastr. Nebo již zmíněná triangulace.
- Přidání barvy - Body jsou často potřeba obarvit. Běžný způsob pro dosažení barevných bodů, je převzít hodnoty RGB z nějaké rastrové fotky . Existuje mnoho dalších způsobů, jak nastavit barvu, třeba na základě výpočtů nebo z externích databázi.
- Zmenšení velikosti - Cílem je snížení celkového objemu dat snížením počtu bodů.

CloudCompare

Je volně dostupný software pro zpracování 3D mračna bodů . Původně byl navržen k provádění srovnání mezi dvěma hustými 3D mračny bodů nebo mezi mračnem bodů a trojúhelníkovou sítí. Vnitřně si vytváří octree strukturo pro jednoduší manipulaci a zpracování. Postupně byl rozšířen až se z něj stal software pro zpracování mračna bodů, včetně mnoha pokročilých algoritmů. registrace, převzorkování, zpracování barevných/normálních/skalárních polí, výpočet statistik, správa senzorů, interaktivní nebo automatická segmentace atd.).

Vision Lidar

Výkoný komerční software, který dokáže provést cokoliv si člověk umíní. Mezi funkce softwaru patří:

- Pokročilý proces klasifikace využívající unikátní technologii Deep Learning GPU
- Vytvoření plánku půdorysu budovy z klasifikace.
- Detekce dopravního značení pomocí intenzity.
- Vybarvení point cloudů za pomocí fotek z 360° kamery
- Měření výšky stromu, průměr kmene, povrch koruny a průměr koruny z výšky hrudníku.

3.3 Vykreslovaní point cloudů

Vykreslování point cloudů je náročný proces z důvodu, že velikost takových souborů muže dosahovat desítky až stovky gigabajtů. Proto většina aplikací pro interaktivní zobrazení využívá algoritmů, které zasílají data po částech. Existuje mnoho metod jak rozdělit velké point cloudy na menší podoblasti. Tato operace se provádí nezávisle na zobrazování, z důvodu úspory výkonu a času. [18]

Vždy je vytvářená nějaká struktura jako například octree, kd-tree nebo quadtree. Struktura může být aditivní nebo nahrazující. Rozdíl je v ukládání bodů v různých úrovních. Aditivní schéma sloučí více úrovní struktury k zobrazení všech bodů. Nahrazující zobrazuje vždy jen určitou úrovni struktury, která obsahuje všechny body z nadřazených úrovní. Při využití nahrazujícího schematu jsou tak data uložena opakovaně v každé části struktury.

Nejednoduší vykreslovaní point cloudů je reprezentace bodů jako jednoho bodů na obrazovce. Je to rozhodně nejrychlejší řešení ale při přiblížení na část scény je při nízkém počtu bodů nemožné rozeznat na co se díváme. Z toho důvodu jsou body reprezentovány větším geometrickým útvarem. Často to jsou čtverce nebo kruhy. Ale i toto řešení má své problémy. Velikost daných geometrických tvarů musí být dynamicky upravována. Kdyby byly všechny body vykreslovány stejnou velikostí začnou se při velkém množství překrývat a budou ztrácet detaily.

Pokud nám nezáleží na plynulosti zobrazení, ale chceme co nejlepší vizuální reprezentaci dat lze využít pokročilých metod. Jedna z metod je surface splatting [19]. Body jsou zpracovaný a uloženy jako jednotlivé pixely. Pro všechny pixely jsou vypočítaný váhy pro body v této vzdálenosti . Pixely mají vyšší váhu čím blíže jsou ke středu bodu. Váhy jednotlivých bodů jsou pak sečteny a součet je normalizovány.

Aby bylo zajištěno, že body ze kterých se vypočítává váha jsou pouze body, které patří ke stejnému povrchu, je použito hodnoty hloubky. Jsou akceptovány jen body v určitém rozmezí. Vznikne tak plynulí přechod mezi sousedícími body a jejich barvami. Toto řešení je vhodnější pro neinteraktivní zobrazovače pro svoje větší nároky. O úroveň složitější je provádět triangulaci dat na zařízení v průběhu vizualizace. Data musí být uložena a posílaná tak aby mohli být okamžitě převáděná na trojúhelníky.

3.4 Dostupná řešení pro point cloud vykreslovaní

Existuje několik řešení, které umožňuji vykreslovaní point cloudů ve webovém prohlížeči.[14] Všechny řešení jsou postaveny na WebGl API.

Potree

Postaven na základě zmíněné webgl knihovny Tree.js. Umožňuje zobrazení dat ze serveru po převedený do potree formátu [16]. Díky implementaci jako zasuvný modul pro Tree.js framework lze prohlížeč snadno rozšířit tak, aby zobrazoval další geometrické modely spolu s point cloudy.

Potree nabízí nejvíce nástrojů pro práci a měření point cloudů. Nástroje jako měření délky, objemu, vytvoření výškového profilu. Pro vykreslování bodů jsou k dispozici dva režimy: standart a high quality. Režim high quality pracuje s již zmíněným surface splattingem a míchá barvy pro vznik plynulého přechodu mezi body.

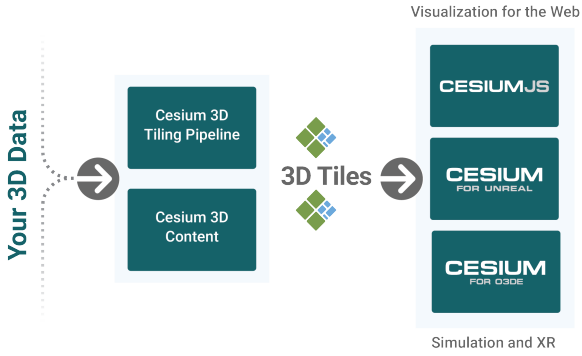
Ve spoluprací s Hobu, Inc vznikla verze kompatibilní se serverovou datovou strukturou Entwine point tile 4.5.

Cesium

Cesium je platforma pro vytváření aplikací jako jsou mapy a digitální zeměkoule ale umožňuje i vykreslovaní point cloudů. Je založena na tzv. 3D Tiles. 3D Tiles je otevřená specifikace pro sdílení, vizualizaci, spojování a interakci s masivním heterogenním 3D geoprostorovým obsahem . Platforma má několik aplikací: Cesium ion, CesiumJS, Cesium for Unreal a Cesium for O3DE.

Cesium ion nabízí 3D obsah v cloudu spravovaný společností jako jsou point cloudy, fotogrammetrií nebo je také možno nahrát vlastní 3D data . Tato služba pak slouží jako

serverové řešení pro tvorbu aplikací s potřebou geologických dat . Umožňuje tak rychlé řešení pro streamování obsahu přes internet pomocí zmíněných 3D Tiles. [5]



Obrázek 3.1: Ekosystém platformy Cesium

CesiumJS je volně dostupná JavaScriptová knihovna pro tvorbu 3D reprezentací planet a map a snadným použitím. Vývojáři napříč průmyslovými odvětvími, od letectví až po digitalizaci měst, používají CesiumJS k vytváření interaktivních webových aplikací. Jak již bylo zmíněno umožňuje i práci s point cloudy. Hlavní výhoda oproti ostatní řešením je jednoduchá implementace umístění point cloudů na 3D reprezentaci zeměkoule.

Cesium for Unreal a Cesium for O3DE jsou oba zásuvné moduly do existující herních enginů. Umožňuji přístup k 3D obsahu včetně terénu země, reprezentaci zeměkoule , 3D měst a fotogrammetrie.

plas.io

Toto řešení umožňuje načíst malé point cloudy přímo ze souborového systému, nebo využívá Greyhound serveru pro načítaní dat větších scén. Greyhound byl server pro streamování point cloudů. V tuto chvíli už není dále vyvíjen. Byl nahrazen serverovou strukturou o které budeme mluvit později.

Poskytuje funkční implementaci formátu ASPRS LAS a dokáže zpracova i data komprimovaná LASzip pomocí modulu LASzip NaCl [2]. Nevýhoda Plasio je dostupnost v současnosti pouze pro webové prohlížeče na bázi Chromium.

[2]Native Client byl nástroj pro efektivní a bezpečné spouštění zkompilovaného kódu C a C++ v prohlížeči, nezávisle na operačním systému uživatele.

Kapitola 4

Návrch řešení

Kapitola zabývající se návrhem svého řešení prohlížeče point cloudů.

4.1 WebGPU

WebGPU je nové webové rozhraní, které odhaluje možnosti moderní počítačové grafiky, konkrétně Direct3D 12, Metal a Vulkan, pro provádění renderovacích a výpočetních operací na grafickém procesoru(GPU) [2]. Mezi hlavní výhody oproti WebGL patří přístup k většímu počtu instrukcí . Další výhodou je volnost specifikace. Při vytváření specifikace se nemusí držet již existujícího řešení jak v případě WebGL. Implementace je navržena pro podporu multy procesové architektury prohlížeče.

WebGPU musí mýt přistup k nízkým systémovým procesům. Jelikož je časté že aplikace v prohlížeči čekají na přistup k systémovým zdrojům a mohou selhat, je proces pracující s ovladačem grafické karty umístěn do odděleného kontejneru. Tento proces pak komunikuje s prohlížečem pomocí asynchronní mezi procesorové komunikací.

Pro dosažení maximální bezpečnosti a rychlosti procesy ovladače GPU žijí pouze v procesu GPU, včetně velkých alokací (jako jsou vyrovnávací paměti a textury). Procesor používá tyto typy WebGPU (GPUBuffer, GPUTexture, GPURenderPipeline, ...) většinou jen jako ukazatel na objekty, které se vyskytují v procesu GPU.

Z toho plyne, že paměť CPU a GPU jsou v prostředí WebGPU oddělené. Například objekt GPUBuffer, který popisuje buffer nacházející se v paměti grafické karty může využívat například 150 bajtů paměti CPU, ale mít alokovanou paměť o velikosti 1 GB na GPU.

4.1.1 Práce s Javascript API

Algoritmus 1: Inicializace WebGPU v prostředí javascript

```
1: const webGpu = navigator.gpu;
2: if (!webGpu) then
3:     return goToFallback();
4: end if
5: adapter = await webGpu.requestAdapter({powerPreference: "high-performance"});
6: device = await adapter.requestDevice();
```

Adapter (GPUAdapter) je objekt, který slouží jako identifikuje konkrétní implementaci WebGPU v systému. Systém muže mít dvě různé implementace hardwaru (např. integrované a dedikované GPU). Pro získání adaptéru aplikace zavolá navigator.gpu.requestAdapter().

Při volání je možno předat parametry které mohou ovlivnit, jaký adaptér bude vybrán. Například parametr powerPreference určuje preferenci spotřeby nebo forceSoftware k vynucení softwarové implementace grafické karty .

Device (GPUDevice) představuje logické připojení k adaptéru WebGPU. Vytváří jednoduchou reprezentaci zařízení a zprostředkuje jediné spojení tak, že se může pomocí něho programovat , jako by byl program jediným vlastníkem adaptéru. V rámci tohoto zapouzdření je zařízení vlastníkem všech z něj vytvořených objektů, jako WebGPU textury, buffery atd. , které lze uvolnit, kdykoli dojde k odpojení zařízení. Různé elementy na jedné webové stránce mohou mít každá své vlastní zařízení WebGPU.

Práce s pamětí (GPUBuffer)

GPUBuffer představuje alokovanou paměť použitelnou pro operace grafické karty. K této paměti lze přistupovat lineárně, protože rozložení paměti je v sekvencí zasebou. Při používání WebGPU, aplikace potřebují přenášet data z JavaScriptu do GPUBuffer velmi často a ve velkém množství. Příkladem potřebných dat jsou síťová data, parametry scény a funkcí atd. Existují dva způsoby předání dat. Funkce GPUQueue.writeBuffer a mapováním do paměti pro zápis.

Aplikace také často potřebují přenášet data z GPU do Javascriptu . Mezi možné data patřá snímky obrazovky, statistiky z výpočtů, výsledky simulace atd. Tento přenos lze provést mapováním vyrovnávací paměti pro čtení.

Pipeline

Pipeline popisuje práci, která má být provedena na GPU, jako sekvenci příkazu. Příkazy draw nebo dispatch spouštějí tuto sekvenci příkazu s definovanými vstupy, vystupy a připojenými zdroji.

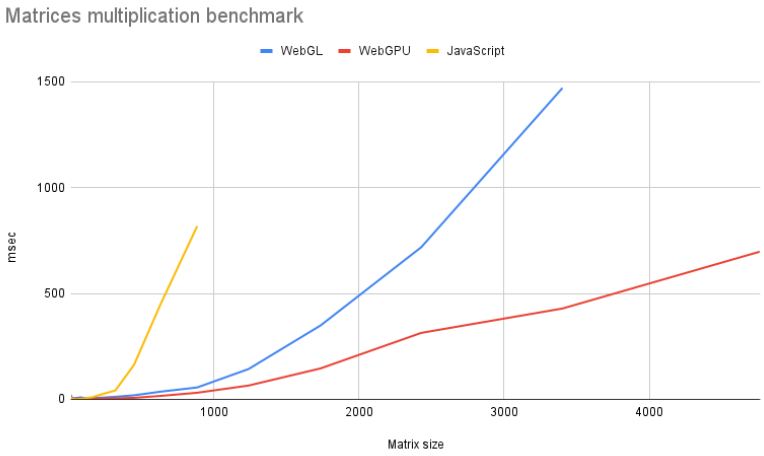
Existují dva druhy pipeliny: GPUComputePipeline a GPURenderPipeline. Příkaz dispatch používá GPUComputePipeline ke spuštění fáze compute shaderu. Draw používá GPURenderPipeline ke spuštění vícefázového procesu vertex a fragment shaderu.

Z důvodů,že většina existujících řešení je na bázi WebGl, a WebGPU by mělo nabízet větší výkon, rozhodl jsem se při implementaci aplikace využít toto API.

WebGPU může být obtížnější než jiná grafická programová rozhraní . Její struktura je ale více v souladu s designem moderních grafických karet. Díky těmto vlastnostem by neměla vést pouze k rychlejším aplikacím, ale také aplikacím.

4.1.2 Porovnání s WebGL

WebGPU má být nástupce WebGL. Ve srovnání s WebGL slibuje WebGPU lepší výkon a lepší kompatibilitu s moderním hardwarem, ale nejznámější funkcí WebGPU je speciální programové rozhraní pro provádění výpočtů na GPU.



Obrázek 4.1: Benchmark násobení matic. [1]

Z grafu můžeme dokázat, že výpočetní shadery WebGPU jsou v praxi přibližně 3,5x rychlejší než výpočty WebGL. Přičemž mají výrazně vyšší limity na množství zpracovávaných dat a také neblokují hlavní vlákno. To umožňuje lepší interakci s aplikací při provádění výpočtu. Umožnilo by to provádění složitých výpočtů v reálném čase. Využít této výhody lze pro vytvoření aplikací na úpravy videa a zvuku, fyzikální simulace v reálném čase nebo strojové učení, přímo v okně prohlížeče.

4.2 WGSL

WGSL(WebGPU Shader Language) je jazyk pro popis shaderů určených pro WebGPU [7]. Tento jazyk je jediný přímo podporovaným shader jazykem WebGPU.

Životní cyklus shaderu. Cyklus programu WGSL a shaderů, obsahuje čtyři klíčové události. První dvě jsou metody rozhraní WebGPU používané k přípravě programu na spuštění WGSL . Poslední dva jsou začátkem a koncem provádění shaderu.

- Vytvoření modulu shaderu - K tomu dochází, když je volána metoda CreateShaderModule(). Metodě je předán zdrojový text pro daný shader v jazyce WGSL.
- Vytvoření Pipeline - Událost je vyvolána metodo CreateRenderPipeline(). Muže být přidán jeden nebo více dříve vytvořených modulů shaderu,spolu s dalšími konfiguračními informacemi.
- Spuštění průchodu shaderu - GPU je vydán příkaz draw nebo dispatch, zahájí se provádění dříve definované pipeliny a zavolá se funkce vstupního bodu.

[1] Převzatý graf, dostupný z: `http://pixelscommander.com/wp-content/uploads/2021/10/Matrices-multiplication-benchmark-1.png`

- Ukončení průchodu shaderu - při dokončení veškerá práce v shaderu jsou provedeny tyto kroky : všechny přístupy ke zdrojům jsou ztraceny a pokud existují výstupy z aktuální části shaderu, jsou předávány do přes pipelinu do dalšího průchodu.

Dříve zmíněný vstupní bod je funkce která provádí práci pro konkrétní fázi shaderu. WGSL definuje tři fáze shaderu: compute, vertex, fragment.

Fáze vertex mapuje vstupní atributy pro jeden bod na výstupní atributy pro bod. Body jsou převedeny do grafických primitiv (trojúhelník, bod), které jsou poté rastrovány za účelem vytvoření fragmentů.

Fragment shader zpracuje každý fragment, případně vytvoří výstup fragmentu. Výstupem fragmentu jsou aktualizované parametry bodu. Mezi ně patří barva a hloubka. Každá fáze shaderu má svou vlastní sadu funkcí a omezení.

4.3 Server

Cílem této práce bylo vytvoření webové aplikace pro zobrazení point cloudů. Proto serverová část řešení byla převážně vytvořena z existujících částí. Server bude sloužit jen pro ukládání dat a zasílaní směrem k webovému prohlížeči. Veškerá logika je řešena přímo v prohlížeči .

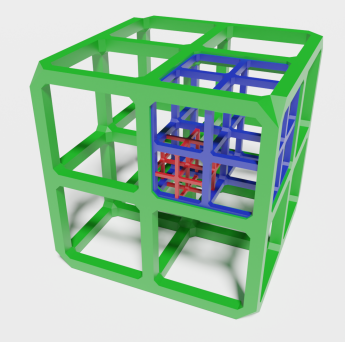
Server pro lokální požití lze vytvořit za pomocí volně dostupného programu apache2. Při nutnosti přístupu k datům přes internet je zapotřebí serveru s SSL certifikátem. Nebo lze data umístit na cloudové úložiště od společností Goole nebo Amazon.

4.4 Serverová struktura dat

Jak bylo zmíněno v kapitole Vykreslovaní point cloudů 3.3 je zapotřebí aby data určená pro zasílání byly uspořádané do nějaké rychlé a přehledné struktury. Zde si přiblížíme řešení, které bude využito pro implementaci vlastního řešení.

4.4.1 Octree

Octree je stromová datová struktura, ve které má uzel vždy osm potomků. Tato struktura je široce využívaná v 3d grafice. Hlavním důvodem pro její popularitu je skutečnost možnosti rozdělení krychlového prostoru na osm stejně velkých podprostorů jednoduchým a rychlím způsobem.



Obrázek 4.2: Ukázka vnitřní struktury z Octree.

4.5 Entwine point tile

Reprezentace dat na serveru je nedílným faktorem v rychlosti načítaní scény. Entwine point tile (EPT) je datová struktura založená na octree, pro ukládání a stremovaní point cloudových dat. Struktura se skládá z hlavního souboru ept.json, složek ept-hierarchy, ept-data. Soubor ept.json obsahuje informace o modelu/scéně. Mezi důležité parametry patří:

bounding box

Bouding box[2] je uložen ve formátu [xmin, ymin, zmin, xmax, ymax, zmax] popisující prostorové hranice octree struktury .

schéma uložených dat

Pole objektů, které představují indexované pole – každé pole musí mít název, typ a velikost. Navíc pole X, Y, Z pro pozice bodů v prostoru můžou obsahovat offset a scale. , dataType - typ souboru ve složce ept-data , počet bodů - celkový počet bodů uložených v schématu.

srs

Objekt popisující prostorový referenční systém pro tuto strukturu. Určení pozyce pomocí horizontálního a vertikálního kódu souřadnicového systému, který je dán autorita (obvykle "EPSG") .

Složka ept-hierarchy obsahuje minimálně soubor 0-0-0-0.json, ve kterém je seznam existují uzlů s jménem souboru a kolik bodů obsahují. Jméno souboru je textový řetězec ve formátu D-X-Y-Z.

- D - Hloubka (Depth) zanoření do octree struktury.
- X, Y, Z - Pozice uzlů v dané hloubce v rámci jednotlivých os.

Pokud hodnota počtu bodů je -1, je informace o daném uzlu a všech jeho potomků je uložen v souboru D-X-Y-Z.json . EPT je aditivní schéma, to znamená že body nejsou duplicitně uloženy v hlubších uzlech. Pokud tedy chceme zobrazit všechny body v dané oblasti, musíme zkombinovat všechny boby z uzlu s nejvyšším rozlišením se všemi překrývajícími se uzly s nižším rozlišením.

Data jsou uložena ve složce ept-data. Názvy souborů odpovídají těm ze složky ept-hierarchy. Data jsou uložena v jednom ze tří formátů. LAS/LAZ, binární nebo zstandard.

4.5.1 Generace struktury

Pro generaci struktury slouží program entwine. Je dostupný jako docker image [3] nebo Conda environment [4]. Program obsahuje dva dílčí příkazy *build* a *convert*. Příkaz build slouží k vytvoření struktury ze souboru. Soubor může být formátu, který je podporován PDAL knihovnou [5]. Při procesu generace provádí program vzorkování hustoty, intenzity a

[2] Bounding box - imaginární krychle sloužící pro určení hranic objektu.

[3] Docker image je šablona, která obsahuje sadu pokynů pro vytvoření kontejneru, který lze spustit na platformě Docker.

[4] Conda environment je adresář, který obsahuje kolekci balíčků kihoven a programů.

[5] Point Data Abstraction Library - C++ knihovna pro překlad a manipulaci s daty point cloudů.

dalších parametrů pro správné rozdělení do uzlů. Navíc kontroluje i vizuální kvalitu modelu. Umožňuje převod z několika různých souborů. Interně musí vyřešit různé rozsahy hodnot intensity [13].

Convert je využíván pro převod EPT struktury do jiného formátu. V současnosti je jediná možná konverze, a to do již zmíněného formátu Cesium 3D Tiles. K vytvoření 3D Tiles je potřeba nejprve převést soubor/soubory do EPT reprezentace, až poté je možná konverze.

4.6 LAS/LAZ

Modely point cloudů můžou být uloženy v mnoha formátech. Mezi široce používané i populárni patří LAS formát. Je používán hlavně při tvorbě point cloud za pomocí LiDAR technologie. LAS[15] (LASer) je binární otevřený formát, vytvořený za účelem ukládání všech dat zjískaných při použití LiDARu . Pozice bodu je uložena jako 32/64 bitový integer. Mezi další parametry formátu patří:

- Red, Green, Blue - uloženy jako 8/16 bitový integer. Dohromady reprezentují barvu bodu
- Intesity - 16 bitové přirozené číslo, reprezentující velikost návratového pulzu
- Classification - 8 bitové přirozené číslo, určující co určitý reprezentuje.
- Return Number - návratové číslo pulzu pro daný výstupní pulz.
- Number of Returns - počet návratů z jednoho pulzu laseru.
- GPS Time - 64 bitová hodnota obsahující časové razítko, určující čas ve který byl bod pořízen.

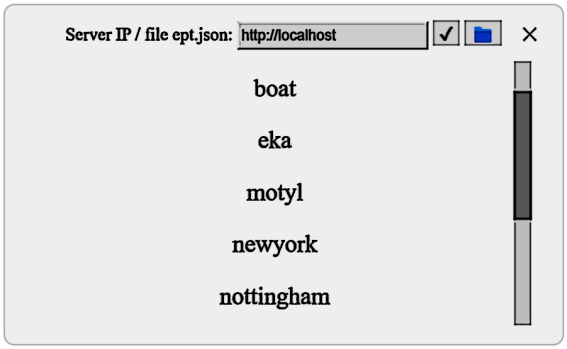
Pro zmenšení přenesených dat jsem zvolil komprimovanou variantu LAS formátu - LASzip (LAZ)[12]. Tato varianta byla vytvořena Martinem Isenburgem .

Velikost souboru LAZ jsou v porovnání s LAS o sedmdesát až devadesát osm procent menší. Například LAS souborů o velikosti 5,3 terabajtů , které National Land Survey of Finland poskytuje na svých veřejných serverech, je komprimovaná do pouhých 0,8 terabajtů.

Každá výhoda musí být vyvážená jednou nevýhodou. Díky kompresi je práce s LAZ soubory mnohem náročnější na procesor zařízení z důvodu nutnosti dekomprese. Záleží tedy čeho se snažíme dosáhnout. Úspory přenesených dat nebo snížení výpočetní náročnosti aplikace na procesor zařízení.

4.7 Uživatelské prostředí

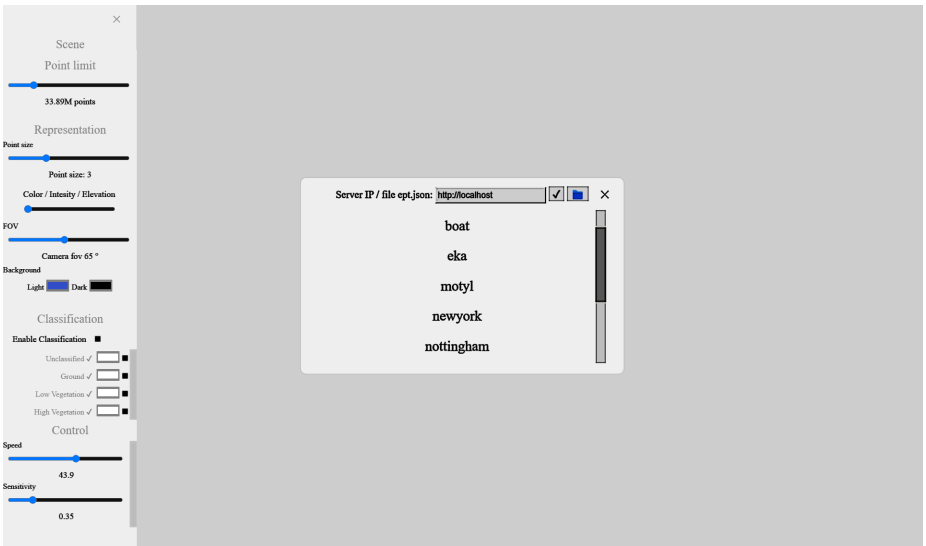
Cílem uživatelského prostředí je vytvořit jednoduché a neintruzivní načítaní a úpravy scény. Při otevření stránky s aplikací je uživateli okamžitě umožněno načtení point cloudu. Jsou na výběr tří možností. Otevřít lokální složku dostupnou na zařízení s EPT strukturou. Zadáním url souboru ept.json uloženého na serveru. Nebo poslední možnost zadání adresy serveru s jednoduchým php skriptem, který vrátí všechny scény dostupné na něm.



Obrázek 4.3: Návrh vzhledu okna pro výběr scény

Všechny možnosti a nastavení scény jsou dostupná v levém vysouvacím panelu. Ten se vysouvá kliknutím tlačítka v horním levém rohu. Panel obsahuje pět položek. První je tlačítko slouží k otevření okna s výběrem scény. Další čtyři položky po kliknutí rozbalí odpovídající nábytku. Point limit slouží k nastavení maximálnímu počtu bodů. Maximální hodnota posuvníku je nastaven podle scény omezená na sto milionů. Representation položka skrývá nastavení velikosti bodu, výběr zdroj barvy bodů, zorné pole kamery a gradient barvy pozadí. Posuvník k výběru zdroje barvy má tři pozice. Pro druhou a třetí je možno nastavit rozsah hodnot. Gradient je tvořen ze dvou barev. Jasnější barva je vždy směrem k nebe a tmavší gradientem k zemi.

Classification sekce se stará o nastavení vykreslení bodů podle jejich hodnoty klasifikace. Uživatel muže přepínat jedním tlačítkem mezi klasifikovaným vykreslovaní a normálním. Fajfka za názvem filtru značí jestli se daný typ bodů má vykreslovat. Přepnutí probíhá kliknutím na název. Výběr barvy pro určitý typ bodů lze pomocí standardního html color pickeru. Pro vypnutí zvolené barvy slouží zaškrtávací políčko za vybranou barvou.



Obrázek 4.4: Návrh vzhledu celé aplikace

Poslední část s názvem control skrývá ovládání kamery. Uživatel má možnost změnit rychlost pohybu kamery v prostoru. Nebo nastavit citlivost míši.

Poslední část uživatelského prostředí je ukazatel počtu snímků za minutu. Ten je umístěn uprostřed v horní části obrazovky.

Kapitola 5

Implementace

Tato kapitola se zabývá vlastni implementaci interaktivního prohlížeče point cloudů. Při implementaci byl převážně využit programovací jazyk typescript. Dalším již zmíněným jazykem pro popis shaderu je WGSL. Jelikož WebGPU není ještě standardizováno, není možno zaručit funkčnost v novějších vydání prohlížeče. Řešení dostupné na DVD je otestováno a funkční ve webovém prohlížeči Chrome Canary [1].

5.1 Rozdělení do modulů

Pro zlepšení přehlednosti je aplikace rozdělena na několik modulů.

App

Modul app se stará o praci s uživatelským rozhraním. Připravuje scénu k načtení a zahajuje proces načítaní.

Octree

Tento modul obsahuje implementaci octree struktury pro účely této aplikace. Obsahuje třídu Octree, která slouží k vytváření a ukládání struktury. Pro tyto účely obsahuje sadu metod. Uzel obsahuje mnoho parametrů, některé jsou při vytvoření následně nastaveny. Parametry jako: počet bodů, název odpovídajícího souboru, stav načtení, bounding box, střed, okazy na existující GPUBuffery, objem a samozřejmě proměnné pro svoje potomky.

addNode(node) parametr metody předá funkci objekt reprezentující uzel načtený ze složky ept-hiearchy. Pro přidání do struktury je zapotřebí najít rodiče k danému uzlu. Pro tento problém využívá třída Octree metodu selectParent().

Metoda selectParent() prohledává již existující prvky struktury postupným zanořováním za pomocí EPT formátu D-X-Y-Z. Pro vyhledání uzlu se funkce musí rozhodovat na každé úrovni do jakého potomka se musí zanořit. Při rozhodování si napřed vypočítá střed aktuálního uzlu na úrovni hledaného uzlu .

Provede se výpočet pro každou osu a následně vybere do jaké podoblasti patří. Návratové hodnoty funkce jsou uzel a číslo. Zmíněné číslo určuje kolikátý je potomek vyhledaného rodiče. Pokud je číslo nula, znamená to, že uzel je již ve struktuře. Vytvoří se nové uzly

[1] Dostupné na adrese: `https://chromium.cypress.io/win/canary/103.0.5042.0` pro operační sytém Windows

Algoritmus 2: Výpočet středu pro jednu osu

```
1: (((2 ** hledany.D) * (aktualni.X ) /(2 ** aktualni.D))
2: + ( (2 ** hledany.D) * (aktualni.X + 1) /(2 ** aktualni.D)) )/2
```

funkcí createNode(). Při inicializací jsou nataveny parametr, názven souboru, počtem bodů, bounding box a dostupnost dat.

Scene

Scéna slouží jako reprezentace scény point cloudu. Je reprezentovaná jako třída Scene. Uvnitř objektu je uložena konfigurace scény a hierarchie vytvořená ze souboru složky ept-hierarchy. Zároveň obsahuje uživatelem nastavené parametry pomocí uživatelského prostředí. Zároveň se stará o převádění vstupů od uživatele. Hlavní metodou je ale loadScene(). Funkce je volána při zvolení scény uživatelem. Stará se o načtení a nastavení všech potřebných proměnných potřebných pro vykreslování. Vytváří objekt kamery.

lazLoader

Obsah modulu zahrnuje pouze třídu LAZLoader. Třída reprezentuje nastavení a správu pracovníku loader.gl[2]. loaders.gl je kolekce open source knihoven pro načítání a zápis dat pro formáty souborů zaměřených na vizualizaci velkých dat, včetně, geoprostorových a 3D formátů. Při implementaci je využít modul pro načítání LAS a LAZ souborů. Tento modul je vytvořen na základě implementace zmíněného programu plas.io od Udaye Vermy a Howarda Butlera.

K hlavní výhodám knihovny loaders.gl patří: Podpora pracovníků – část odpovědná za získaní, dekompresi a přípravě dat se automaticky spouští v na novém procesu webu, takže hlavní vlákno zůstává volné pro jiné úkoly, zatímco je operace dokončena.

Podpora streamování – program může načítat jen část souboru, což umožňuje zpracování souborů větších než je paměť zařízení .

Při implementaci jsem narazil na problém. Proces pro načítaní dat je vytvořený a pak následně znovu použity pro další soubory. Při načítání dat se ale často stávalo že proces selhal při realokaci paměti.

Při vytváření procesu lze zakázat znovu požívání, ale je to za cenu snížení rychlosti načítání. Zpomalení souvisí s znovuvytvářením nového procesu. Naštěstí jsem našel jednoduché řešení. Po načtení třiceti souborů se proces ukončí a při dalším požadavku je vytvořen nový.

Camera

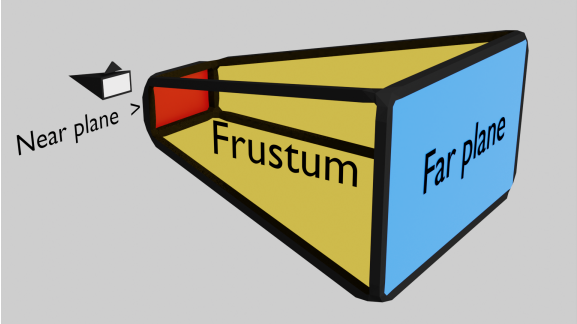
Implementace kamery je skryta v tomto modulu. Při implementaci kamery jsem využil knihovny gl-matrix pro prací s maticemi. Využívám funkce mat4.perspective(out: mat4, fov: number, aspect: number, near: number, far: number), pro výpočet projekční matice. Paremetr fov (Field of view) určuje zorné pole kamery, aspect poměr šířky k výšce obrazovky, near vzdálenost near plane pro ořezávání, far vzdálenost far plane . Druhá matice se nazývá zobrazovací matice (angl. view matrix). Je vytvořena z pozice, pitch a yaw kamery.

[2]Knihovna dostupná z: `https://github.com/visgl/loaders.gl`

Algoritmus 3: Vytvoření view matice

```
1: mat4.fromValues(right.x, up.x, foward.x, 0,
2:                 right.y, up.y, foward.y, 0,
3:                 right.z, up.z, foward.z, 0,
4: - vec3.dot(right, position),- vec3.dot(up, position),- vec3.dot(foward, position), 1);
```

Tyto dvě matice jsou následně vynásobeny. Výsledná matice tvoří pomyslný čtyřboký komolý jehlan kterému se říká viewing frustum . Využívá se pro implementaci určení viditelných objektů ve scéně tzv. clipping.



Obrázek 5.1: Ukázka frustumu

Ovládání kamery

Člověk je schopen ovládat kamera pomocí klávesnice a myši. Klávesy WSAD slouží k pohybu kamery v prostoru. Po stisknutí levého tlačítka myši je převeden pohyb myši na změnu úhlu pohledu. Rychlost kamery lze ovládat pomocí GUI nebo kolečkem myši.

Render

Modul render obsahuje funkce na přípravu a komunikaci s API WebGPU a logiku pro načítaní scény. Celý modul tvoří třída Render. Po vytvoření objektu je zapotřebí zavolat metodu start. Ta zahajuje proces přípravy k vykreslování inicializací WebGPU. Při úspěšném připojení na hardware přes API je spuštěn cyklus render. Podrobnosti k implementaci dalších funkcí budou popsané v sekci vizualizace 5.4.

5.1.1 webpack

Nevýhoda vytváření modulů je vznik mnoha souborů, WebPack tento problém řeší. Webpack přeloží moduly se závislostmi a vygeneruje statickou reprezentaci daných modulů . Dokáže provádět i optimalizace pro kód a odstranění komentářů pro nasazení do provozu.

5.2 Výběr scény

V kapitole uživatelské prostředí 4.7 již bily popsány tři způsoby otevření/ načtení nové scény. Pro implementaci otevření scény z lokálního souborového systému jsem využil Javascript API File System Access. Uživatel po kliknutí na ikonu složky bude vyzdán k vybrání

složky. Poté, co byla vybrána musí uživatel udělí webové aplikaci přístup k dané složce. Je to z důvodu bezpečnosti, protože po uděleni oprávněni muže program přistupovat ke všem souborům v dané složce.

Funkcí showDirectoryPicker() je otevřeno okno pro zvolení složky. Návratová hodnota je manipulátor složky. Přes něj lze iterovat pro prohledání složky. Pokud obsahuje i soubor ept.json pokračuje se dále v načítání.

Při zadání zadání adresy serveru je na něj zavolán HTTP požadavek GET. Je očekáván JSON soubor obsahující seznam modelů / scén. Ty jsou zobrazeny uživateli. Kliknutím na název je zahájen proces načítání.

5.3 Vnitřní datová struktura

Jak již bylo zmíněno v sekci EPT data na serveru jsou uloženy v octree struktuře. Při načítaní scény je převeden soubor s uzly 0-0-0-0.json do vnitřní octree struktury. Soubor je převeden na seznam objektů a před prováděním dalších operacemi je seřazen vzestupně podle hloubky. Popis převodu do interní reprezentace je popsán v kapitole . Struktura je uložena v objektu Scene pod proměnou hierarchy. Vysklít hodnot -1 v poli s počtem bodů je řešen až v metodě updateVisibleNode(). Proces načtení nových uzlů je stejný jak u kořenového souboru 0-0-0-0.json.

5.3.1 Načítání scény

Načítaní jednotlivcích uzlů probíhá na základě pozice kamery ve scéně. Části scény jsou načítány do doby než nastane jeden ze dvou stavů. První je dosažení limitu maximálního počtu bodů na obrazovce nastaveného uživatel. Druhým stavem je plné načtení celé scény. Při každém snímku je zavolaná funkce load(). Pokud byla změněna pozice kamery od posledního snímku, je volána funkce updateVisibleNode(). Cílem funkce je naplnit seznam viditelnými uzly struktury. Funkce je následně volaná rekurzivně pro všechny existující potomky.

Detekce viditelných uzlů

První úkol funkce updateVisibleNode() je zjištěni viditelnosti. Toto probíhá ve funkci checkClip().

Algoritmus 4: Algoritmus pro detekci průniku uzlů s frustumem

Input: Seznam bodů 9 bodů bounding boxu

1: **for** $k = 0$ to $k < 9$ **do**
2: $vec = transformMat4(poziceBodu[k], worldViewProjMatrix)$
3: **if** *abs(vec.x) < vec.w and abs(vec.y) < vec.w and abs(vec.z) < vec.w* **then**
4: return true;
5: **end if**
6: **end for**

Pokud je pozice kamery uvnitř bounding boxu je daná část automaticky považována za viditelnou. Po určení jestli je uzel uvnitř viewing frustumu jsou vypočítaný parametry pro

určení přednosti načtení. Parametry jako vzdálenost kamery ke středu uzlům, procentní zastoupení objemu v prostoru frustum, vzdálenost ke středu obrazovky. Všechny viditelné častí s parametry jsou uloženy do seznamu. Seznam s viditelnými uzly je seřazen tak aby první záznam byl ten co se má načíst jako první. Algoritmus seřadí seznam následujícím způsobem:

Algoritmus 5: Řadící algoritmus pro uzly octree

```
1:  array.sort((a,b) => a.distance * (b.volume / b.centerdist) - b.distance *
      (a.volume / a.centerdist))
```

Následujícím krokem ve funkci load() je načtení uzlu od začátku seznamu. Pro již načtené uzly jediná akce je připočtení do počtu projitích bodů. Jak již bylo zmíněno EPT struktura je aditivní. Proto je v uzlu uložen jeho rodič. Teto skutečnosti je využito pro načtení všech rodičů daného uzlu, pro dosažení největšího rozlišeni v dané oblasti. Načítaní probíhá dokud není součet bodů větší než uživatelem nastavený limit bodů. Pokud je uzel vyhodnocen jako viditelný a není momentálně v paměti je zahájen proces načítaní dat pro daný uzel. K načítaní a dekompresi souboru ve formátu LAZ je využito knihovny loaders.gl[3]. Při zavolání je vytvořen proces na novém vlákně. Výhody tohoto řešení je možnost paralelního načítání více uzlů a asynchronnosti operace.

Po dekompresi dat jsou data předána funkci setGpuBuffer(). Zde jsou dostupné informace namapovány do paměti GPU. Odkaz na jednotlivé části je uložen do odpovídajícího uzlu v octree struktuře.

Optimalizace detekce viditelných uzlů

Velké scény s miliardami bodů mohou být rozdělené do desítek tisíc souborů. Pro určovaní průniku takového množství uzlů každý snímek je zapotřebí příliš mnoho času. Proto je zapotřebí procházet jen část struktury octree. Optimalizace je implementovaná následujícím způsobem. Pokud po vyhodnocení uzlu o hloubce vyšší jak tři, je zjištěno že není viditelná nebo není ještě načtená, přeskočí se kontrola viditelnosti všech potomků. V případe že kamera nevidí celou scénu je toto dostačující řešení, ale jakmile je vidět celá je toto řešení nedostatečné.

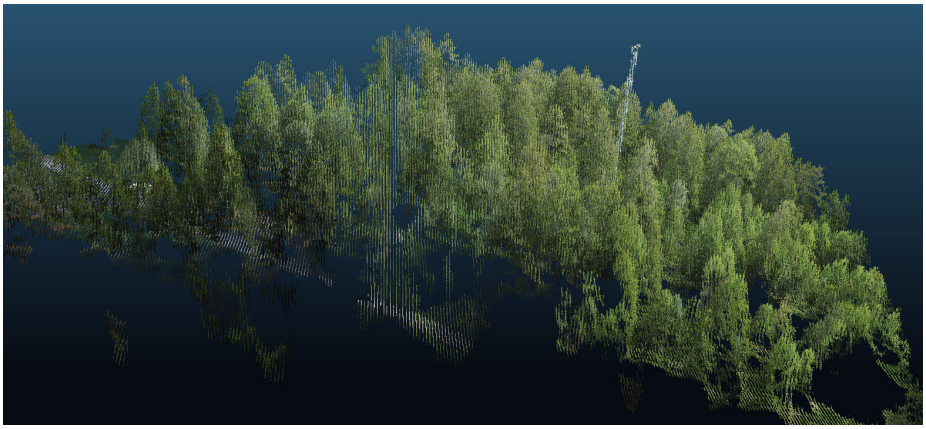
Pro tuto situací je zde druhá optimalizace. Při procházení je sčítán počet načtených, viditelných bodů. Jakmile je překročen limit maximálních bodů zvolený uživatelem jsou kontrolovány uzly do páté úrovně zanoření. Pro zmenšení Pět úrovni je dostatečné pro dostatečné detaily při zobrazovaní celé scény a představuje dostatečnou optimalizaci. Navíc pro snížení využiti procesoru a sníženi spotřeby je metoda updateVisibleNode() volaná jen na prvním snímku a při změně jakéhokoliv parametru kamery.

5.3.2 Offset

Data která jsou určena pro referenci s reálným světem mají jeden problém. Pozice bodů je uložena ve světovém měřítku. To znamená že hodnoty můžou nabývat hodnot v řádech statisíců. Jelikož jsou data o pozicích bodů ve WGSL ukládána v číslech s plovoucí desetinnou čárkou nastává problém. Přesnost takového čísla se snižuje s jeho velikostí. Problém

[3]Dostupné na stánce: `https://loaders.gl`

muže nastat při pohybu s kamerou. Díky nepřesnosti mohou body začít zdánlivě oscilovat ve svém okolí nebo můžou vznikat viditelné pruhy.



Obrázek 5.2: Ukázka vzniku pruhů při snížení přesnosti

Možné řešení je zvýšení přesnost ze základního 32 bitového čísla na 64 bitové. Není to ale ideální řešení, data uložená v paměti se zdvojnásobí a práce s nimi je složitější. Naštěstí jsou v konfiguraci scény, načtené z ept.json souboru, uloženy hodnoty offset. Hodnota offset je vzdálenost od středu souřadnicového systému pro jednu osu. Od všech souřadnic je odečtena hodnota pro každou osu. Tímto krokem je přemístěn model do středu souřadného systému a je zvýšena přesnost.

5.4 Vizualizace

Vizualizace probíhá ve třídě Render. Před začátkem vykreslování je za potřební inicializovat WebGPU rozhraní a nastavení render pipeliny.

Vykreslování probíhá v metodě render() třídy Render. Předem proběhne kontrola velikosti canvasu. Pokud se od předchozího průběhu změnila jeho velikost, jsou vytvořeny nové textury s novým rozlišením. Dalším krokem je zavolání funkce update(). Zde je upravena projekční a zobrazovací matice podle již zmíněného postupu 2. Po dokončení je provedena funkce load(). Nové hodnoty proměnných pro shadery jsou uloženy do uniformního buffru.

Po vzoru optimalizace detekce uzlů je i zde přidána podmínka pro úsporu energie a výkonu. Při splnění alespoň jedné z následujících podmínek je zavolána funkce prepareGPU().

- Změna velikosti okna prohlížeče
- Jakákoliv pohyb kamery
- Byla načtena nová část scény do paměti GPU.
- Změna nastavení vzhledu uživatelem

Na začátku jsou textury nastaveny do výchozího stavu. Ty jsou základem pro vytvoření a zahájení vykreslovacího průchodu (angl. render pass). Je průchodu přiřazena již připravena render pipeline a uniformní paměť s proměnnými pro shadery. Body a jejich informace jsou

uloženy po jednotlivcích částech v odpovídajících uzlech octree. Jak již bylo zmíněno dříve, viditelné uzly jsou seřazeny v seznamu visibleNodes. Přes tento seznam probíhá iterace. Pro načtené uzly je do render passu přiřazeny odpovídající buffery s pozicemi, barvou nebo intensitou a klasifikací bodů. Každá část je vykreslovaná jako nová instance. Počet bodu je podobně jako při načítaní dat sčítán a využit k omezení maximálního počtu bodů, které se budou najednou vykreslovat.

Po dokončení přiřazovaní bufferu, je ukončen render pass a je přidán do WebGPU fronty. Následuje připravení druhého průchodu. Stejně jako u prvního průchodu jsou prvně připraveny textury. Kromě dvou textur do kterých bude vykreslen výsledek, jsou pomocí funkce copyTextureToTexture() zkopírován výsledek prvního vykreslovaného průchodu. Jsou předány fragment shaderu přes WebGPU Biding Group, které slouží k předávání dat shaderům.

Příkazem draw je přidán příkaz pro vykreslení dvou trojúhelníků. Znovu je ukončená definice render passu a je vložen do fronty.

Už je jen pomocí funkce requestAnimationFrame() zavolána grafika k vykonání popsaného procesu. Výsledný snímek je následně zobrazen v html canvasu a celý proces se opakuje.

5.5 Shadery

Shadery jsou napsány v již zmíněném jazyce WGSL. Aplikace využívá sadu dvou vertex a fragment shaderů. Vykresování bodů probíhá ve dvou průchodech. První průchod je volán vertex.wgsl a fragment.wgsl shader. Při drůhém průchodu jsou pak volány shadery vertexFullScrean.wgsl a fragmentFullScrean.wgsl. Pokud je uživatelem nastavena velikost bodu na nulu, je druhý průchod přeskočen. Po prvním průchodu vzniknou dvě textury s rozlišením canvasu. Bod je zde reprezentován jako jeden pixel. První textura obsahuje barvu bodů, druhá jejich hloubkou neboli vzdálenost od kamery.

5.5.1 Vertex shadery

vertex.wgsl

Shader má dvě hlavní funkce. V první části je pozice bodů převedena do NDC (normalized device coordinates) pomocí dvojice matic. Projekční a zobrazovací matice jsou předány shaderu v uniformní paměti a vynásobeny s pozicí bodů. Mezi uniformními proměnnými je dále pozice kamery, vzdálenost far Plane, typ barvy bodů, maximální intensita. Pro druhý průběh je za potřeby hloubkové textury. Hodnoty se pohybuji mezi nulou a jedničkou a značí vzdálenost od kamery. Výpočet probíhá vestavěnou funkcí WGSL distance() . Druhá část

Algoritmus 6: Výpočet hloubky

```
1:  depth = clamp( distance(uniforms.camPosition,pointPosition) /
      uniforms.farPlane ,0.0,1.0 );
```

se zaměřuje výběr zdroje barvy pro body. Uniformní proměnná switch, která nabývá tří hodnot, určuje zdroj barvy bodu. Pokud je proměnná rovna nule, data o barvě jsou brána z LAS souboru. Při jedničce se bere intensita uložená v LAS souboru. Dvojka značí vizualizaci nadmořské výšky daného bodu.

Algoritmus 7: Určení barvy bodu v závislosti na nadmořské výšce

```
1:  normColor =(pointPos.z - (uniforms.offsetElev)) / abs(uniforms.offsetElev -
      uniforms.maxElevation);
2:  normColor = clamp(normColor, 0.0,1.0) ;
3:  r = sin(normColor * pi / 2.0 );
4:  g = sin( (normColor * pi ) ) ;
5:  b = cos( clamp(normColor,0.0, 0.5) * pi );
6:  color = pack4x8unorm(vec4<f32>(r, g, b,1.0)) ;
```

Výstup je tedy struktura skládající se z pozice v NDC, hloubky a barvy uložené jako čtyři osmi bitové čísla bez znaménka spojené do jedné hodnoty.

Klasifikace

Pokud je uživatelem zapnuta klasifikace bodů, probíhá kontrola hodnoty v bufferu classifacion s polem obsahující povolené hodnoty. Body které projdou kontrolou jsou bud zpracováni stejně jako bylo popsánou výše, nebo jsou obarveny uživatelem zvolenou barvou.

vertexFullScrean.wgsl

Výstupem jsou dva trojúhelníky pokrývající celou obrazovku. Toto je příprava pro fragmentFullScrean shader.

5.5.2 Fragment shadery

fragment

Úkol tohoto fragment shaderu je jednoduchý. Převezme informace od vertex.wgsl shader a přiřadí je do framebufru. Pokud jsou dva body na stejné pozici (pixelu) je zapsán ten s menší hodnotou hloubky.

fragmentFullScrean

Tento shader se zabývá změnou velikosti jednotlivých bodů. Velikost bodů je ovlivněna dvěma parametry:

- Uživatelem nastavená maximální velikost bodů
- adaptivní velikost na základě vzdálenosti od kamery a počtu sousedních bodů

Na vstupu jsou hodnoty jako - pozice aktuálně zpracovávaného pixelu, barevná a hloubková textura s daty po prvním průchodu, očekávaná velikost bodu, barva pozadí / světa a náměr kamery . Pro každý pixel je zavolán tento shader. Prvním krokem je prohledání okolí. Velikost okolí je určen proměnou pointSize stejnou pro obě osy. Každému pixelu je kontrolována hodnota hloubky uložené v textuře. Dvě nejnižší hodnoty jsou uloženy společně s jejich souřadnicemi. Současně je prováděn součet sousedících bodů . Jsou počítaný jen body s hloubkou blízké nejmenší hodnotě. Body jsou od pozadí rozpoznávány díky hodnotě hloubky . Pozadí má vždy hodnotu 1.0 a maximální hodnota bodu je 0.99.

Jak již bylo zmíněno na začátku hodnoty vzdálenosti od kamery a počtu sousedních bodů slouží k určení velikosti bodu. Čím více bodu je v okolí, tím menší bude vykreslen a se vzdáleností se body taktéž zmenšují. Pokud má být bod menší než prohledávané okolí muže nastat stav kdy nejbližší bod už není v dosahu. V takovém případě je použit druhý bod. V případě že i tento bod nevyhovuje, zůstane pixel takový jaký byl po prvním průchodu s výjimkou pozadí. Pozadí je vykresleno podle následujícího algoritmu.

Algoritmus 8: Výpočet hloubky

```
1: texSize: vec2 = textureDimensions(texture);
2: shift = (uniforms.pitch / (pi / 2.0) ) * texSize.y ;
3: ds = (pixelCoord.y + shift) / texSize.y;
4: ds = 1.0 - clamp(ds, 0.0 , 1.0);
5: color = vec4<f32>(ds * R.High + R.Low,ds * G.High + G.Low, ds * B.High
     + B.Low, 1.0);
```

Hodnoty R.Hight až B.Low jsou určené uživatelem prostředím GUI a tvoří gradient od tmavší ke světlejší barvě.

5.6 Uvolňovaní paměti

Uvolňovaní paměti je neideální ale nevyhnutelný proces. Přesun dat do paměti grafické karty je pomalí proces. Program se snaží využít paměť grafiky do posledního bajtu. Opětovné načítání a přesun dat je nechtěný proces. Po načtení uzlů jsou data jen v paměti grafiky. Ve chvíli kdy zbývá minimum volné grafické paměti, jsou při provádění metody updateVisibleNode() přidávaný načtené uzly, které označí funkce jako mimo zorné pole kamery, do seznamu. Ukládá se tam odkaz na uzel, vzdálenost od kamery a objem.

Po dokončení metody je seznam seřazen podobným algoritmem jak pro viditelné uzly 5. Místo vzestupně je ale seřadí sestupně. Přes seznam je spuštěná iterace for cyklem. Samotné uzly nejsou smazaný, jen všechny instance GPUBuffer jsou odstraňovaný dokud není místo pro nové. Navíc jsou nastaveny jako nenačtené. Při další potřebě uzlu bude muset program provést načtení jak bylo popsáno v kapitole Načítání scény 5.3.1.

5.7 Iterace řešení

Tvorba softwaru je iterativní proces při kterém se vždy dá něco vylepšit. Vytváření vlastní implementace vizualizace point cloudů nebylo výjimkou.

První způsob implementace zobrazení bodů jako čtverce v prostoru, je jeden z příkladů. Do paměti grafické karty byly pro každý bod uloženy čtyři stejné záznamy. Každý bod byl vykreslen jako čtverec skládající se ze dvou trojúhelníku. Řešení bylo plně funkční i s dynamicky se měnící velikostí v závislosti vzdálenosti od kamery. Ale zároveň mělo dva velké problémy. Velikost dat v bufferch grafické karty byla čtyřnásobná a po načtení 20. milionech bodech byla paměť o velikosti 6GB zaplněná.

Druhý problém byl ve výkonosti řešení. Rychlost vykreslování se blížil řešením na bázi WebGL. Při zobrazení 10 milionů bodu byla snímková frekvence těsně pod 60. WebGPU by mělo poskytovat mnohonásobně rychlejší řešení. Bylo jasné že tímto směrem cesta nevede.

Kapitola 6

Testování

V této kapitole se budeme zabývat vyhodnocením vytvořené aplikace. Porovnání s existujícími řešeními a ukázkou výsledků na několika vybraních point cloudech.

6.1 Performance

Tato část slouží k zhodnocení výkonu a prohlížeče point cloudů.

Tabulka 6.1: Použity hardware a software na testování

Paměť	Procesor	GPU	OS	Webový prohlížeč
16GB	i7, 8750H 2,2GHz	GTX 1060 mobile	Windows 11	Chrome Canary 103.0.5042

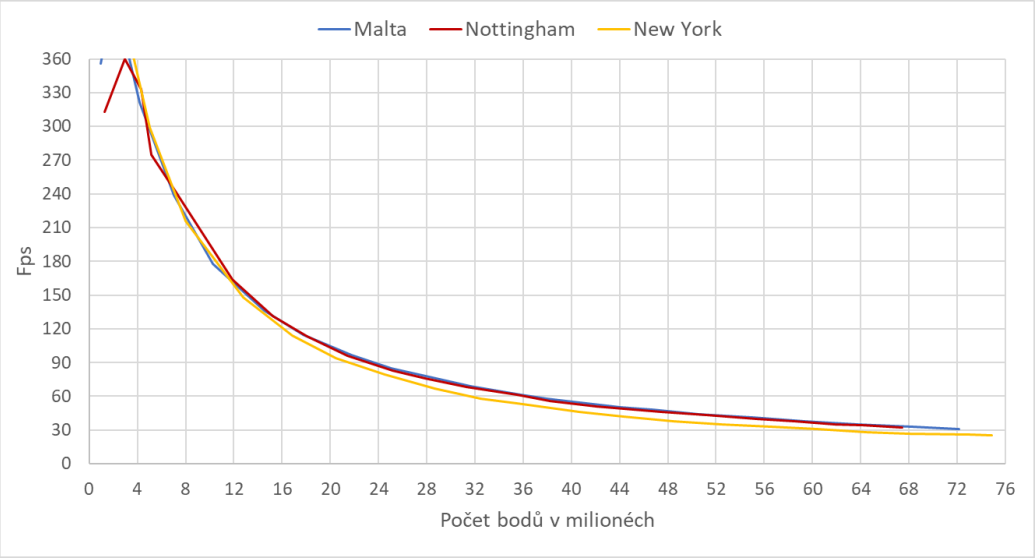
Tabulka 6.2: Informace k testovaným scénám

Název	Počet bodů	počet uzlů
Nottingham	67,8M	1605
Malta	72,2M	1911
New Yourk City	4 755,02 M	71600

Bližší informace o testovacích scénách se nacházejí v části 6.3. Pro kolekci dat byl v prohlížecí vypnutý V-Sync[1], za pomocí argumentů při spuštění $--disable-frame-rate-limit$ a $--disable-gpu-vsync$.

Měření snímku probíhá pomocí metody performance.now(), která vrací hodnotu představující čas, který uplynul od načtení aktuální záložky ve webovém prohlížeči. Pro každý snímek je zvýšeny čítač. Po uplynutí sekundy od posledního výpisu, je počet snímku podělen přesným časem od výpisu. Hodnota je vypisována do horní středu obrazovky.

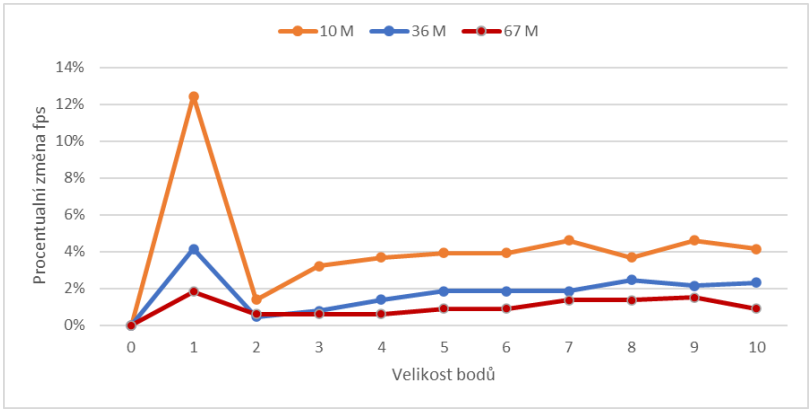
[1]Technologie pro zamezení roztržení obrazu (angl. screen tearing), zamezením grafické kartě v provádění čehokoli související s paměti displeje, dokud monitor nedokončí svůj aktuální obnovovací cyklus.



Obrázek 6.1: Graf porovnání point cloudů různé velikosti

V grafu můžeme pozorovat minimální dopad na výkon při velkém množství uzlů. Rozdíl mezi point cloudem z Malty a New Yorku je v jednotkách snímku. Pokud budeme považovat 60 snímků za sekundu za přijatelný výkon pro interaktivní aplikaci, muže za pomocí teto implementace dosáhnou vykreslovaní až 36 milionů bodu na testovaném hardwaru. Když se smíříme s 30 snímky za sekundu, lze dosáhnou při menším počtu uzlů octree struktury 77 milionům bodů.

Testování závislost velikosti a počtu bodů na počet snímků za sekundu.



Obrázek 6.2: Graf závislost velikosti a počtu bodů na počet snímků za sekundu

Z grafu můžeme zjistit, že menší počet bodů způsoby znatelný pokles snímků při zvýšení velikosti bodů. Celkový pokles snímku je následující: 10% pro 67 milionů bodů 20% při 36 milionech a 45% u 10 milionů bodů. Očekávaným zjištěním je velký propad snímků při zapnutí dvojího průchodu.

Rychlost načítaní scény

Sekce se zabývá měřením rychlostí načítaní scény různými způsoby a z různých serveru. Sloupec v mezi paměti značí, že část scény je uložená webovým prohlížečem lokálně na počítači.

Tabulka 6.3: Seznam použitých serverů

Zařízení	typ úložiště	OS
localhost	HDD	Windows 11
raspberry pi 3b+	SDHC karta	Raspbian

Tabulka 6.4: Naměřené hodnoty

Zdroj	počet bodů	v mezi-paměti	čas
localhost	67 M	ne	1:57
localhost	67 M	ano	1:38
souborový systém	67 M	ne	1:35
raspberry pi	68 M	ne	2:58
raspberry pi	68 M	ano	2:25

Z výsledků je patrné zrychlení načítaní při uložení do mezi paměti webovým prohlížečem. Naměřené hodnoty z lokálního serveru odpovídají načítaní o rychlosti 680 tisíc bodů za sekundu. U načítání ze serveru umístěném na raspberry pi, lze pozorovat nedostatečná rychlost zařízení. Celkový čas je vyšší a zároveň znatelnější rozdíl při načtení z mezi paměti prohlížeče. Nejrychlejší způsob načítání je přímo ze souborového systému.

6.2 Porovnání

V sekci bude porovnáno řešení popsané v této práci s existujícími řešeními zmíněnými v kapitole 3.4. Hlavní rozdíl oproti existujícím řešením je výkonost řešení.

Potree klesne vykreslování pod hranici 60 snímku za sekundu při standardním režimu při 8,5 milionech bodů a hight quality při třech milionech. Vzhledově je ale tento režim nevyrovnatelný.

Plas.io se chová podobně jak potree při standardním režimu. Hranice 60 snímků je dosažitelná do zobrazování 8,5 milionů bodů najednou.

Výsledky mého řešení byli popsány v předchozí kapitole. Největší zrychlení procesu vykreslováním je díky zvolenému programovacího rozhraní WebGPU . Jednoznačné porazí všechny existují prohlížeče založené na WebGL. Momentální nevýhoda je dostupnost WebGPU jen ve vývojových verzích webových prohlížečů. Proto se může stát že aktuální verze není stabilní nebo vůbec nefunguje. Během vývoje teto aplikace tři verze prohlížeče Chrome Canary při inicializaci programového rozhraní WebGPU selhaly a nebylo možno je zprovoznit.

V opačném případě, WebGL je dostupný skoro ve všech populárních prohlížečích s výjimkou mobilního prohlížeče Opera Mini [9]. Využití WebGPU je momentálně omezeno na počítače. V budoucnosti by měla přibít podpora pro mobilní zařízení.

Při porovnání ostatních funkcí je jednoznačně nejlepší prohlížeč potree. Potree nabízí velké spektrum nástrojů pro měření, zpracování různých parametrů. Pokud někdo potřebuje nějaké nové funkce muže je do implementovat v prostředí Three.js, jelikož potree je dostupný jako zástupní modul. Plas.io nabízí stejně jako implementace popsaná touto prací, různé režimy zobrazení daného point cloudu.

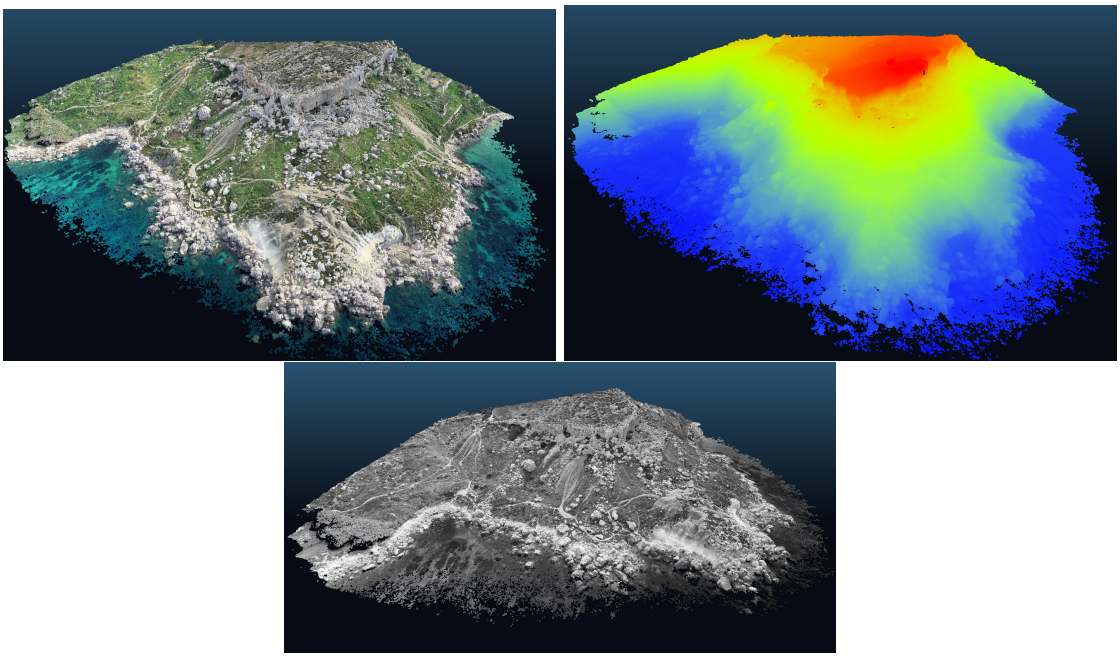
Cesium samotné nenabízí žádné možnosti interakce s body. Uživatel si ale může pomocí CesuimJS snadno vytvořit vlastní řešení.

6.3 Příklady

Sekce se zabívá ukazkou a popisem point cloudů vykreslených pomocí implementovaného řešení popsaného v kapitole Implementace.

Průzkum výběžku Selmun, severní Malta, 2021 [2]

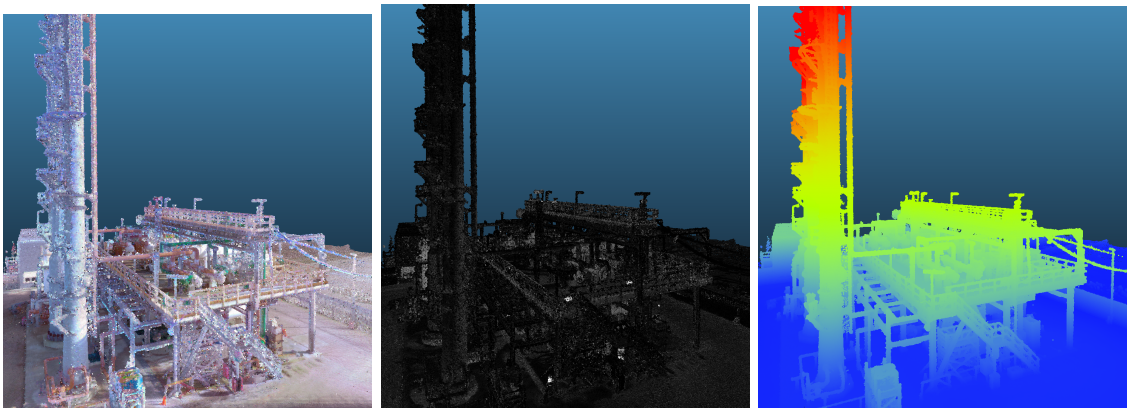
Tato data byla shromážděna technikou fotogrammetrie pomocí DJI Phantom 4 Pro nad výběžkem Selmun, který se nachází podél severovýchodního pobřeží Malty, centrálního Středomoří. Scéna má 72,222,641 bodů o hustotě: 394,73 bodů/m2.



Obrázek 6.3: Barevná, Nadmořská výška, intesita vráceného paprsku

Chemická továrna[3]

Data pořízena LiDAR technology za pomocí ZEB Horizon s příslušenstvím ZEB Discovery. Célá scéna obsahuje 68,227,900 bodů a v EPT reprezentaci zabírá 581 MB na disku.



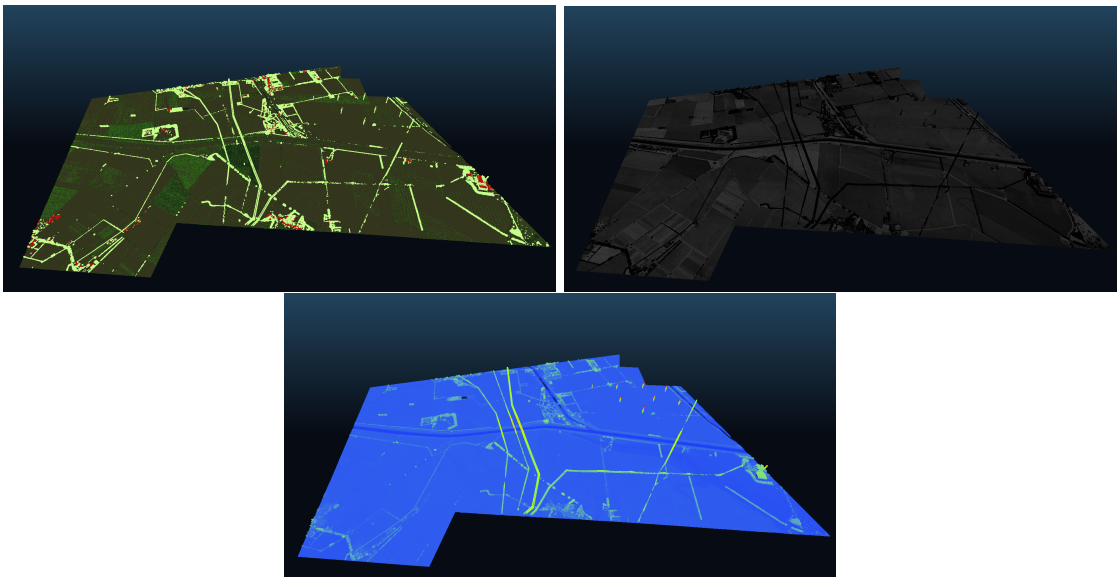
Obrázek 6.4: Zleva: barevná, intenzita návratu a nadmořská výška

[2]Dostupne z: `https://doi.org/10.5069/G9D21VSC`
[3]Dostupne z: `https://samples.geoslam.com/Potree/Chemical_Plant/files/Chemical_Plant_las.zip`

UK[4]

Data byla pořízena LiDAR technology z letadla. Data jsou pořízená v rámci národního programu LIDAR agentury pro životní prostředí. Cílem je poskytovat přesné údaje o nadmořské výšce v prostorovém rozlišení 1 m pro celou Anglii do konce roku 2021.

Země byla rozdělena do přibližně 230 logických bloků s průzkumem zahájeným v listopadu 2016. Průzkumy se zveřejňují prostřednictvím portálu DEFRA Data Services Portálu.

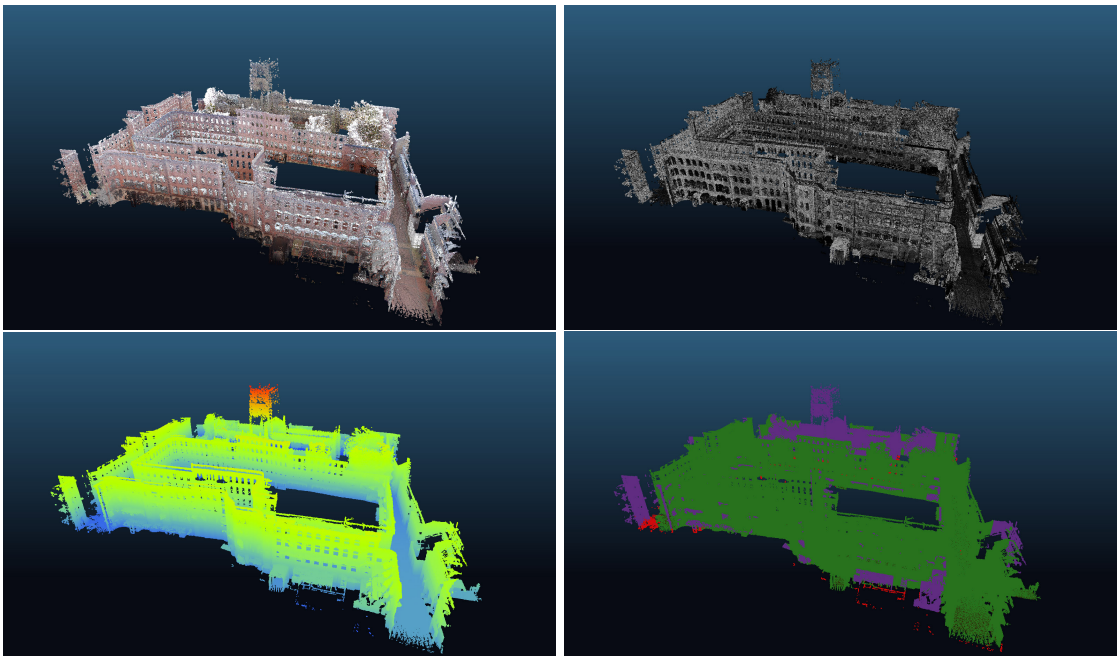


Obrázek 6.5: Zleva : Klasifikace, intenzita návratu a nadmořská výška

[4]Dostupne z: `environment.data.gov.uk/DefraDataDownload`

Nottingham Lace Market[5]

Data pořízena LiDAR technology za pomocí ZEB Horizon s příslušenstvím ZEB Discovery od společnosti GeoSLAM. Celá scéna obsahuje 67,771,076 bodů a v EPT reprezentaci zabírá 514 MB na disku.



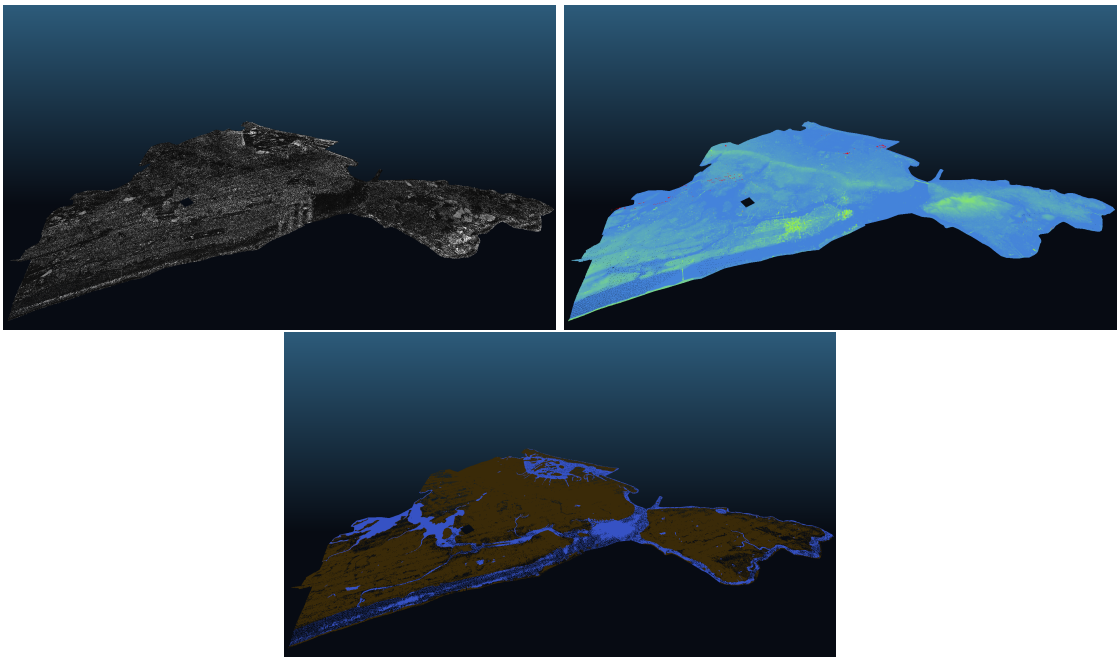
Obrázek 6.6: Zleva: Barva, intenzita návratu, nadmořská výška a klasifikace

[5]Dostupne z: `https://samples.geoslam.com/Potree/Nottingham_Lace_Market/files/Nottingham_Lace_Market_las.zip`

New York city[6]

V reakci na rostoucí potřeby vysoce kvalitních údajů o nadmořské výšce v celých Spojených státech, U.S. Geological Survey 3D Elevation Program (3DEP) systematicky dokončuje vytváření celonárodního point cloud , aby poskytl první národní data o nadmořské výšce ve vysokém rozlišení.

Od února 2019 jsou všechny data uloženy na volně dostupném amazon bucketu [6]. Bucket je označení pro pojmenované cloudové úložiště, zde to konkretně jsou data uložená na serverech Amazonu. Celá scéna obsahuje 4,755,025,996 bodů a je uložena na serverech amazonu.



Obrázek 6.7: Od horního levého okraje : Intenzita návratu, nadmořská výška a klasifikace

[6]Dostupne z: `https://portal.opentopography.org/usgsDataset?dsid=NY_NewYorkCity`

Kapitola 7

Závěr

Cílem této práce bylo seznámení s procesem tvorby interaktivních zobrazovacích aplikací pro datově náročné 3D scény ve webovém prohlížeči. Pro výběr konkretního řešení byli nastudovány různé typy řešení. Podrobně také bylo nastudováno programovací rozhraní WebGPU a jeho jazyk pro psání shaderů WGSL. Z pohledu serverového řešení bylo zvoleno jednoduché datové schéma Entwine point tile, které nabízí jednoduché ale elegantní řešení.

Následně byla vytvořena implementace vlastní interaktivní aplikace pro vykreslování point cloudů. Na vytvořené aplikaci byli provedeny testy na zjištění výkonosti řešení.

Výsledky byly porovnány s nastudovanými řešeními. Ze znatelně lepší výkonosti řešení je jasné ,že programovací rozhraní WebGPU opravdu poskytuje rychlejší a efektivnější přístup ke grafické kartě .

Aktuální verze aplikace je dostupná na webové stránce[1]. Kde bude aktualizovaná při změně standartu WebGPU.

Možná vylepšení

Cíle do budoucna je možnost přepsání aktuálních shaderů pomocí compute shaderu. Podle výzkum Markuse Schuetze je použiti compute shaderů až desetkrát rychlejší. Limitujícím faktorem je fakt, že znatelné vylepšení se projevují při renderování velkých instancí s buffery obsahující více jak milion bodů [17]. Při využití Entwine point tile struktury jsou instance bodů velké maximálně stovky tisíc bodů, takže toto řešení by nemělo velký dopad na výkonost. Dalším možným vylepšením je stejně jako u potree možnost měření vzdálenosti. Implementací takové funkce se zvedne rozsah použitelnosti aplikace.

[1] Odkaz na stránku: `https://pointcloudviewer.xyz`

Literatura

[1] 3DStreamingToolkit. *3D Streaming Toolkit Documentation* [online]. 3DStreamingToolkit, 2018 [cit. 2022-05-08]. Dostupné z: https://3dstreamingtoolkit.github.io/docs-3dstk/.

[2] Beaufort, F. *Access modern GPU features with WebGPU* [online]. Corentin Wallez, 26. září 2021. Revize 23 března 2022 [cit. 2022-05-01]. Dostupné z: https://web.dev/gpu/.

[3] Burger, D. *WebGPU vs Pixel Streaming* [online]. Medium, 7. února 2021 [cit. 2022-04-20]. Dostupné z: https://medium.com/swlh/webgpu-vs-pixel-streaming-a-view-from-afar-18c7819db2fd.

[4] Catuhe, D. *Babylonjs* [online]. 2022 [cit. 2022-03-20]. Dostupné z: https://www.babylonjs.com.

[5] Cesium GS, I. *The Cesium Platform* [online]. Cesium GS, Inc, květen 2022 [cit. 2022-05-02]. Dostupné z: https://cesium.com/platform/.

[6] Communications a Publishing. *The USGS 3D Elevation Program (3DEP) is excited to announce the availability of a new way to access and process lidar point cloud data from the 3DEP repository.* [online]. USGS, 07. února 2019 [cit. 2022-04-28]. Dostupné z: https://www.usgs.gov/news/technical-announcement/usgs-3dep-lidar-point-cloud-now-available-amazon-public-dataset.

[7] David Neto (Google), M. C. M. A. I. *WebGPU Shading Language* [online]. Květen 2022 [cit. 2022-05-6]. Dostupné z: https://www.w3.org/TR/WGSL/.

[8] Döllner, J., Hagedorn, B. a Klimke, J. *Server-Based Rendering of Large 3D Scenes for Mobile Devices Using G-Buffer Cube Maps* [online]. Proceedings of the 17th International Conference on 3D Web Technology, srpen 2012. Dostupné z: https://hpi.de/fileadmin/user_upload/fachgebiete/doellner/publications/2012/DHK2012/paper.pdf.

[9] Group, K. *WebGL - 3D Canvas graphics* [online]. caniuse, duben 2022 [cit. 2022-05-08]. Dostupné z: https://caniuse.com/webgl.

[10] Group, K. *WebGL specification and resources* [online]. Khronos Group, duben 2022 [cit. 2022-04-28]. Dostupné z: https://www.khronos.org/webgl/.

[11] Higgins, S. *LASzip* [online]. navvis, 23. února 2021 [cit. 2022-04-29]. Dostupné z: https://www.navvis.com/blog/everything-you-need-to-know-about-point-clouds-navvis.

[12] Isenburg, M. *LASzip* [online]. rapidlasso, 28. října 2021 [cit. 2022-04-22]. Dostupné z: https://rapidlasso.de/laszip/.

[13] Manning, C. *Trillions of points Massive point clouds as infrastructure* [online]. 2017 [cit. 2022-04-08]. Dostupné z: https://s3.amazonaws.com/entwine.io/slides/foss4g2017.

[14] Natijne, A. van. *Web based visualisation of 3D radar and LiDAR data.* Delft, NL, 2017. Bakalářská práce. Delft University of Technology. Dostupné z: https://repository.tudelft.nl/islandora/object/uuid:b01d66e1-ce16-458e-a350-348659d939e9/datastream/OBJ/download.

[15] Photogrammetry, T. A. S. for a Sensing, R. *LAS Specification 1.4 - R15* [online]. 425 Barlow Place, Suite 210 Bethesda, Maryland 20814-2160: The American Society for Photogrammetry and Remote Sensing, červenec 2019, revidováno 09 7 2019 [cit. 2022-3-10]. 50 s. Dostupné z: http://www.asprs.org/wp-content/uploads/2019/07/LAS_1_4_r15.pdf.

[16] Schuetz, M. *Potree: Rendering Large Point Clouds in Web Browsers.* Vídeň, 2016. Diplomová práce. Technická univerzita Vídeň, fakulta informatiky. Dostupné z: https://www.cg.tuwien.ac.at/research/publications/2016/SCHUETZ-2016-POT/SCHUETZ-2016-POT-thesis.pdf.

[17] Schuetz, M. *Potree in WebGPU* [online]. Khronos Group, květen 2021 [cit. 2022-04-25]. Dostupné z: https://www.khronos.org/assets/uploads/developers/presentations/webgpu-potree_May21.pdf.

[18] Wimmer, M. a Scheiblauer, C. *Instant Points: Fast Rendering of Unprocessed Point Clouds.* Leden 2006. 129-136 s. Dostupné z: https://diglib.eg.org/bitstream/handle/10.2312/SPBG.SPBG06.129-136/129-136.pdf.

[19] Zwicker, M., Pfister, H., Baar, J. van a Gross, M. *Surface Splatting.* New York, NY, USA: Association for Computing Machinery, 2001 [cit. 2022-05-09]. DOI: 10.1145/383259.383300. Dostupné z: https://doi.org/10.1145/383259.383300.

Příloha A

Obsah přiloženého paměťového média

README

Návod k instalaci a spuštění aplikace, url testovacích scén

Plakát.pdf

Plakát prezentující práci, její cíle a výsledky.

/src

Zdrojový kód aplikace a potřebné knihovny pro správnou funkcionalitu.

/pictures

Obrázky vytvořené za pomocí implementované aplikace v plné kvalitě

Příloha B

Manuál

Zdrojové soubory aplikace se nacházejí ve složce src. Projekt je postaven na prostředí node.js. Pro sestavení aplikace je zapotřebí mít nainstalovaný node.js na počítači. Příkazem **npm install** se stáhnou potřebné knihovny. Po dokončení lze spustit kompilaci a spuštění aplikace příkazem **npm start**.

Spustí se lokální server a otevře se prohlížeč chrome. Bohužel nemůžu určit jaká verze prohlížeče se spustí, pro správnou funkcionalitu je zapotřebí webového prohlížeče Chrome Canary.

Po nainstalování je zapotřebí na přepnout flag ve vebovém prohlížeči pro možnost využití WebGPU. Do baru pro adresu zadejte **about:/flags** a v seznamu vyhledejte flag **Unsafe WebGPU** a povolte jeho využívání.