VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

SEMESTRÁLNÍ PRÁCE

Brno, 2016 Martin Bařinka

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

PROGRAMOVÝ MODUL PRO ZOBRAZENÍ MRAČNA BODŮ VE VIRTUÁLNÍ REALITĚ
SOFTWARE MODULE FOR POINT CLOUDS DISPLAY IN VIRTUAL REALITY

SEMESTRÁLNÍ PRÁCE
SEMESTRAL THESIS

AUTOR PRÁCE — **Martin Bařinka**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. Luděk Žalud, Ph.D.**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Semestrální práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Martin Bařinka ***ID:*** 159656
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Programový modul pro zobrazení mračna bodů ve virtuální realitě

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s moderními technologiemi pro zobrazení virtuální reality (VR), především s tzv. helmami pro VR.
2. Seznamte se s možnostmi programování zobrazení v těchto zařízeních na počítačích typu PC. Prozkoumejte možnosti v operačních systémech Microsoft Windows a Linux. Rozeberte výhody jednotlivých programovacích jazyků.
3. Vyberte vhodný framework nebo knihovnu pro programování vlastního systému pro zobrazení masivního mračna bodů (řádově až stovky milionů bodů).
4. Navrhněte nebo vyberte vhodný datový formát kompatibilní s širším spektrem běžně používaných programů pro práci s mračny bodů (CloudCompare, Rhino3D, ...).
5. Realizujte alespoň základ programu, který bude schopen načíst mračno bodů, zobrazit jej ve vybrané helmě VR a umožní plynulý pohyb ve virtuálním prostoru.

DOPORUČENÁ LITERATURA:

Sky Nite; Virtual Reality Insider: Guidebook for the VR Industry; November 1, 2014; New Dimension Entertainment; ISBN-13: 978-0990999928

Termín zadání: 19.9.2016 ***Termín odevzdání:*** 6.1.2017

Vedoucí práce: prof. Ing. Luděk Žalud, Ph.D.
Konzultant semestrální práce:

doc. Ing. Václav Jirsík, CSc., *předseda oborové rady*

ABSTRAKT

Cílem práce bylo seznámit se s hardwarem a softwarem zařízení pro virtuální realitu, prozkoumat možnosti vývoje a navržením softwarového řešení pro zobrazování medicínských dat, z CT, MRI nebo RoScan vyvíjeným na VUT.

KLÍČOVÁ SLOVA

Virtuální realita; mračna bodů; mesh; 3D zobrazování

ABSTRACT

The goal of my task was to understand hardware and software of equipment for virtual reality, investigate possibilities to develop and design software solution for medical dates from CT, MRI or RoScan, developed at VUT.

KEYWORDS

Virtual reality; pointcloud; mesh; 3D rendering

BAŘINKA, Martin *Programový modul pro zobrazení mračna bodů ve virtuální realitě*: semestrální projekt. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, Rok. 25 s. Vedoucí práce byl prof. Ing. Luděk Žalud, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svůj semestrální projekt na téma „Programový modul pro zobrazení mračna bodů ve virtuální realitě“ jsem vypracoval(a) samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto semestrálního projektu jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu prof. Ing. Luděk Žalud Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

Úvod **8**

1 Teoretická část studentské práce **9**
1.1 Virtuální realita 9
1.1.1 Stereoskopie 9
1.1.2 Headsety Virtuální reality 10
1.2 Možnosti programování zařízení pro VR 12

2 Výběr datového formátu a Implementace programu **14**
2.1 Datový formát 14
2.1.1 Textový formát 14
2.1.2 Stereo Lithography STL 14
2.1.3 Wavefront OBJ 15
2.1.4 Polygon file format PLY 15
2.1.5 Point Cloud Data 15
2.1.6 Finální výběr 15
2.2 Engine 16
2.2.1 Požadavky virtuální reality 16
2.2.2 Vykreslování bodů 16
2.2.3 Omezení procesorové zátěže 17
2.2.4 Podporovaný formát 17
2.2.5 Segmentace mračna 18
2.2.6 Grafické rozhraní 18
2.2.7 Zobrazení mračna 19
2.3 Ovládání programu 21

3 Závěr **23**

Literatura **24**

SEZNAM OBRÁZKŮ

1.1 VFX1 headgear 10
1.2 HTC Vive 11
1.3 Oculus Rift 12
2.1 Pohled na načtené mračno 19
2.2 Demonstrace změny velikosti bodů 20
2.3 Demonstrace přimíchávání škály skalárního pole 20
2.4 Hlavní menu 22
2.5 Menu objekty 22

ÚVOD

Cílem práce je seznámit se s hardwarem i softwarem zařízení pro virtuální realitu, prověřit možnosti vývoje a navržení softwarového řešení pro zobrazování medicínských dat, jako například CT, MRI a RoScan, který je vyvíjen na VUT.

Práce byla zadána mým vedoucím a zaujala mě, přestože jsem doposud o tomto tématu nic nevěděl a veškeré teoretické informace musím čerpat z internetu.

V rámci mé práce jsou vstupní data ve formě mračna bodů nebo meshe a úkolem této práce je zpracovat program, nebo alespoň základ, který bude schopen mračno bodů načíst a zobrazit v helmě virtuální reality. Prvním úkolem bude vyhledání informací na internetu o technologiích virtuální reality, způsobech vývoje aplikací, datových formátech a vývoji 3D aplikace.

Poté musím vybrat vhodné knihovny a vhodný datový formát tak, aby má aplikace mohla fungovat s mě dostupným headsetem HTC Vive a aby byl datový formát kompatibilní s již používanými programy.

Díky tomuto programu bude moct uživatel vidět zobrazené data jako plastický obraz, což v medicíně může umožnit zkvalitnění diagnostiky.

1 TEORETICKÁ ČÁST STUDENTSKÉ PRÁCE

1.1 Virtuální realita

Virtuální realita (VR) je technologie, umožňující uživateli interagovat se simulovaným prostředím. Jde o vytváření vizuálního, sluchového, hmatového či jiného vjemu, budícího subjektivní dojem skutečnosti pomocí počítače, speciální audiovizuální helmy, brýlí popřípadě obleku.

1.1.1 Stereoskopie

Za nejjednodušší VR se dá považovat stereoskopie, což je technika vytváření 3D vjemu pomocí dvou 2D obrazů.

Stereoskopie využívá přirozené vlastnosti člověka, tzv. binokulárního vidění. To znamená, že se obrazy vnímané simultánně oběma očima spojí v jeden a navíc nám umožňuje vnímat hloubku prostoru. Každé oko vidí obraz mírně posunutý a mozek poté vytvoří na základě těchto vjemů 3D obraz. Existuje určité procento lidí, u kterých tato schopnost není vyvinuta a ti nejsou schopni správně 3D obraz vidět.

Tradiční stereoskopie vytváří 3D iluzi pomocí dvou 2D obrazů, s různými perspektivami téhož objektu, které se příliš neliší od toho, co by oči přirozeně viděly. Tím vlastně donutí mozek tyto dva rozdílné obrazy zpracovat jako v realitě a mozek vytvoří iluzi prostorového vidění.

Anaglyf

Je to technika, která nám umožňuje vytvářet stereoskopický obraz. Obraz pro každé oko je na tomtéž displeji, ale je v opačném odstínu, typicky červené a azurové. Pozorovatel má pak brýle, které mají barevné filtry a tak každé oko vidí pouze jeden z obrazů.

Nevýhodou tohoto řešení je ztráta barevné informace, protože pomocí ní je zakódovaná hloubka obrazu.

Polarizace

Další způsob zobrazení stereo obrazu je pomocí polarizačních brýlí. Tyto brýle mají polarizaci jednoho oka kolmou proti druhému. Obraz se pak zobrazuje nebo promítá střídavě přes dvě polarizační skla a brýle pak tyto obrazy odseparují.

Nevýhodou tohoto řešení je nedokonalá polarizace, která plně neodfiltruje druhý obraz a tak vznikají artefakty v obraze.

Obr. 1.1: VFX1 headgear

Aktivní zatmívací brýle

Tento způsob pracuje na principu multiplexování. Brýle, synchronně s displejem nebo projektorem, střídavě přepínají obraz pro obě oči.

Nevýhodou tohoto způsobu je velká cena.

LCD brýle

Toto jsou brýle, které mají 2 displeje, na které se promítají jednotlivé obrazy. Tyto brýle nemění svůj obraz při pohybu hlavou, jsou vhodné například pro 3D filmy. Brýle, které podporují sledování pohybu se řadí již do headsetů.

Nevýhodou této technologie je vysoká cena, zahřívání a nutné propojení kabelem.

1.1.2 Headsety Virtuální reality

VFX1

Za první komerční headset se dá považovat VFX1. Byl prodáván v devadesátých letech. Skládal se z helmy, ovladače a ISA karty.

Tento headset měl dva LCD displeje s rozlišením 263x230 a zorný úhel 45°.[13]

HTC Vive

HTC Vive je headset pro virtuální realitu, vyvinut firmou HTC a Valve Corporation, byl vydán 5. dubna 2016. Tato sada obsahuje headset, dva prostorově sledované

Obr. 1.2: HTC Vive

ovladače a dva snímače polohy. Uživatel se může pohybovat po místnosti velikosti 4.6m na 4.6m.[20]

HTC Vive využívá dva displeje s rozlišením 1080x1200[21] a obnovovací frekvencí 90Hz. Zařízení používá více než 70 senzorů jako jsou gyroskopy, akcelerometry a laserové snímače.[20][18] Kamera na přední straně headsetu pomáhá identifikovat objekty před uživatelem a zabrání tak kolizi.

Tento headset je podporován platformou SteamVR, vyvíjenou společností Valve Corporation.

Oculus Rift

Oculus Rift je headset od společnosti Oculus VR, vydaný 28.3.2016.

Oculus Rift používá dva displeje s rozlišením 1080x1200, obnovovací frekvencí 90Hz a zorným úhlem 110°.[16]. Navíc má zabudovaná i sluchátka. Pozice je sledována kromě akcelerometrů a gyroskopů také pomocí stacionární IR kamery, která snímá IR světla na headsetu a kompenzuje chyby inerciální jednotky v headsetu.

Oculus Rift je podporován Oculus Runtime a Steam VR.

Obr. 1.3: Oculus Rift

1.2 Možnosti programování zařízení pro VR

Všecha tato zařízení pro svou funkci vyžadují, aby daný program s nimi komunikoval pomocí speciálního API.

Současný trh je velmi roztříštěný, každý výrobce headsetu má vlastní API a runtime, navzájem nekompatibilní. Tuto situaci se snaží zlepšit Valve, které vyvinulo OpenVR API primárně pro Vive a nabádá ostatní výrobce, aby dodali podporu jejich zařízení. OpenVR již podporuje Oculus, Vive a OSVR.

Jako další ohlásila vývoj jednotného API organizace Khronos, jejímž cílem je vyvinout API, které nahradí API všech výrobců a tím ulehčí práci vývojářů aplikací. Zároveň by se i zlevnil vývoj aplikací a rozšířil se okruh potenciálních zákazníků. Valve již ohlásila, že do tohoto projektu předává své rozhraní OpenVR. [19]

Já mám k dispozici Vive a Oculus, výběr mých možností se proto zúží na:

- Unity 3D, což je multiplatformní herní engine vyvinutý společností Unity Technologies. Tento engine podporuje všechny současné headsety.
- Unreal Engine, multiplatformní herní engine vyvinutý společností Epic games. Tento engine podporuje všechny současné headsety.
- Použít OpenVR C++ API a naprogramovat si vlastní aplikaci od základu.

Herní engine ulehčuje práci vývojářům, protože za ně řeší základní funkce všech her, např. kolize objetků, grafické efekty a v našem případě řeší i přístup k headsetu, tím pádem se mohou zaměřit více na samotný obsah hry. Na herní engine jsou kladeny jiné nároky, než na mou aplikaci. Geometrická náročnost modelu je u většiny

her je podstatně menší než u mračna bodů, které potřebuji zpracovávat já. Naopak využívají fotorealistické efekty, mapř. různé odlesky, efekty osvětlení, které já nevyužiju. Herní engine neobsahují algoritmy, které se využívají při práci s mračny bodů. Případné rozšíření funkcionality by bylo velmi náročné. Enginy používají vysokoúrovňové skriptovací jazyky, které nejsou primárně optimalizovány pro rychlost. Takže by se mohlo stát, že v případě přidávání dalších funkcí bych mohl narazit na limity schopností těchto enginů. Dalším problémem by mohlo být přidávání knihoven. Herní enginy mají navíc licenční podmínky, které by nás mohly limitovat. Proto jsem zvolil cestu naprogramování vlastního enginu.

2 VÝBĚR DATOVÉHO FORMÁTU A IMPLEMENTACE PROGRAMU

2.1 Datový formát

Existuje velké množství datových formátů pro ukládání mračen bodů, proto nebylo možné se všemi detailně zabývat. Na základě požadavků aplikace vyplynuly následující kritéria:

První kritérium je široká podpora různých programů, které mohou být zdrojem dat pro mou aplikaci. Tím eliminujeme nutnost dalšího programu, který by sloužil pouze k převodu formátu na mnou podporovaný.

Druhé kritérium je flexibilita formátu. Některé formáty podporují jen pevně dané vlastnosti, což se časem může negativně projevit.

Třetí kritérium je jednoduchost integrace podpory tohoto formátu.

2.1.1 Textový formát

Tento formát ukládá všechna data jako text. První řádek těchto formátů je hlavička, která udává jaké vlastnosti a v jakém pořadí v souboru jsou. Seznam některých vlastností:

- X,Y,Z jako souřadnice bodů
- R,G,B jako barva bodu v rozsahu 0 až 255
- Rf,Gf,Bf jako barva bodu v rozsahu 0 až 1
- Nx,Ny,Nz jako vektor normály bodů

Některé formáty mají na druhém řádku údaj o počtu bodů v souboru. Jednotlivé body jsou pak v souboru odděleny novým řádkem. Tento formát bodů je široce rozšířený. Existuje několik variací tohoto formátu, většinou ale mají jen malé odlišnosti. Jedna z odlišností je oddělovač sloupců, může to být mezera, tabulátor, čárka nebo středník. Nevýhodou tohoto formátu je pomalé zpracování a neefektivní ukládání. Je to zatím jediný podporovaný formát mého programu.

2.1.2 Stereo Lithography STL

Tento formát se používá hlavně pro 3D tisk, podporuje pouze povrchovou geometrii objektu bez barvy, textury nebo jiných běžných CAD atributů, proto je pro mračna bodů i meshe nevhodný.

2.1.3 Wavefront OBJ

Tento formát popisuje pouze geometrii objektu, tj. vrcholy, texturovací souřadnice, normály bodů a stěny. Materiály, které popisují vizuální stránku objektu, jsou v externím .mtl souboru. To je soubor, který definuje světloodrazivé vlastnosti pro počítačovou grafiku. Tento formát je nevhodný jak pro mračno bodů, tak pro mesh, protože se při vykreslování zpravidla používá pouze barva.

2.1.4 Polygon file format PLY

Tento datový formát podporuje jak mračna bodů, tak i meshe. Formát také dovoluje uložit vlastnosti jako barvu, normály a také uživatelem definované parametry. Podporuje široké spektrum datových typů: 1 až 4 bytové celočíselné jak znaménkové, tak neznaménkové, a plovoucí řádovou čárku jak v jednoduché, tak ve dvojité přesnosti.

Typický soubor obsahuje seznam vrcholů v x,y,z formátu a seznam stěn jako indexy do seznamu vrcholů.

Struktura tohoto souboru se skládá z hlavičky a vlastních dat. Hlavička je seznam řádků textu, které popisují data v souboru. Jednotlivé parametry jsou určeny klíčovým slovem `element`, za nímž následuje popis vlastnosti a seznam vlastností tohoto elementu. Tyto jsou pak uvedeny klíčovým slovem `property` za nímž následuje jeho datový typ a název.

Je široce podporován (Blender, CloudCompare . . .) a je rozšiřitelný. Existuje ve dvou verzích, binární a textové.[17]

2.1.5 Point Cloud Data

Tento formát vznikl jako součást Point Cloud Library[8][22]. Existuje v textové i binární verzi. Jeho textová verze má velmi podobnou strukturu jako ostatní textové formáty.

Jeho hlavní výhoda je podpora organizovaných mračen. To jsou mračna, jejichž body představují nejaký obraz nebo matici, kde se data rozdělují na řádky a sloupce. Příklady takovýchto mračen jsou například data z TOF kamer. Výhoda spočívá ve znalosti vztahů mezi sousedními body, čímž se zrychlí operace s nejbližšími body.

Integrace tohoto formátu je jednoduchá, protože by se využila celá knihovna knihovna.

2.1.6 Finální výběr

Z výše uvedených datových formátů se mi jeví jako nejlepší datový formát PLY, protože podporuje jak mračna, tak meshe a podporuje vlastní parametry.

2.2 Engine

Tento engine je napsán v C++ a jsou používány pouze svobodné knihovny. Aplikace je koncipována jako multiplatformní, momentálně je schopná běžet na PC s OS Microsoft Windows a GNU/Linux.

Aby se engine dal spustit i pod OS GNU/Linux, musel jsem ako 3D API použít OpenGL[6], v mém případě ve verzi 3.3 Core Profile. Pro kontrolu funkcí OpenGL za běhu, používám knihovnu The OpenGL Extension Wrangler Library [7]. Tato knihovna je pod modifikovanou BSD licencí[4], Mesa 3-D licencí[5] a Khronos licencí[3]. Pro vytvoření okna, kontextu a snímání uživatelského vstupu je použita knihovna GLFW[2] verze 3, tato knihovna je pod licencí zlib/libpng[15].

2.2.1 Požadavky virtuální reality

Protože je engine primárně určen pro vykreslování ve virtuální realitě, jsou na něj kladeny jisté požadavky. Aby vjem virtuální reality nebyl rušivý nebo dokonce nepříjemný je nutné dodržet dostatečnou snímkovací frekvenci a velmi krátkou odezvu na povely operátora. Proto je nutné se věnovat výkonové optimalizaci.

2.2.2 Vykreslování bodů

Nejprve bylo nutné vyřešit způsob vykreslení bodů. Pokud je budeme vykreslovat jako body s nějakou velikostí, budou na kameru vždy natočeny kolmo a budou mít vždy stejnou velikost nehledě na vzdálenost od kamery a to bude mít negativní dopad na výsledný vjem operátora. Proto jsem se rozhodl vykreslovat je jako malé krychle. To vyřeší problém hloubky obrazu a zlepší výsledný vjem, za cenu zvýšeného počtu.

Nejjednodušší způsob vykreslení krychle je vykreslovat každou stěnu zvlášť.

Místo kreslení čtyřúhelníku se ale kreslí po dvou trojúhelnících, protože u trojúhelníků jsou všechny body vždy v rovině a toto může ovlivnit osvětlení. Další problém čtyřúhelníků je problematická lineární interpolace mezi jednotlivými body, což negativně ovlivní obarvení nebo otexturování. Takto se počet vrcholů zvýší 36 krát.

Teoreticky je rychlost vykreslování velmi závislá na počtu vrcholů a proto jsem začal uvažovat o optimalizaci. Pokud se budou krychle vykreslovat maximálně optimalizované, pomocí dvou trojúhelníkových vějířů, tak se počet vrcholů pouze zšestnáctinásobí. V tom případě se ale musíme vzdát normál, které se používají pro simulaci osvětlení, protože v případě krychle je jeden vrchol společný pro tři stěny tudíž jsou potřeba 3 normály na vrchol. Pokud se normál nechceme vzdát, musíme každou stěnu kreslit odděleně, což lze dvěma způsoby. První a jednodušší je aktuální

stav. Druhý způsob je kreslení stěn jako trojúhelníkového pruhu, kde počet vrcholů naroste 24 krát.

Tuto teorii jsem se pokusil experimentálně ověřit. Výsledek byl opačný od očekávání. V případě nulové optimalizace se testovací scéna vykreslovala zhruba 117 snímků za vteřinu, v případě trojúhelníkových pruhů zhuba 63 a v případě vějířů 98. Přesné vysvětlení pro toto nemám, nicméně všechny zmínky na internetu naznačují[9][10], že by mohlo jít o optimalizaci na úrovni hardwaru grafických karet.

2.2.3 Omezení procesorové zátěže

Dále je nutné omezit veškterá OpenGL volání, protože jejich režie je nezanedbatelná. Proto je nutné se vyhnout všem zastaralým technikám vykreslování. Pokud bych například vykresloval bod po bodu v cyklu byl by zatěžován procesor mnohem více než grafická karta a výkon by byl velmi nízký. Proto je nutné využívat prvky jako Vertex Buffer Object[12] a Vertex Array Object[11].

Protože se krychle skládá z více vrcholů některé vlastnosti, jako barva či skalární pole, by se pro každý vrchol opakovali. Toto opakování jsem eliminoval geometrickým instancováním. Je to způsob jak nakreslit mnoho opakujících se objektů (body ve formě krychlí) s lišícími se parametry (pozice, barva). Do paměti grafické karty se načte pouze jedna krychle a pak seznam pozic, barev popřípadě dalších parametrů. Postupně se pak pro všechny parametry v seznamu kreslí krychle.

2.2.4 Podporovaný formát

Kvůli jednoduchosti jsem na začátku implementoval pouze omezenou podporu textového formátu. Počáteční implementace byla velmi jednoduchá ale také velmi pomalá. Testovací soubor 50 milionů řádků na i7-6700k při taktu 4.5GHz trval načíst 78.3s. Doba načítání souboru mě brzdila při vývoji, protože jsem musel při každém spuštění čekat na načtení souboru. Toto vyústilo v optimalizaci.

Původní implementace využívala pro čtení `std::ifstream`, řádky se načítaly v cyklu pomocí `std::getline` a pomocí `std::stringstream` a operátoru `»` se jednotlivé sloupce přímo převáděly na `float`. Všechna data se pak ukládala do `std::vector`.

Pak jsem si ověřil že značnou část času trvá převod řetězce na `float`. Když jsem přestal převádět na `float` a nechával jednotlivé čísla v řetězci doba se zkrátila na 44.8s.

Při pátrání na internetu jsem našel několik zmínek o tom že C funkce jsou obecně rychlejší.

Konečná implementace je téměř v čistém C. Nejprve se ze souboru do paměti načte blok dat o aktuálně nastavené hodnotě v programu, ten se poté v cyklu dělí na řádky pomocí `strchr` a tento řádek se pak ve vnořeném cyklu dělí na jednotlivé čísla pomocí `strtok`. Tyto čísla se ukládají v řetězci do dvourozměrného pole. Jakmile se v bloku nenajde další prázdný řádek, tak se k tomuto bloku přidá další načtený blok. Poté se spustí paralelně na ostatních jádrech procesoru konverze řetězců na `float` a zároveň se v hlavním vlákně rozděluje aktuálně načtený blok. Této implementaci trvá na stejném PC načíst stejný soubor pouze 43.9s.

2.2.5 Segmentace mračna

Mračna, která mají přes jeden milion bodů, jsou příliš náročná na vykreslování, proto je nutné sáhnout po optimalizaci. Velmi často není celé mračno viditelné na obrazovce. Díky tomu lze využít tzv. frustum culling[14], což je technika, která nám umožnuje zjistit, které objekty, nejsou v záběru pohledu, tudíž není nutné je vykreslovat. Aby toto bylo možné je nutné mračno rozdělit na segmenty. Při segmentaci se mračno rozdělí do tzv. octree, což je datová struktura, kde každý uzel má právě 8 potomků. To se velmi hodí při rozdělování třírozměrných objektů, kde se objekt rozdělí na poloviny podle všech rozměrů. Rozděluje se podle počtu bodů v jednom segmentu, pokud je v segmentu víc bodů, než aktuálně v programu nastavená hodnota, segment se rozdělí na 8 podsegmentů. Toto se provádí rekurzivně pro všechny segmenty.

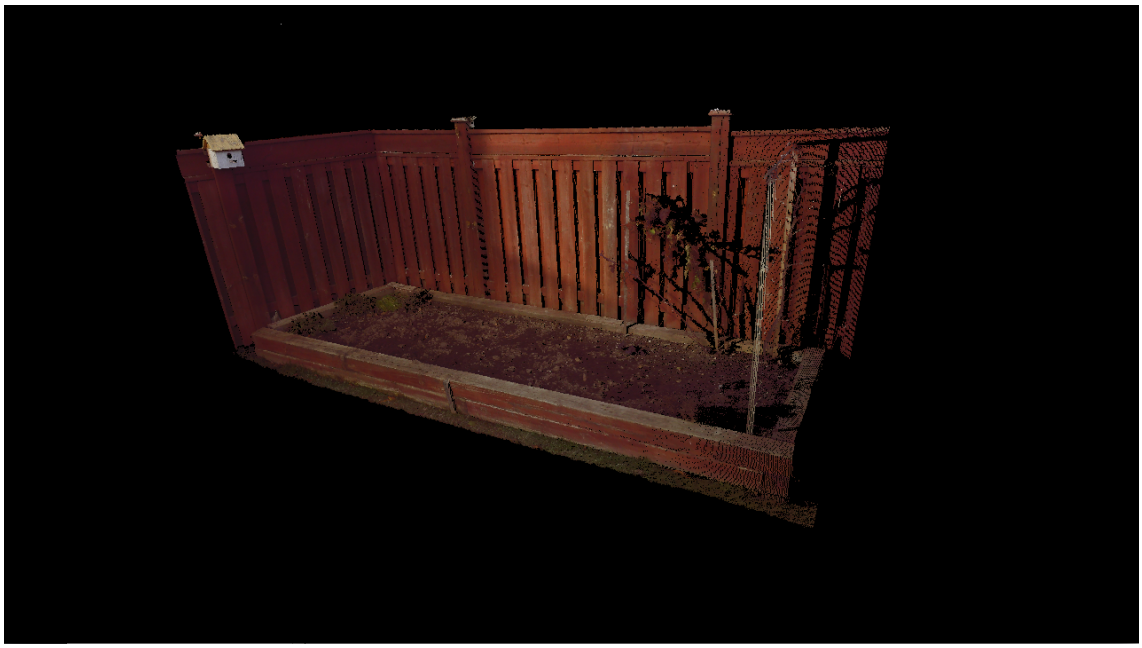
2.2.6 Grafické rozhraní

Samotné OpenGL umí jen vykreslovat základní tvary a také nemá žádnou podporu vykreslování písem. Toto dělá tvorbu grafického rozhraní poměrně složitou a to beru v úvahu pouze vzhled. Jakákoli funkčnost tohoto rozhraní už jde mimo OpenGL úplně.

Existuje řada knihoven, které tuto funkcionalitu doplňují. Většina jsou buď neznámé malé neudržované projekty a nebo velmi komplexní knihovny, které naopak příliš zasahují do samotné funkčnosti zbytku programu.

Vybrat vhodnou knihovnu je velmi obtížné, protože pro zhodnocení knihovny je nutné si ji vyzkoušet což je časově náročné a informace na internetu nehodnotí příliš objektivně.

Pro tento demonstrační program jsem se rozhodl naimplementovat vlastní jednoduché grafické rozhraní.



Obr. 2.1: Pohled na načtené mračno

2.2.7 Zobrazení mračna

Změna velikosti bodů

Mračna mají různé hustoty bodů. Aby mračno nebylo příliš řídké nebo se naopak body nepřekrývaly, bylo nutné implementovat změnu velikosti těchto bodů.

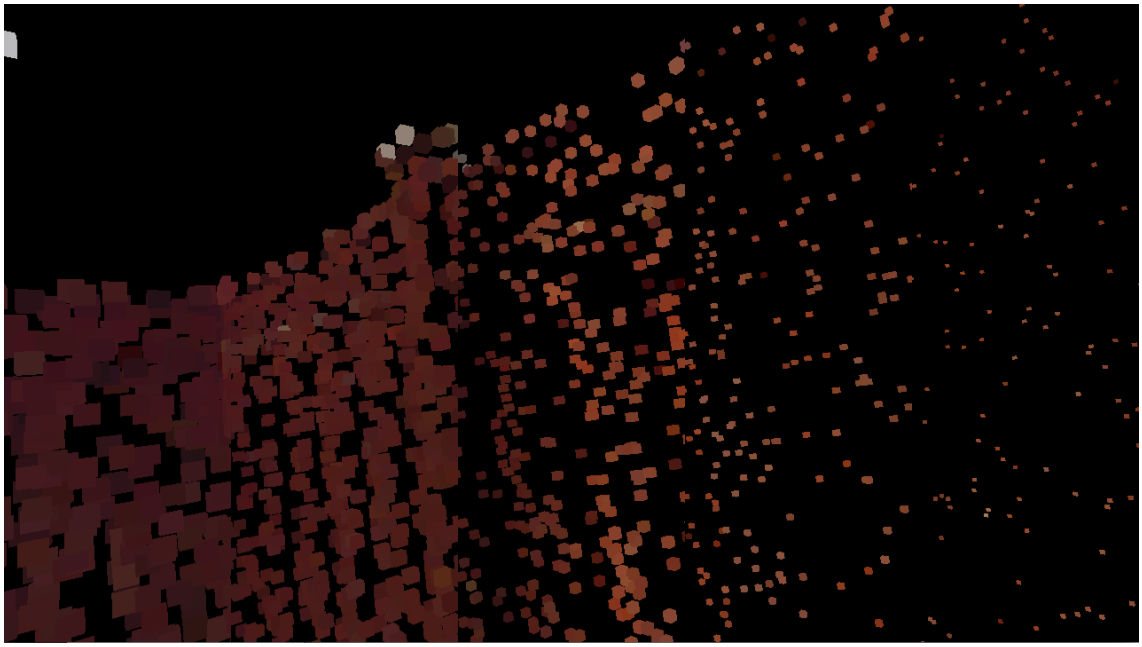
Velikost bodů se mění skokově podle funkce 2^n, kde n je celé číslo viz obrázek 2.2. Počáteční hodnota tohoto čísla je -8.

Zobrazení skalárního pole

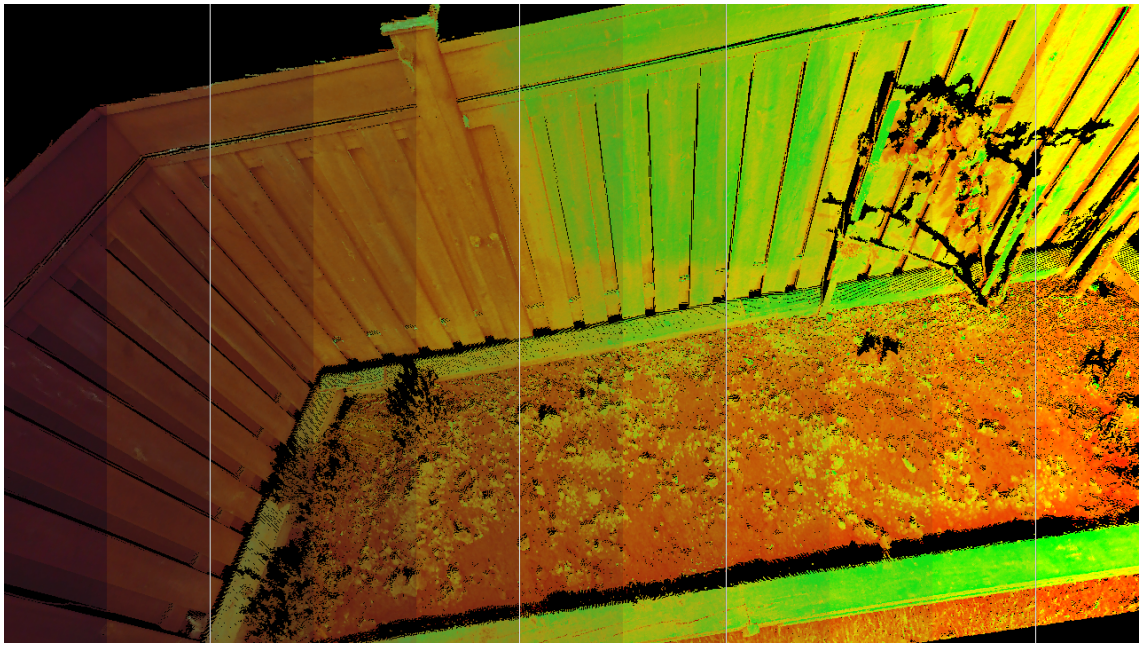
Některá mračna mají uloženo tzv. skalární pole. To znamená, že ke každému bodu je přiřazeno číslo. Toto číslo má po domluvě nějaký význam, například teplota.

Aby se dalo toto pole zobrazit je nutné ho převést na barevnou škálu. Já ho převádím tak že celý rozsah tohoto pole upravím na rozsah 0 až 120. Tuto hodnotu poté použiji jako odstín HSV barevného modelu s maximální sturací a hodnotou. Tuto barvu pak převedu do RGB jakodruhou možnou barvu bodu.

Tuto výslednou barvu lze poté prolínat s originální barvou podle alpha blendingu[1] viz obrázek 2.3.



Obr. 2.2: Demonstrace změny velikosti bodů



Obr. 2.3: Demonstrace přimíchávání škály skalárního pole

2.3 Ovládání programu

Veškeré ovládání programu bylo nejprve implementováno pomocí klávesnice.

Pomocí kláves W a S se pohled posouvá po ose Z, A a D po ose X a Alt a mezerník po ose Y. Pomocí šipek nahrou a dolů se pohled natáčí podle osy X, pomocí šipek doleva a doprava podle Y a pomocí kláves Q a E se naklápí podle osy Z.

Klávesou R se velikost bodu zdvojnásobí, klávesou F sníží na polovinu. Klávesami T a G se plynule přechází mezi barvou mračna a skalárním polem.

Klávesou M se vyvolá menu. Šipkami nahoru a dolů se vybírají jednotlivé prvky v menu, výběr se potvrdí klávesou Enter, pro vrácení o úroveň nahoru klávesa Escape.

Program se uzavře stisknutím Escape mimo menu.

Po vyvolání menu lze vidět položku objects, viz obrázek 2.4, v ní je seznam všech načtených mračen do programu a položku load (obrázek 2.5), pomocí ní lze načíst mračno. Pokud operátor vybere již načtené mračno, bude mít v nabídce možnost toto mračno odstranit.

Poté jsem implementoval částečné ovládání pomocí pravého ovladače.

Pro pohyb po scéně je nejprve nutné stisknout trigger, poté táhnout ovladačem a nakonec trigger uvolnit. Stisknutím horního tlačítka na pravém ovladači se vyvolá menu. Pohybem prstu po trackpadu se posouvá položkami v menu, triggerem se výběr potvrdí. Horním tlačítkem se také vrací na vyšší úroveň menu a menu uzavírá.

Obr. 2.4: Hlavní menu

Obr. 2.5: Menu objekty

3 ZÁVĚR

Jako datový formát jsem vybral PLY, pro svou flexibilitu. V případě, že by se využila funkcionalita Point Cloud library by se využil i formát PCD.

Aplikace svůj demonstrační účel splnila. Lze načítat nekolik mračen bodů, pohybovat se v prostoru, měnit velkosti bodů a zobrazit skalární pole. Hardwarové nároky počítače jsou závislé pouze na složitosti načteného mračna.

Pro další vývoj aplikace bude vhodné přehodnotit celkový návrh a vybrat vhodnou knihovnu pro grafické rozhraní, aby bylo možné implementovat lepší uživatelské rozhraní, díky kterému bude možné pohodlné ovládání po přidání dalších funkcí a také získat uživatelskou zpětnou vazbu pro zkvalitnění

Další vylepšování by se mohlo týkat výkonu. Pro virtuální realitu se musí vykreslovat vždy dva nezávislé snímky. Tyto snímky by mohly vykreslovat dvě grafické karty. V tomto by velmi pomohlo použití nových API jako je Vulkan nebo DirectX12, které jsou na tuto situaci lépe navrženy než předchozí API.

LITERATURA

[1] Alpha Blending.
URL <https://en.wikipedia.org/wiki/Alpha_compositing#Alpha_blending>

[2] GLFW.
URL <http://www.glfw.org>

[3] Khronos Licence.
URL <http://glew.sourceforge.net/khronos.txt>

[4] Licence GLEW.
URL <http://glew.sourceforge.net/glew.txt>

[5] Mesa 3-D Licence.
URL <http://glew.sourceforge.net/mesa.txt>

[6] OpenGL.
URL <https://www.opengl.org>

[7] The OpenGL Extension Wrangler Library.
URL <http://glew.sourceforge.net/index.html>

[8] Point Cloud Library.
URL <http://pointclouds.org/>

[9] Pre and post-T&L vertex caches optimization.
URL <http://gamedev.stackexchange.com/a/34130>

[10] To Strip or Not To Strip.
URL <http://hacksoflife.blogspot.cz/2010/01/to-strip-or-not-to-strip.html>

[11] Vertex Array Object.
URL <https://www.khronos.org/opengl/wiki/Vertex_Specification#Vertex_Array_Object>

[12] Vertex Buffer Object.
URL <https://www.khronos.org/opengl/wiki/Vertex_Specification#Vertex_Buffer_Object>

[13] VFX1.
URL <https://en.wikipedia.org/wiki/VFX1_Headgear>

[14] View Frustum Culling.
URL <http://www.lighthouse3d.com/tutorials/view-frustum-culling/>

[15] zlib/libpng licence.
URL <https://opensource.org/licenses/zlib-license.php>

[16] Binstock, A.: Powering the Rift. 5 2015.
URL <https://www3.oculus.com/en-us/blog/powering-the-rift/>

[17] Bourke, P.: PLY - Polygon File Format.
URL <http://paulbourke.net/dataformats/ply/>

[18] Buckley, S.: This Is How Valve's Amazing Lighthouse Tracking Technology Works. 5 2015.
URL <http://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-tec

[19] Ježek, D.: Khronos Group vyvine univerzální VR standard. 12 2016.
URL <http://diit.cz/clanek/khronos-group-vyvine-univerzalni-vr-standard>

[20] Kelion, L.: HTC reveals virtual reality headset with Valve at MWC.
URL <http://www.bbc.co.uk/news/technology-31664948>

[21] Maiberg, E.: Valve and HTC reveal Vive VR headset.
URL <http://www.gamespot.com/articles/valve-and-htc-reveal-vive-vr-headset/1100-6425606/>

[22] Rusu, R. B.; Cousins, S.: 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.