VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

REKONSTRUKCE POVRCHU Z MRAČNA BODŮ

SURFACE RECONSTRUCTION FROM POINT CLOUDS

BAKALÁŘSKÁ PRÁCE

BACHELOR´S THESIS

AUTOR PRÁCE JAN HORÁK
AUTHOR

VEDOUCÍ PRÁCE ING. ROSTISLAV HULÍK
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá rekonstrukcí povrchu z mračna bodů. Popisuje návrh, implementaci a testování metody založené na detekci rovin a Delaunayho triangulaci. Aplikace umožňuje zrekonstruovat povrch ve formě trojúhelníkové sítě.

Klíčová slova

mračno bodů, rekonstrukce povrchu, trojúhelníková síť, polygonální model, PCL, C++

Abstract

This thesis deals with the reconstruction of the surface from point clouds. It describes the design, implementation and tests of method based on the detection of planes and Delaunay triangulation. The application allows to reconstruct the surface in the form of a triangle mesh.

Keywords

point cloud, surface reconstruction, triangle mesh, polygonal model, PCL, C++

Citace

Jan Horák: Rekonstrukce povrchu z mračna bodů, bakalářská práce, Brno, FIT VUT v Brně, 2013

Rekonstrukce povrchu z mračna bodů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rostislava Hulíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Horák
10. května 2013

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu mé bakalářské práce Ing. Rostislavovi Hulíkovi za odbornou pomoc, cenné rady a vynaložené úsilí a čas, který mé práci poskytl.

Obsah

1 Úvod

Jak jsem zjistil v průběhu posledního roku, kdy jsem pracoval na své bakalářské práci, pojem rekonstrukce povrchu z mračna bodů zní pro většinu lidí tajemně a netuší, co se pod ním skrývá. Alespoň tedy poté, co ho uslyší. Když se nad tím trochu zamyslí, většinou je napadne, že cílem mé bakalářské práce je vzít spoustu bodů a vytvořit z nich nějaký model či něco podobného. Což je celkem přesný popis toho, čím se v mé práci zabývám.

Mračna bodů skutečně nejsou ničím jiným než určitou množinou bodů v prostoru. V této chvíli běžného člověka nejspíš napadnou myšlenky jako odkud se ta mračna berou, k čemu jsou či jaký to má vlastně všechno smysl? Proč vlastně rekonstruovat povrch z bodů, proč nevzít rovnou povrch? Na tyto a snad i na mnohé další otázky vám odpoví moje práce.

Práce je rozdělena do šesti hlavních kapitol. Ihned po úvodu následuje kapitola rozbor, ve které se snažím uvést čtenáře do tématu práce tak, aby mu porozuměl i v případě, že o podobném tématu čte poprvé. V této kapitole jsou též odkazy na spoustu dalších zajímavých informací, na články, kde jsou podrobně rozepsány témata, jež zde třeba jen okrajově zmiňuji apod. Třetí kapitolou je návrh mé aplikace. V této kapitole se nachází seznam vytyčených cílů pro aplikaci a seznam prostředků, které jsem se rozhodl využít pro jejich dosažení. Dá se v ní nalézt podrobný popis algoritmu mé práce stejně jako informace o vstupech výstupech či programovacím jazyku. Následuje kapitola implementace, ve které je program popsán z hlediska konkrétní implementace. Její součástí je podkapitola o použitých knihovnách a poté kompletní rozbor mé aplikace a informace o implementování funkčnosti. Předposlední kapitolou je kapitola testování a výsledky, ve které se věnuji návrhy testů, testovaným datům a samozřejmě výsledkům testů. Poslední kapitolou je samotný závěr, kde shrnuji mé poznatky a zkušenosti z bakalářské práce a předhazuji možnosti dalšího vývoje aplikace. Přeji příjemnou četbu.

2 Rozbor

Tato kapitola uvede čtenáře do problematiky rekonstrukce povrchu z mračna bodů, vysvětlí pojmy vyskytující se v této oblasti a načrtne některé způsoby řešení.

2.1 Mračno bodů

Mračno bodů (v angličtině Point Cloud) [1] je soubor vrcholů v trojrozměrném souřadném systému. Vrcholy jsou většinou popsány pomocí jejich x, y a z souřadnic. Mračno bodů nejčastěji reprezentuje vnější povrch objektů. Na obrázku 2.1 je zobrazeno mračno bodů v programu MeshLab [2].

Obrázek 2.1: Ukázka mračna bodů

Mračna bodů jsou jsou často vytvořeny pomocí 3D scannerů [3]. Tato zařízení zkoumají povrch objektů z důvodu sběru informací pro tvorbu modelu. Tyto informace mohou být například tvar nebo barva. Mračno bodů bývá výstupním datovým typem některých 3D scannerů. Znázorňuje pak informaci o bodech na povrchu, které scanner naměřil.

Mračno bodů si můžete vytvořit i doma. Stačí vám k tomu pouze fotoaparát či kamera a vhodný program, který dokáže najít body společné na více fotografiích. Takovým programem může být například program Bundler [4].

Mračna bodů se používají například v CAD modelech, metrologii či při vizualizaci existujích objektů.

Mračna bodů se dělí na neuspořádaná a uspořádaná. Neuspořádaná mají pouze informace o x, y a z souřadnicích bodů, zatímco uspořádaná znají i některé další informace, například 2d souřadnice bodů v různých snímcích. Moje aplikace se bude věnovat neuspořádaným mračnům. Pro uspořádaná mračna můj algoritmus funguje taktéž, ale informací navíc, které jsou v uspořádaných mračnech uloženy, nijak nevyužívá.

Dalším dělením je dělení na hustá a řídká mračna. Hustá mračna obsahují více bodů a povrch se z nich snadněji rekonstruuje. Mračno na obrázku 2.1 je typickým řídkým mračnem, obsahuje pouze vrcholové body.

2.1.1 Formáty souborů

Mračna bodů mohou být uchovávána v mnoha různých textových či binárních datových typech. Ty textové většinou obsahují přímo uložené x, y a z souřadnice bodů. Nejjednodušším textovým formátem, se kterým se lze občas setkat, je formát xyz, který obsahuje právě jen souřadnice. Tento formát není příliš vhodný, ostatní formáty mají většinou hlavičku obsahující informace o počtu bodů, jejich typu a podobně.

Jedním z těchto formátů je formát pcd (point cloud data) [5]. V hlavičce formátu specifikujeme například počet bodů, informace o bodech (kromě standardních souřadnicí můžeme mít uloženy i některé další jako například barvy či normály k povrchu objektu).

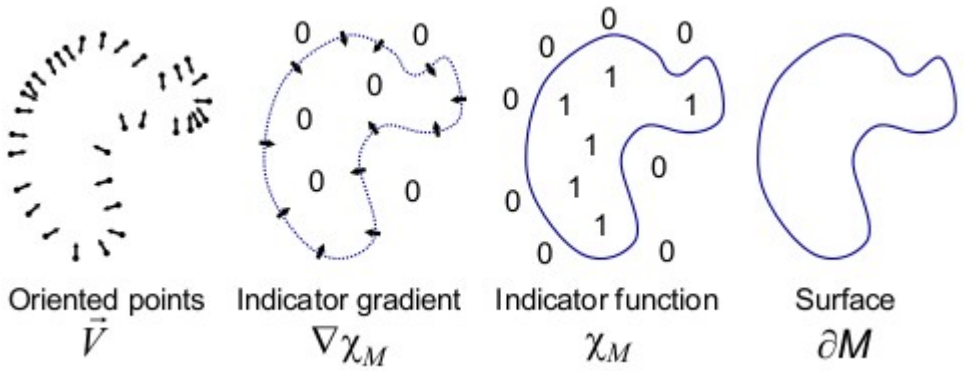
Dalším formátem na ukládání mračen bodů může být například formát ply (polygon file format) [6], který je sice primárně určen k ukládání mnohoúhelníkových sítí, nicméně může nést informaci pouze o vrcholech. Obecně se dá použít jakýkoliv formát, ve kterém je možnost ukládat souřadnice samostatných vrcholů.

2.2 Rekonstrukce povrchu

Ve většině aplikací nelze použít přímo mračna bodů, ale je potřeba nejprve zrekonstruovat povrch například do trojúhelníkové (či obecně mnohoúhelníkové) sítě nebo NURBS povrchu. Existuje spoustu různých metod pro rekonstrukci povrchu z mračna bodů, jež jsou postavené na různých algoritmech.

Jednou používanou metodou je algoritmus power crust [7], jehož výhodou je vytváření vodotěsných modelů. Algoritmus nejprve aproximuje střední osy transformace (v angličtině medial axis transform, zkratka MAT), což jsou body, které mají více než jeden nejbližších hraničních bodů, a poté provede inverzní transformaci, kterou vytvoří povrch z této struktury.

Dalším algoritmem je Poissonova rekonstrukce [8]. Tento algoritmus si poradí i s neúplnými či nekompletními daty, potřebuje však orientované mračno bodů. Tato metoda převádí rekonstrukci na Poissonovu rovnici, tj. vypočítat skalární funkci x, jejíž Laplacián (divergence gradientu) se rovná divergenci vektorového pole V. Tento algoritmus je zobrazen na obrázku 2.2.



Obrázek 2.2: Poissonova rekonstrukce

Posledním zde popisovaným algoritmem je algoritmus MPU (Multi-level Partition of Unity Implicits) [9]. Tento algoritmus spočívá ve třech krocích. Tou první je po částech kvadratická funkce, která zachycuje lokální tvar povrchu. Následují váhové funkce, které tyto části prolínají a spojují, a na závěr metoda dělení založená na oktárních stromech, která přizpůsobuje lokální tvary celkovému povrchu.

3 Návrh

Tato kapitola si klade za cíl přiblížit čtenáři návrh algoritmu pro rekonstrukci povrchu z mračna bodů, který je v této práci řešen.

3.1 Cíl aplikace

Zadáním mé aplikace bylo rekonstruovat povrch z mračna bodů. Hlavní otázka zněla, co bude výstupním typem dat, zda trojúhelníková síť, obecně mnohoúhelníková síť či ještě něco jiného. Nakonec jsem se rozhodl pro trojúhelníkovou síť, především proto, že jejím výstupem jsou pouze konvexní mnohoúhelníky a výstup se pak lépe zpracovává.

3.1.1 Typ aplikace

U projektu bylo potřeba rozhodnout, zda se bude jednat o zásuvný modul pro nějakou již existující aplikaci či zda se bude jednat o zcela samostatný program. Výhodou samostatné aplikace je především možnost návrhu dle vlastních potřeb a také to, že autor nemusí studovat dokumentaci aplikace, do které zásuvný modul dělá. Mezi výhody zásuvného modulu patří především fakt, že autor nemusí implementovat základ aplikace a může se soustředit přímo na jádro problému. Z aplikací, pro které bych mohl tvořit zásuvný model, přicházely v úvahu zejména OpenFlipper [10] a MeshLab, já jsem se ale nakonec rozhodl pro vlastní aplikaci.

Dalším důležitým faktorem bylo uživatelské rozhraní programu. Přicházelo v úvahu jako grafické uživatelské rozhraní, tak pouze konzolová aplikace. Rozhodl jsem se udělat pouze konzolovou aplikaci a soustředit se spíše na problémy spojené s vlastním algoritmem. Hlavním důvodem byl fakt, že není příliš pravděpodobné setkání běžného uživatele s mojí aplikací. Předpokládám, že uživatelé, kteří rekonstruují povrch z mračna bodů jsou schopni spolupracovat s konzolí. Nicméně uživatelské rozhraní může být součástí dalšího rozvoje aplikace.

3.1.2 Požadavky na aplikaci

Jedním z důležitých bodů návrhu jsou i požadavky na aplikaci. Tyto požadavky by měly být určeny předem a neměly by se v průběhu implementace měnit. Požadavky na moji aplikaci jsou tyto:

* Aplikace zrekonstruuje povrch z mračna bodů
* Vstupem aplikace bude mračno bodů, výstupem trojúhelníková síť
* Aplikace bude závislá pouze na použitých knihovnách
* Aplikace bude multiplatformní

* Aplikace bude v průběhu uživatele informovat o probíhajících úkonech

3.2 Typy vstupních a výstupních dat

Jednou z otevřených otázek mé aplikace bylo, jaké budou formáty možných vstupních a výstupních dat. Snažil jsem se brát v potaz rozšířenost jednotlivých formátů a jejich snadnou čitelnost.

3.2.1 Vstupní data

Formátů, ve kterých mohou být uložena mračna bodů, je skutečně mnoho, jak již bylo zmíněno v kapitole 2.1.1. Při mém rozhodování hrála roli rozšířenost formátu a snadnost konverze na tento formát. Hledal jsem tedy vhodný textový formát, protože ten je znalý uživatel schopný přepsat či vytvořit i ručně, pokud chybí nějaký automatizovaný nástroj. Nakonec jsem se rozhodl pro textový formát pcd, který ukládá body standardním způsobem zápisu jejich x, y a z souřadnic, před nímž je krátká hlavička zapsaná v tabulce 3.1.

Název pole	Specifikace pole	Příklad
Version	Verze pcd	# .PCD v.7 - Point Cloud Data file format
Fields	Názvy dimenzí	FIELDS x y z
Size	Velikost každé dimenze v bytech	SIZE 4 4 4
Type	Typ každé dimenze jako char	TYPE F F F
Count	Počet elementů každé dimenze	COUNT 1 1 1
Width	Výška (u organizovaných mračen, u neorganizovaných počet bodů)	WIDTH 486
Height	Šířka (u organizovaných mračen, u neorganizovaných vždy 1)	HEIGHT 1
Viewpoint	Získaná pozice pozorovatele	VIEWPOINT 0 0 0 1 0 0 0
Points	Počet bodů v mračnu	POINTS 486
Data	Typ dat (textová nebo binární)	DATA ascii

Tabulka 2.1: Hlavička formátu pcd

Jak je vidět z tabulky, v případě znalosti souřadnic bodů není žádný problém tuto tabulku vytvořit ručně. Tento formát má i další výhodou související s volbou knihoven, která je popsána v podkapitole 4.1.

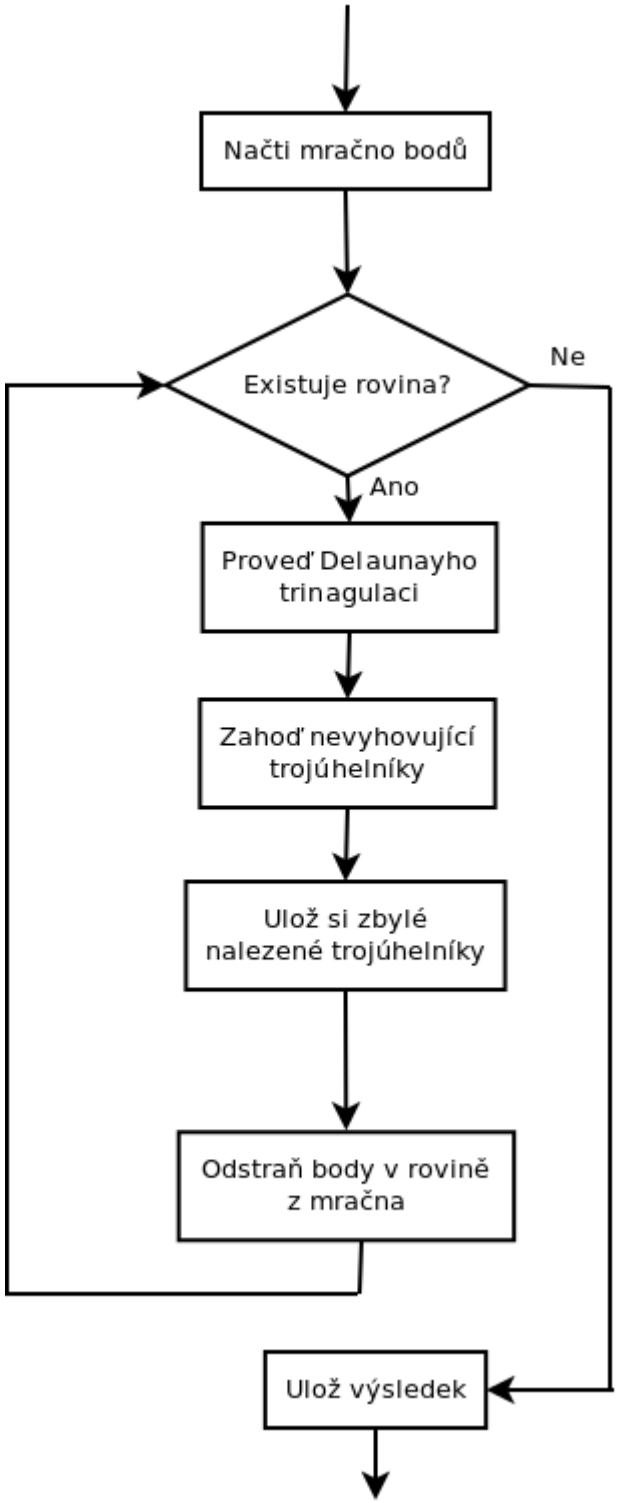
3.2.2 Výstupní data

Prioritou u formátu výstupních dat byla především jeho rozšířenost a podpora v aplikacích. Moje první volba padla na formát tri, který je určen přímo na trojúhelníkové sítě, zjistil jsem však, že některé aplikace pro zpracování sítí tento formát nepodporují. Proto jsem se nakonec rozhodl pro formát ply, který je určen pro obecně mnohoúhelníkové sítě. Formát ply je podobně jako pcd textový (pcd může být i binární, ale já používám pouze jeho textovou verzi) a jdou z něj snadno vyextrahovat potřebná data. Ply ukládá síť ve dvou seznamů. V prvním seznamu jsou souřadnice vrcholů, ve druhém seznamu jsou pak odkazy na vrcholy z prvního seznamu, které jsou ve společném mnohoúhelníku.

3.3 Algoritmus

Nyní se již dostáváme k otázce, jakou metodu na rekonstrukci povrchu zvolit. Nejprve jsem implementoval metodu greedy search (o té bude později zmínka v kapitole 5.1.2), ale poté jsem se ze studijních důvodů rozhodl nespokojit s žádnou z rozšířených metod, ale zkusit nějakou vlastní, která sice kombinuje známé algoritmy, ale mým vlastním způsobem. Algoritmus je postaven na hledání rovných ploch v mračnu, ze kterých jsou poté vytvořeny samostatné sítě. Je jasné, že tento algoritmus může fungovat pouze pro objekty s dostatečným počtem rovin (například budovy) a nebude mít dobré výsledky pro ostatní objekty. Moje metoda má také potenciál složitá mračna zjednodušit a na rozdíl od metody greedy search přichází s globálním pohledem na mračno, což může mít své výhody i nevýhody.

Po načtení vstupních dat potřebujeme algoritmus pro vyhledání rovin v mračnu bodů. Tímto algoritmem byla zvolena metoda RANSAC [11], která je více rozepsána v podkapitole 3.3.1. Body v rovině byly převedeny na trojúhelníkovou síť pomocí Delaunayho triangulace [12], jež je popsána v podkapitole 3.3.2. Delaunayho triangulace ovšem pospojuje všechny body v konvexní obálce. To však algoritmus dělat nemá, protože tím přijdeme o případné díry či menší nerovnosti povrchu. Proto je poté do algoritmu zahrnuto ještě zahazování některých trojúhelníků, které je popsáno v podkapitole 3.3.3. Algoritmus je popsán vývojovým diagramem na obrázku 3.1.

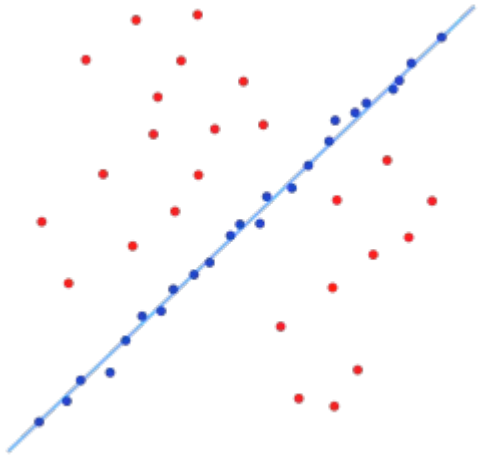


Obrázek 3.1: Vývojový diagram algoritmu

3.3.1 Hledání rovin metodou RANSAC

RANSAC (neboli Random Sample Consensus) je iterativní metoda pro detekci matematické modelu v datech. Ke své správné funkci potřebuje metoda takzvané outliers, tedy body, které nesplňují kritéria. RANSAC třídí body do dvou kategorií, jednak to jsou body vyhovující rovině, takzvané inliers, a body rovině nevyhovující, takzvané outliers. Metoda nepřináší deterministické výsledky, její výstupy jsou správné jen s určitou pravděpodobností, která se zvyšuje s počtem iterací. Na obrázcích 3.2 a 3.3 je uveden příklad použití metody RANSAC v rovině pro hledání přímky.

Obrázek 3.2: Body v rovině



Obrázek3.3: Nalezená přímka metodou RANSAC

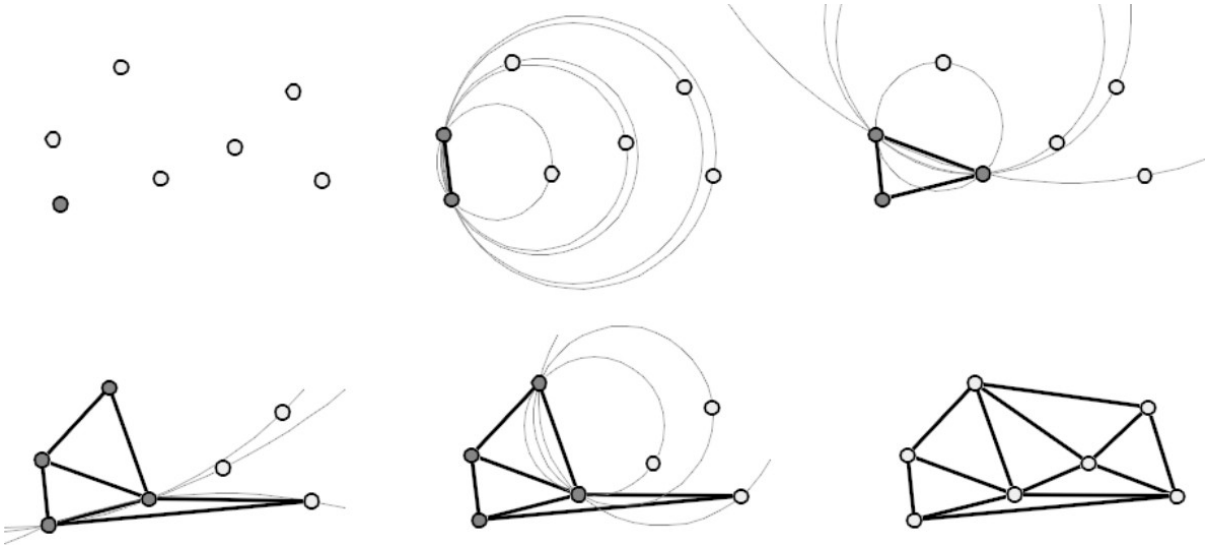
Při detekci rovin v mračnech bodů touto metodou náhodně vybereme určitý vzorek bodů a proložíme jím rovinu. Poté pro všechny zbylé body zjistíme, zda s určitou odchylkou do této roviny patří či nikoliv. Ty, které do ní patří označíme jako inliers, ty zbylé jako outliers. Tento proces opakujeme tak dlouho, jak dlouhý je počet iterací. Nakonec vybereme tu rovinu, do níž patří největší počet bodů.

U metody RANSAC je několik důležitých parametrů, které ovlivní výsledek. Tím prvním je počet iterací, čím více iterací, tím větší je pravděpodobnost správného výsledku. Druhým důležitým parametrem je minimální počet bodů v rovině (roviny obsahující malý počet bodů nás v některých případech nemusejí zajímat). Posledním důležitým parametrem je tolerance vzdálenosti inlierů od roviny. Čím větší, tím více bodů bude splňovat podmínku roviny. Při příliš vysokém čísle se může stát, že podmínku roviny budou splňovat všechny body v mračnu. Při malém čísle naopak nemusíme najít vůbec žádnou rovinu.

3.3.2 Delaunayho triangulace

Poté, co máme nalezenou rovinu, je potřeba pospojovat body tak, aby z nich vznikly trojúhelníky, Tomuto procesu se říká triangulace. Já jsem si pro svůj projekt vybral Delaunayho triangulaci. Hlavním důvodem byla její rozšířenost a existující knihovny zabývající se tímto tématem. Delaunayho triangulace tak jak ji budu používat je pouze 2D algoritmus, proto je nutné před samotným použitím triangulace rovinu převrátit tak, aby souřadnice z byla nulová.

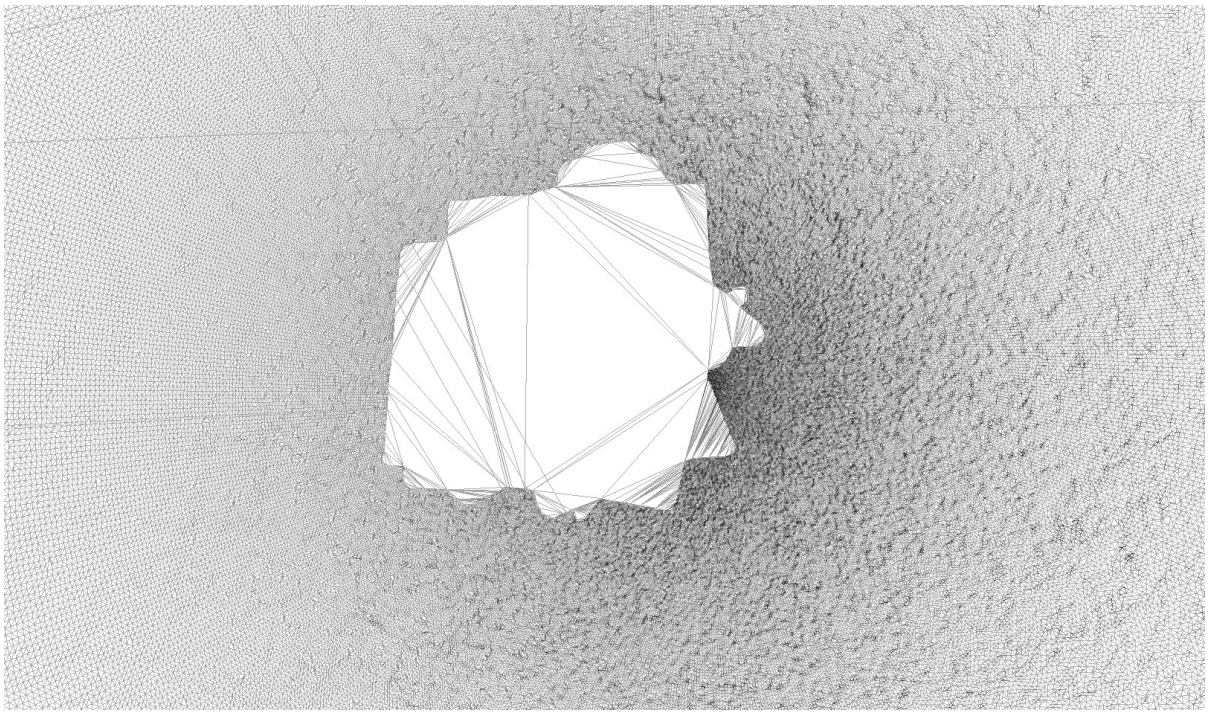
Dále budeme uvažovat pouze o bodech v nalezené rovině. Vybereme náhodný bod z mračna. K tomuto bodu nalezneme nejbližší bod a vytvoříme mezi nimi hranu. Poté inkrementálně přidáváme po jednom další body. Pokud bod leží na nějakém již existujícím vrcholu, zahazujeme ho. Pokud leží uvnitř nějakého již existujícího trojúhelníku, trojúhelník zrušíme a vytvoříme tři nové. Pokud leží na nějaké hraně, zrušíme tuto hranu a vytvoříme dva nové trojúhelníky místo jednoho předchozího. Pokud leží mimo všech trojúhelníků, hledáme opsané kružnice nad všemi potenciálními trojúhelníky složené z nově přidávaného bodu a všech existujících hran. Vybereme tu, která má nejmenší poloměr a vytvoříme zde trojúhelník. Celý proces lze pozorovat na obrázku 3.4.



Obrázek 3.4: Delaunayho trinagulace

3.3.3 Odstranění velkých trojúhelníků

Problém delaunayho triangulace spočívá v tom, že vytváří konvexní obálku nad rovinou, což ne vždy chceme. Roviny mohou obsahovat díry či nekonvexní okraje, které nám delaunayho triangulace zahrne do obálky. Proto chceme některé velké trojúhelníky odstranit. Jak takové trojúhelníky mohou vypadat je vidět na obrázku 3.5. Je zde znázorněna díra v rovině po Delaunayho triangulace bez odstranění velkých trojúhelníků.



Obrázek 3.5: Díra v rovině, na kterou byla mylně použita triangulace

Otázkou zůstává, jaké nastavit kritérium pro výběr těch trojúhelníků, které do výsledné sítě nepatří. Já jsem se nakonec rozhodl pro délku nejdelší stěny. Pokud ta přesáhne určitou mez, je trojúhelník smazán. Pravděpodobně existují i lepší kritéria, která mohou kombinovat délky stran s velikostí úhlů, případně porovnávat velikost trojúhelníku s velikostí okolních trojúhelníků. Tato vylepšení mohou být dalším rozvojem mé práce.

3.4 Volba nástrojů

Při každém návrhu aplikace vyvstane otázka volby nástrojů. Správná volba může ušetřit spoustu času u implementace. Jedná se jak o programovací jazyk, tak o podpůrné knihovny.

3.4.1 Programovací jazyk

Jednou z nejdůležitějších voleb bylo určení programovacího jazyka. Mé dosavadní zkušenosti byly především v jazycích C/C++ a Java. Ze zadání jsem si mohl vybrat mezi jazyky C/C++, C# a Python. Vzhledem k mým velmi malým znalostem jazyka C# a o něco lepším, ale stále nepříliš dobrým znalostem jazyka Python, jsem se rozhodl pro jazyk C++. Výhodou C++ je i to, že se v práci s 3D grafikou často používá, tudíž jsem neměl problém s nalezením vhodných knihoven pro podporu mého programu.

4 Implementace

Tato kapitola se věnuje popisu implementace aplikace. Popisuje použité knihovny, rozhraní aplikace a popis implementace jednotlivých částí.

4.1 Knihovny

Psaní práce takového rozsahu bez nějakých knihoven či dalších podpůrných zařízení by bylo zbytečně zdlouhavé a náročné, proto jsem se rozhodl využít ve svém programu služby dvou speciálních knihoven. Jednu pro práci s mračny bodů a jednu pro delaunayho trinagulaci.

Pro práci s mračny bodů jsem použil knihovnu PCL (point cloud library) [13]. Tato knihovna nabízí širokou škálu možností pro práci s mračny bodů a využil jsem jí téměř při všech částech mé práce.

Druhou knihovnou, kterou jsem použil, je knihovna libdel [14]. Tato knihovna je zaměřena na delaunayho triangulaci a výhodou je, že provede celou triangulaci jednou metodou. Je potřeba jen vytvořit konstruktor, kterému pošlete vektor bodů a pak nad vytvořeným objektem zavolat metodu traingulate.

4.2 Uživatelské rozhraní aplikace

Moje práce je konzolovou aplikací. Uživatel zadává data pouze jako argumenty příkazové řádky, v průběhu práce aplikace již nemůže nic ovlivnit. Je však informován o jednotlivých krocích aplikace. Parametrů pro příkazovou řádku je celkem osm (včetně nápovědy).

4.2.1 Parametry ovlivňující soubory

Prvním druhem parametrů jsou parametry ovlivňující soubory, se kterými bude aplikace pracovat. Jsou celkem tři a to -i, -o a -f. Za každým z těchto argumentů následuje absolutní či relativní cesta k souboru.

- **i** – Značí vstupní soubor ve formátu pcd. Pokud tento parametr není zadán, přiřadí se implicitní hodnota „input.pcd“.
- **o** – Výstupní soubor, do kterého budou vepsána data ve formátu ply. V případě chybějícího argumentu se nastaví hodnota „output.ply“.
- **f** - Značí soubor, do kterého budou zapsány informace o nalezených rovinách, konkrétně se do něj zapíší všechny nalezené roviny, počet bodů v nich a informace o bodech v nich,

konkrétně pořadí bodu v původním mračnu a jeho souřadnice. Pokud tento parametr není zadán, implicitně se nastaví soubor „planes_info.txt“.

4.2.2 Parametry ovlivňující metodu RANSAC

Dalším tři parametry ovlivňují chování metody RANSAC. Jedná se o parametry -d, -t a -m.

- **t** - Značí počet iterací metody RANSAC. Očekává jako argument číslo typu integer. Čím více iterací, tím déle běh aplikace trvá, ale na druhou stranu dosáhneme přesnějších výsledků. Implicitní hodnota počtu iterací je 100.
- **m** - Zadáním tohoto parametru se dá ovlivnit minimální počet bodů, který RANSAC musí nalézt, aby je potvrdil jako rovinu. Argument je číslo typu integer. V případě, že tolik bodů v rovině neleží, ukončí se cyklus hledání a žádná další rovina již není nalezena. Implicitně má tato proměnná hodnotu 10.
- **d** – Tento parametr není celočíselný, ale typu double. Udává maximální vzdálenost inlierů od roviny. Body, které leží dále od konkrétní roviny do ní nebudou započítány, body, které leží blíže ano. Implicitní hodnota je 1.0, tato hodnota je však velmi závislá na vstupním mračnu a při její špatné hodnotě může aplikace vykazovat prazvláštní výsledky.

4.2.3 Ostatní parametry příkazové řádky

- **p** - Poslední parametr ovlivňující běh samotné aplikace. Vyžaduje jako argument číslo typu double. Hodnota tohoto parametru se použije při zahazování velkých trojúhelníků, značí totiž maximální délku nejdelší hrany trojúhelníku. Trojúhelníky s delší hranou, než je hodnota parametru -p tedy budou zahozeny.
- **h** – Po tomto parametru se vypíše nápověda a ukončí běh aplikace. Nápověda obsahuje základní popis programu a popis jednotlivých parametrů v anglickém jazyce.

4.3 Vyhledávání rovin

V této podkapitole se budeme zabývat implementací metody RANSAC. Pokusil jsem se co nejvíce využívat služeb knihovny PCL a nevytvářet kód, který již byl vytvořen.

4.3.1 Nastavení parametrů metody RANSAC

V knihovně PCL existuje objekt typu SACSegmentation, který jsem využil ve svém projektu. Tomuto objektu je potřeba pouze nastavit několik argumentů a on provede vyhledání roviny sám. V případě mého projektu je třeba nastavit metodu segmentace na RANSAC, nastavit počet iterací, který známe

z příkazové řádky, stejně jako maximální vzdálenost bodů od roviny. Dalšími důležitými parametry jsou vstupní data (v tomto případě vstupní mračno bodů) a výstupní data, což je seznam inlierů a koeficienty rovnice nalezené roviny. Tyto operace jsou zabaleny ve funkci *set segmentation parameters*.

4.3.2 Extrakce roviny

Samotná extrakce roviny probíhá ve funkci *extract plane and remains*. V této funkci se mračno bodů rozdělí na dvě, v tom prvním zůstanou pouze body, které jsou součástí nalezené roviny, v tom druhém pouze ty, které součástí roviny nejsou. Toho docílíme pomocí objektu knihovny PCL ExtractIndices, kterému dáme jako vstup vstupní mračno a jako indície k filtraci seznam inlierů nalezených v předchozí funkci. Výstupem této funkce pro další zpracování jsou dvě mračna, jedno, na které uplatníme filtr s hodnotou false a proto v něm zůstanou body, které splňují indicie (tj. body v rovině), druhé, na které uplatníme filtr s hodnotou true a proto se do něj přidají body, které indicie nesplňují (tj. všechny ostatní body).

4.3.3 Transformace bodů z prostoru do roviny

V této části využijeme zejména služeb knihovny Eigen [15] (když nainstalujete knihovnu PCL, nainstaluje se vám i knihovna Eigen, bez které PCL není schopna pracovat, tudíž knihovnu Eigen již nemusíte instalovat). Tato část je v kódu přítomna ve funkci *transform plane to z0*. Z podkapitoly 4.3.1 již máme uloženy koeficienty nalezené rovnice. Vytvoříme z nich vektor, stejně jako vytvoříme vektor výsledné roviny (tj. 0.0, 0.0, 1.0). Z těchto dvou vektorů vytvoříme kvaternion a poté transformujeme celé mračno bodů v rovině tak, aby souřadnice z všech bodů bylo nulová (u většiny bodů nebude nulová, ale bude blízko nuly, protože body nebudou ležet přímo v rovině, ale budou mít drobnou odchylku).

4.4 Triangulace

Druhou důležitou částí mé práce je vytvoření sítě trojúhelníků z bodů v nalezených rovinách. Tato část má dva kroky, tím prvním je Delaunayho triangulace, tím druhým zahazování velkých trojúhelníků. Triangulaci lze najít ve funkci *do delaunay*.

4.4.1 Delaunayho triangulace

Pro implementaci Delaunayho triangulace jsem využil knihovnu libdel. Práce s ní je velmi jednoduchá. Vstupními daty je vektor bodů se souřadnicemi x a y. Tyto body získáme

z přetransformovaného mračna bodů. Dále již jen zavoláme konstruktor s tímto vektorem a poté metodu *triangulate*.

4.4.2 Zahazování velkých trojúhelníků

Výstupem delaunayho triangulace jsou trojúhelníky kolem jednotlivých vrcholů. U těchto trojúhelníků musíme zjistit, zda již nebyly v minulosti započítány (každý trojúhelník je uložen třikrát, jednou pro každý vrchol). Poté porovnáme délky všech stran trojúhelníku s hodnotou parametru -p příkazové řádky a trojúhelníky, které mají některou stranu delší, než je tato hodnota, zahazujeme.

4.5 Spojení do výsledné aplikace

Metoda RANSAC najde vždy jen jednu rovinu (tu s nejvíce body), proto musí celé hledání probíhat v cyklu. Tento cyklus se zastaví v případě, kdy nebudeme schopni nalézt rovinu o zadaném počtu bodů. Jak již víme z kapitoly 4.3.2, v dalším cyklu už nevybíráme rovinu z celého mračna bodů, protože bychom znovu vybrali tu samou, ale jen ze zbylých bodů.

Spojení nalezených trojúhelníků probíhá tím způsobem, že si po celou dobu běhu aplikace pamatuje souřadnice bodů v původním mračnu a výsledný trojúhelníky jsou uloženy jako vektor odkazů na body do tohoto mračna. Problém nastává právě ve chvíli, kdy z mračna vyjmu body v nalezené rovině a další rovinu hledám již jen ve zbylých bodech, protože musím nějakým způsobem zjistit polohu bodů v nalezené rovině v původním mračnu.

Tento problém řeším tak, že si zapamatuji pozice bodů první nalezené roviny v původním mračnu bodů. Když pak naleznu druhou rovinu, posunu odkazy bodů do mračna bez první roviny o hodnotu počtu bodů z první roviny, které mají nižší číslo, a dostanu tak odkazy do původního mračna bodů. Stejným způsobem pak pokračuji u dalších rovin.

Celý algoritmus tedy vypadá takto:

- Zpracuj parametry příkazové řádky (funkce *get_parameters*)
- Načti vstupní mračno (funkce *read_pcd_file*)
- Nastav parametry segmentace rovin (funkce *set_segmentation_parameters*)
- Dokud existuje další rovina
    - vyextrahuj rovinu (funkce *extract_plane_and_remains*)
    - transformuj rovinu tak, aby souřadnice z byla nulová (funkce *transform_plane_to_z0*)
    - proveď Delaunayho trinagulaci (funkce *do_delaunay*)
    - nastav parametry pro novou segmentaci
- Zapiš vytvořenou síť do výstupního souboru (funkce *write_mesh_to_ply*)

5 Testování a výsledky

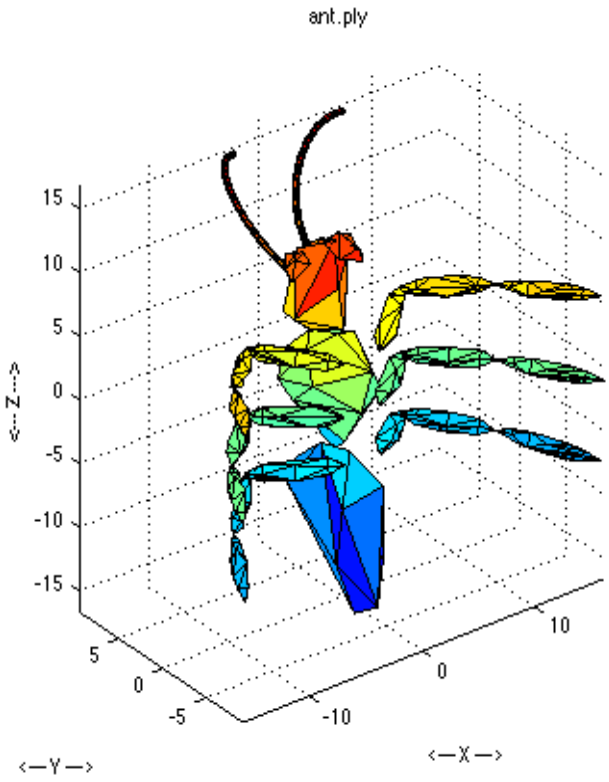
Tato kapitola se zabývá testováním aplikace a výsledky, které testování přineslo. Obsahuje kompletní popis testů, které byly provedeny a závěry z nich vyplývající.

5.1 Popis testů

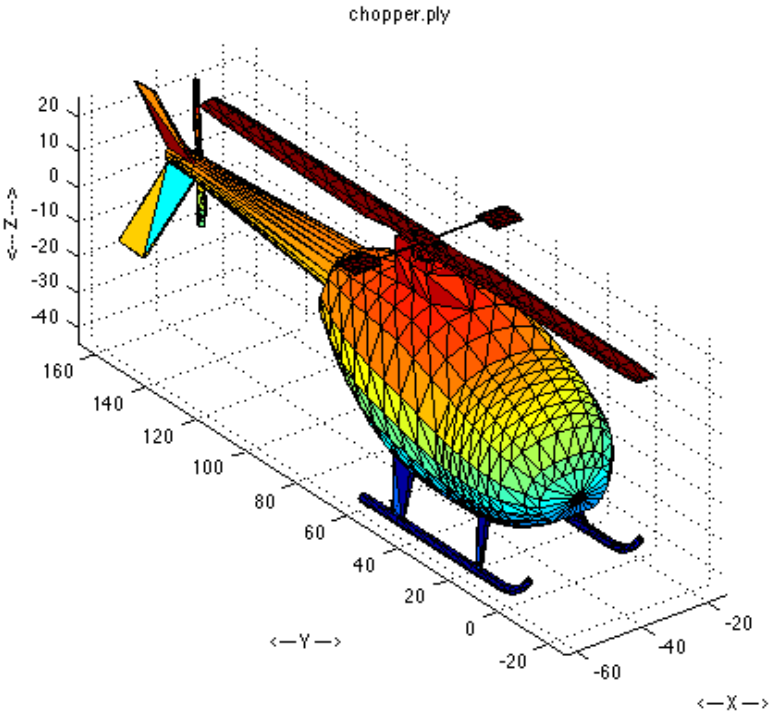
Testování probíhalo na několika vzorcích mračen z nichž některé v sobě obsahovaly rovinné útvary a jiné je neobsahovaly. Aplikace byla testována na mračnech od čtyř set do půl miliónu bodů. Výsledky byly porovnány s výsledky metody *greedy search* z knihovny PCL.

5.1.1 Testovací data

Testy proběhly na celkem pěti různých mračnech. Nejmenšími mračny jsou upravené polygonální sítě ant.pcd a chopper.pcd. Z těchto sítí jsem odstranil informaci o propojení bodů a zůstaly pouze samotné mračno bodů. Tato mračna obsahují 486 a 1066 bodů a nejsou rovinného charakteru. Jejich grafickou podobu je možné vidět na obrázcích 5.1 a 5.2.

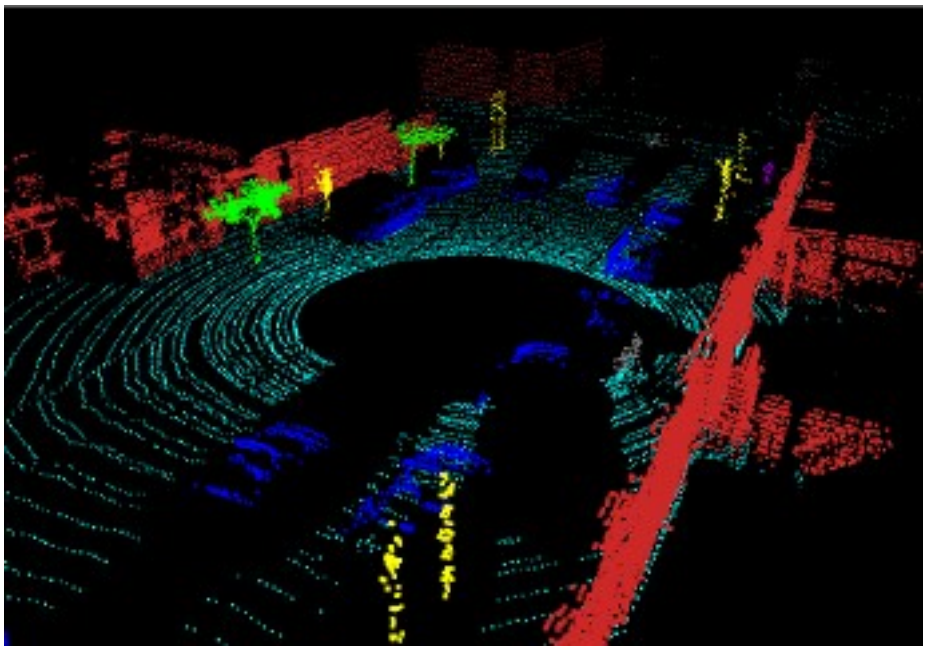


Obrázek 5.1: Testovací soubor ant.ply

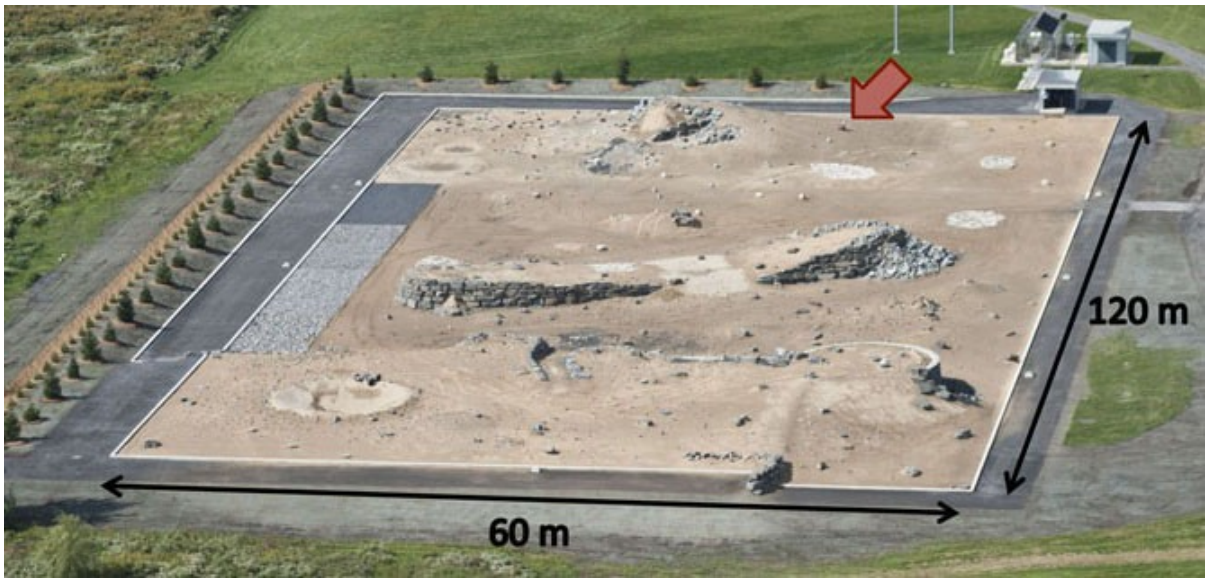


Obrázek 5.2: Testovací soubor chopper.ply

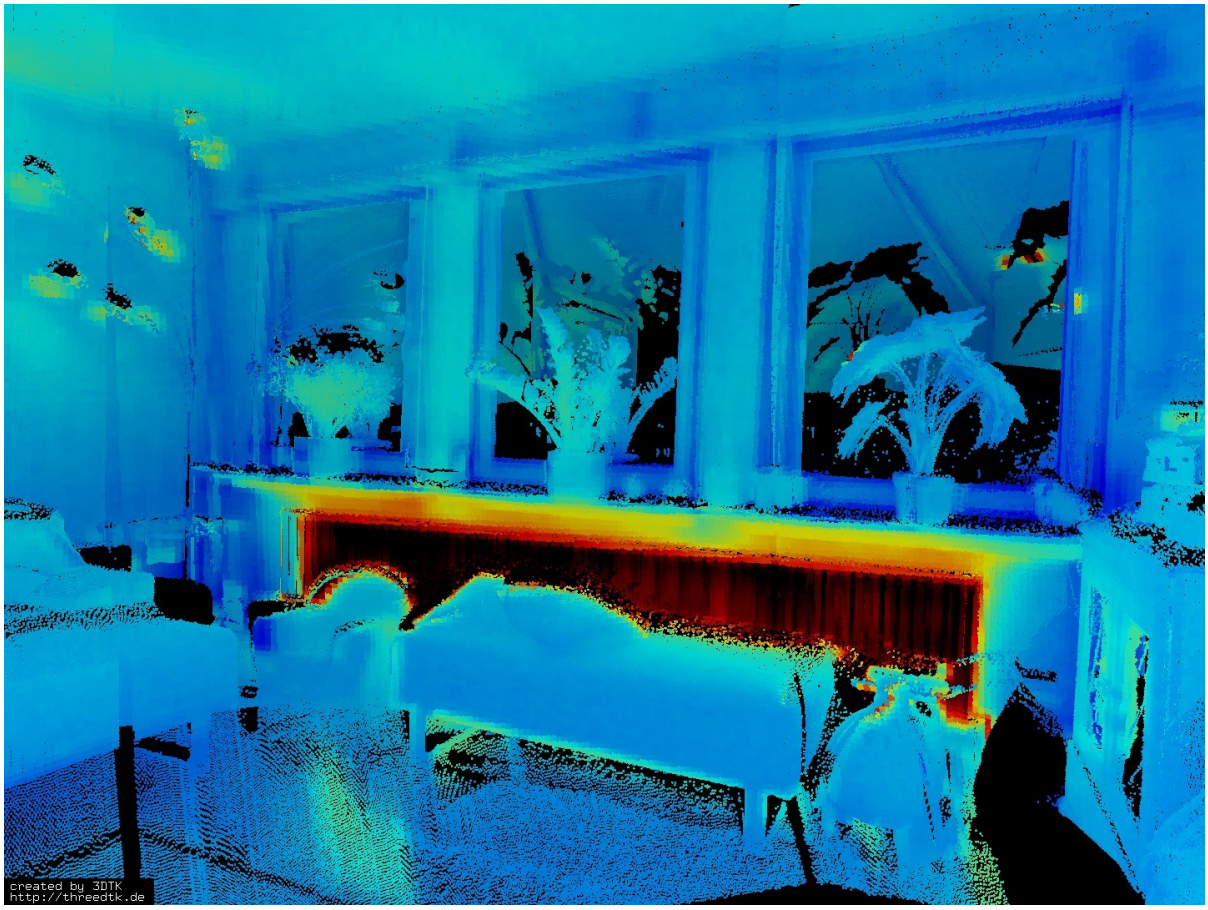
Dalšími testovacími daty byla data obsahující roviny. Jednalo se o mračna obsahující větší počet bodů. Nejmenší z nich urban_scene.pcd obsahuje 65 553 bodů, další z nich, ground.pcd obsahuje 190 090 bodů a poslední house.pcd obsahuje 592 260 bodů. Grafickou podobu jednotlivých dat lze vidět na obrázcích 5.3, 5.4 a 5.5.



Obrázek 5.3: Testovací soubor urban_scene.pcd



Obrázek 5.4: Testovací soubor ground.pcd



Obrázek 5.5: Testovací soubor house.pcd

Veškeré informace o testovacích souborech shrnuje tabulka 5.1.

Název	Počet bodů	Zdroj
Ant	486	http://people.sc.fsu.edu/~jburkardt/data/ply/ply.html
Chopper	1 066	http://people.sc.fsu.edu/~jburkardt/data/ply/ply.html
Urban scene	65 553	http://homes.cs.washington.edu/~kevinlai/datasets.html
Ground	190 090	http://asrl.utias.utoronto.ca/datasets/3dmap/p2at_met.html
House	592 260	http://kos.informatik.uni-osnabrueck.de/3Dscans/

Tabulka 5.1: Souhrn informací o testovaných datech

5.1.2 Metoda greedy search

Výsledky mé aplikace budu porovnávat s výsledky metody greedy search. Jedná se o upravenou implementaci tutorialu pcl o rekonstrukci *Fast triangulation of unordered point* clouds [16].

Tato metoda funguje nejlépe pro vyhlazená mračna bodů bez velkého šumu. Metoda udržuje seznam bodů, do kterého postupně přidává další body a spojuje je s všemi možnými body, které splňují kritéria pro spojování. Metoda se dá použít jak pro jednu mračno bodů, tak pro data z několika scanů, která jsou spojena dohromady. Triangulace je prováděna lokálně přidávám bodů v okolí a připojováním dosud nepřipojených bodů. Metoda má několik parametrů, které ovlivňují výsledky triangulace. Tyto parametry jsou vypsány níže.

- **MaximumNearestNeighbors** – Ovlivňuje okolí bodu. Specifikuje maximální počet sousedů pro každý bod a jeho typická hodnota se pohybuje mezi 50 až 100 sousedy. Čím větší je hodnota parametru, tím více bodů bude zahrnuto do sousedství a tím více trojúhelníků bude ve výsledné síti.
- **Mu** – Značí maximální relativní vzdálenost bodu od nejbližšího bodu, aby byl stále zahrnut do sousedství (např. pokud je parametr *Mu* nastaven na hodnotu 4, nejbližší bod je vzdálen 2, do sousedství budou zahrnuty všechny body do vzdálenosti 8). Typická hodnota tohoto parametr se pohybuje mezi 2,5 a 3. Čím větší je hodnota parametru, tím více bodů bude zahrnuto do sousedství a tím více trojúhelníků bude ve výsledné síti.
- **SearchRadius** - Značí nejdelší možnou hranu trojúhelníku ve výsledné síti. Hodnota této proměnné je absolutní a proto velmi záleží na konkrétním mračnu. Čím větší hodnota, tím více trojúhelníku ve výsledné síti bude, v případě nižší hodnoty budou v síti pouze menší trojúhelníky.
- **MinimumAngle** – Ovlivňuje minimální velikost úhlu v trojúhelníku. Jeho dodržení není garantováno. Typická hodnota je 10 stupňů (Pi/18), uvádí se v radiánech.

- **MaximumAngle** – Ovlivňuje maximální velikost úhlu v trojúhelníku. Na rozdíl od předchozího parametru je jeho dodržení garantováno vždy. Typická hodnota je 120 stupňů (2*Pi/3) a opět se uvádí v radiánech.
- **MaximumSurfaceAngle** – Je zde zaveden pro případ, že jsou v mračnu ostré hrany či rohy a kdy jsou dvě různé strany povrchu velmi blízko sebe. Z tohoto důvodu nejsou body připojeny k aktuálnímu bodu, pokud se jejich normály odklání více, než specifický úhel zadán v tomto parametru. Na většině mračen funguje hodnota 45 stupňů (Pi/4).
- **NormalConsistency** – Úhel v předchozím parametru je vypočítán jako úhel mezi čárami definovanými v normálách, pokud má *NormalConsistency* hodnotu false.

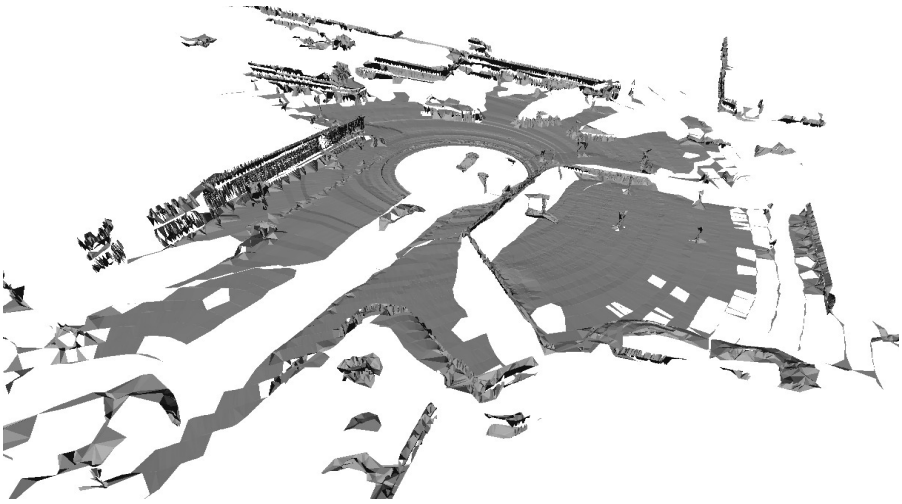
5.2 Průběh testů

U samotného testování byly sledovány dva parametry. Tím prvním byla rychlost rekonstrukce, tím druhým pak kvalita, která byla měřena subjektivním vnímáním. Statistiky byly sledovány jak pro rovinné útvary, tak pro útvary nerovinné.

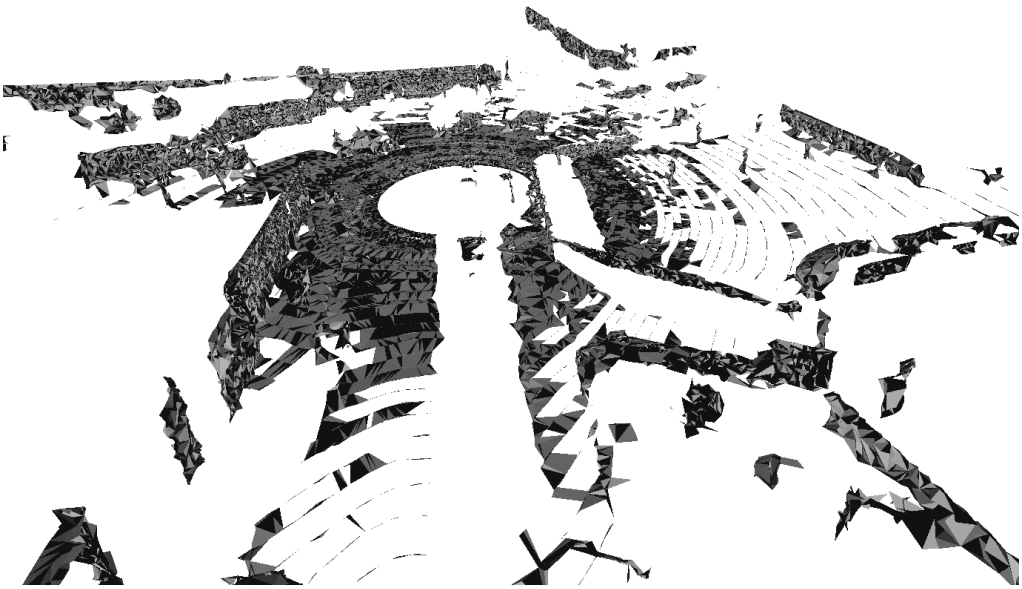
5.2.1 Rovinné útvary

Do této kategorie řadíme testovací soubory ground a urban_scene. Pro testovací soubor urban_scene byla experimentálně nalezeny optimální parametry d = 1.0 (maximální vzdálenost bodu od roviny, aby do ní byl stále ještě započítán) a p = 1.2 (maximální délka hrany trojúhelníka). S těmito hodnotami byla z těchto testovacích dat o velikosti 65 553 bodů vytvořena trojúhelníková síť za 7.95 sekundy.

Konkurenční greedy search byl spuštěn s parametrem SearchRadius = 1.2 (stejně jako u předchozí metody) a MaximumNearestNeighbors = 100. Parametr Mu byl nastaven na hodnotu 7.5, což je sice téměř třikrát více, než jsou doporučené hodnoty, ale při nižší hodnotě tohoto parametru nalezla metoda velmi málo trojúhelníků. Možná také proto byl čas této metody 74.56 sekundy, což je málem desetinásobek času stráveného mojí metodou. Náhled na celkové sítě si můžete prohlédnout na obrázcích 5.6 a 5.7.



Obrázek 5.6: Trojúhelníková síť mračna urban_scene mojí metodou



Obrázek 5.7: Trojúhelníková síť urban_scene metodou greedy_search

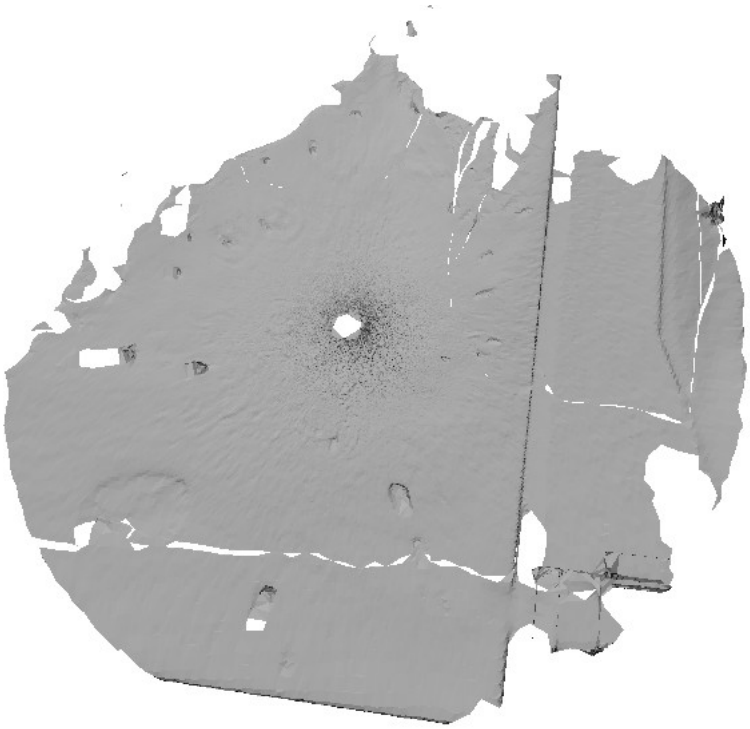
Na první pohled vypadají sítě velmi podobně, přičemž moje metoda vytvořila i některé trojúhelníky, které metoda greedy search ne. Pravděpodobně by tyto trojúhelníky přidalo navýšení parametru Mu za cenu další časové zátěže. Jak je též vidět z porovnání obrázků, síť vytvořená mým algoritmem má všechny trojúhelníky stejně orientované, zatímco síť vytvořená metodou greedy search ne. Tento problém je řešitelný i u metody greedy search, vyžadoval by ale rozšíření algoritmu. S rovnými plochami na těchto datech si tedy poradí spíše lépe můj algoritmus, ovšem na této městské scéně nejsou pouze rovné plochy, ale například i stromy. Na obrázcích 5.8, 5.9 a 5.10 je vidět detail

jednoho stromu z mračna, nejprve v pouze bodové podobě, poté po triangulaci mým algoritmem a nakonec po triangulaci metodou greedy search.

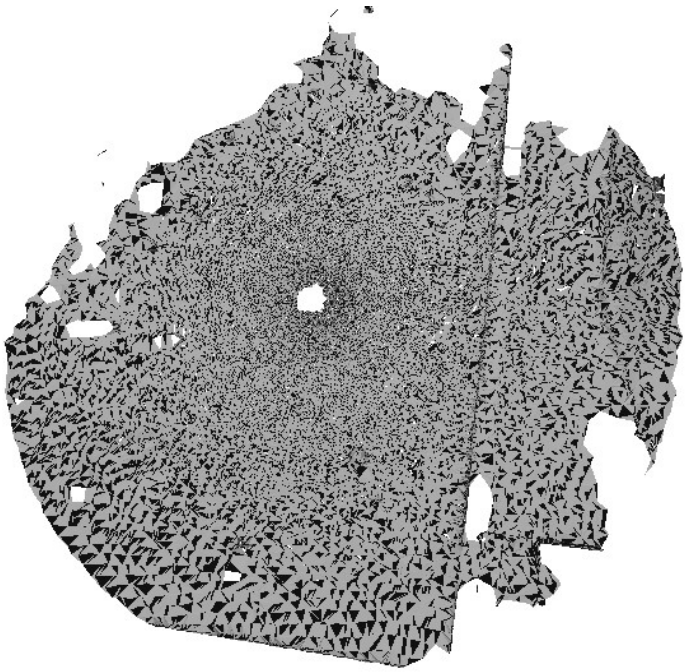
Zleva: Obrázek 5.8: Strom pouze v bodech, Obrázek 5.9: Síť stromu vytvořená mojí metodou, Obrázek 5.10: Síť stromu vytvořená metodou greedy search

Jak je vidět z těchto obrázků, moje metoda zařadí korunu stromu buď do své vlastní roviny nebo, což je pravděpodobnější, do roviny s ostatními korunami stromů a poté zahodí propojení mezi nimi. Spodek kmene naopak zahrne do roviny země a z tohoto důvodu nejsou kmen a koruna stromu propojeny. Greedy search si s tímto problémem poradí lépe a strom tam má doopravdy alespoň nějaký kmen, který ho spojuje se zemí.

Testovací data ground obsahující 190 090 bodů byla testována mojí metodou s parametrem d = 0.4 a p = 1.0. U metody greedy search jsou parametry stejné jako u předchozího vzorku dat mimo parametru SearchRadius, který byl změněn na 1.0 stejně jako u mé metody. Čas mojí metody na těchto datech byl 20.11 sekundy, čas metody greedy search 4 minuty a 27.92 sekundy. Tento rozdíl je do značné míry opět dán vysokou hodnotou parametru Mu u metody greedy search. Pro srovnání při hodnotě tohoto parametru 2.5 místo 7.5 byl čas metody greedy search 2 minuty a 1.13 sekundy, což je méně než polovina, výsledný model však nedosahoval potřebných výsledků. Výsledné modely jsou zobrazeny na obrázcích 5.11 a 5.12.



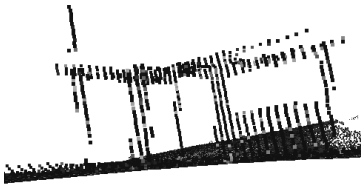
Obrázek 5.11: Trojúhelníková síť mračna ground mojí metodou

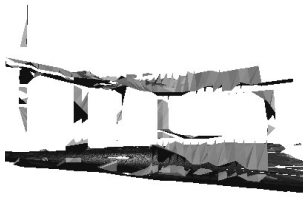


Obrázek 5.12: Trojúhelníková síť mračna ground metodou greedy search

Na tomto mračnu je dobře vidět především zpracování děr, se kterým si oba algoritmy dokázaly poradit. Jedná se jak o okraj, který je na některých místech členitý, tak o díry uvnitř mračna.

Na kraji tohoto mračna stojí nějaká budova, které jakoby chybí stěny, má pouze rohové sloupy. Detail na tuto budovu je zobrazen na obrázcích 5.13, 5.14 a 5.15.





Zleva: Obrázek 5.13: Budova pouze v bodech, Obrázek 5.14: Síť budovy vytvořená mojí metodou, Obrázek 5.15: Síť budovy vytvořená metodou greedy search

Jak je vidět z těchto obrázků, s tímto úkolem si lépe poradila metoda greedy search, která výchozí mračno celkem dobře vystihla. Problémem mé metody pro tento konkrétní úsek je nespojování rovin, což se zde projevuje na tom, že výsledná síť je jakoby neúplná. V případě odstranění tohoto nedostatku by měla už i moje metoda vyhovovat a triangulace by měla být spojitá a vystihovat rysy budovy.

5.2.2 Nerovinné útvary

Dalšími dvěma testovacími vzorky byly dva malé modely, jeden znázorňující mravence, druhý helikoptéru. Účelem těchto dat bylo zjištění chování mého algoritmu na datech, která neobsahují sama od sebe velký počet rovin. Na mračnu ant dosáhl můj program času 0.05 sekundy, zatímco greedy search času 0.46 sekundy. Parametry pro můj algoritmus byly d = 4.0 a p = 2.0 a pro greedy search byl nastavena SearchRadius na 4.0 a Mu na 3.0. Na obrázcích 5.16 a 5.17 jsou zobrazeny výsledky.

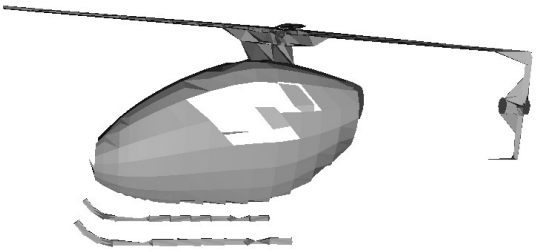
Zleva: Obrázek 5.16: Trojúhelníková síť mračna ant mojí metodou, Obrázek 5.17: Trojúhelníková síť mračna ant metodou greedy search

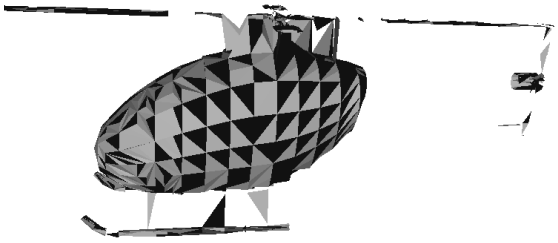
Jak je vidět z těchto obrázků, v obou případech některé trojúhelníky chybí. Jedná se především o oblast nohou a tykadel. Tento problém lze vyřešit změnou parametrů, ovšem v tom případě ve výsledné síti budou i parazitní trojúhelníky, které v ní nechceme. Tento případ je demonstrován na obrázcích 5.18 a 5.19.

Zleva: Obrázek 5.18: Trojúhelníková síť mračna ant mojí metodou, Obrázek 5.19: Trojúhelníková síť mračna ant metodou greedy search

Na obou obrázcích jsou vidět spojené zadní dvě nohy mravence, výrazné je toto spojení především u metody greedy search. Moje metoda navíc spojila tykadla do jedné roviny. Tato síť by pravděpodobně u obou metod musela být po jejich průběhu ručně poupravena.

Druhým nerovinným útvarem, který byl testován, byl soubor chopper. Na tomto mračnu dosáhl můj program času 0.13 sekundy, zatímco greedy search času 0.75 sekundy. Spouštěn byl s parametry d = 10.0 a p = 8.7 pro můj program a Mu 15.0 u metody greedy search. Výsledky jsou zobrazeny na obrázcích 5.20 a 5.21.





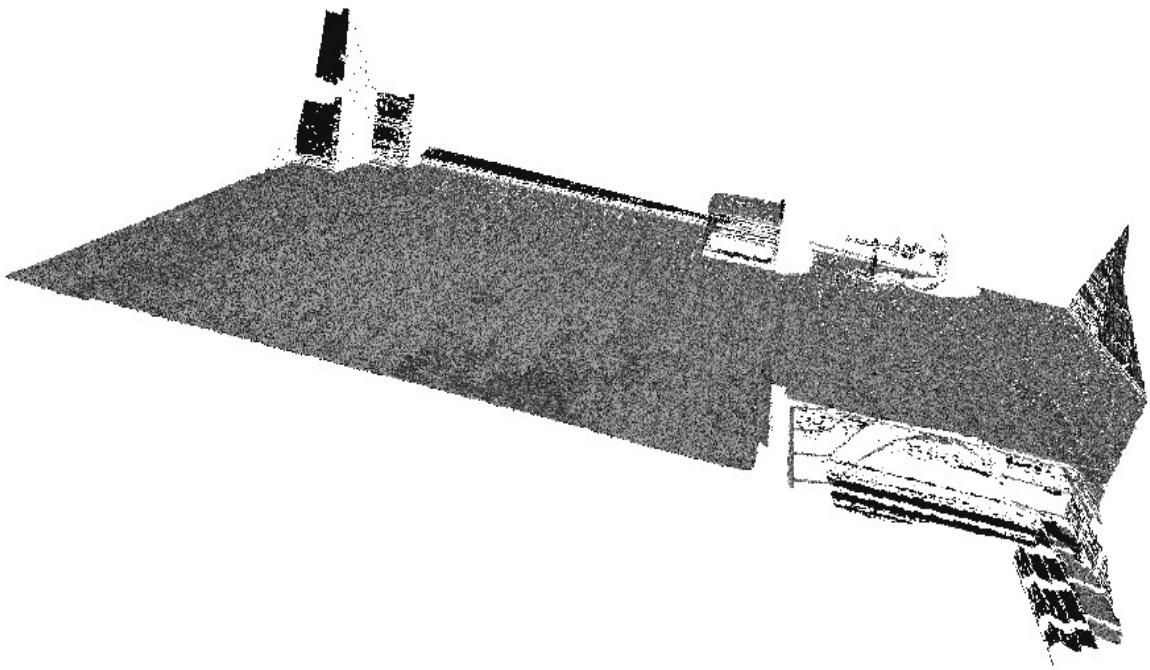
Zleva: Obrázek 5.20: Trojúhelníková síť mračna chopper mojí metodou, Obrázek 5.21: Trojúhelníková síť mračna chopper metodou greedy search

Z obrázků je vidět, že moje metoda nechala uprostřed kabiny vyříznutou díru a nepřipojila ližiny, vrchní vrtuli připojil k zadní vrtuli a naprosto chybí ocas vrtulníku. Algoritmus greedy search taktéž připojil zadní vrtuli k vrchní vrtuli a zanedbal ocas, na rozdíl od mého algoritmu nemá uvnitř

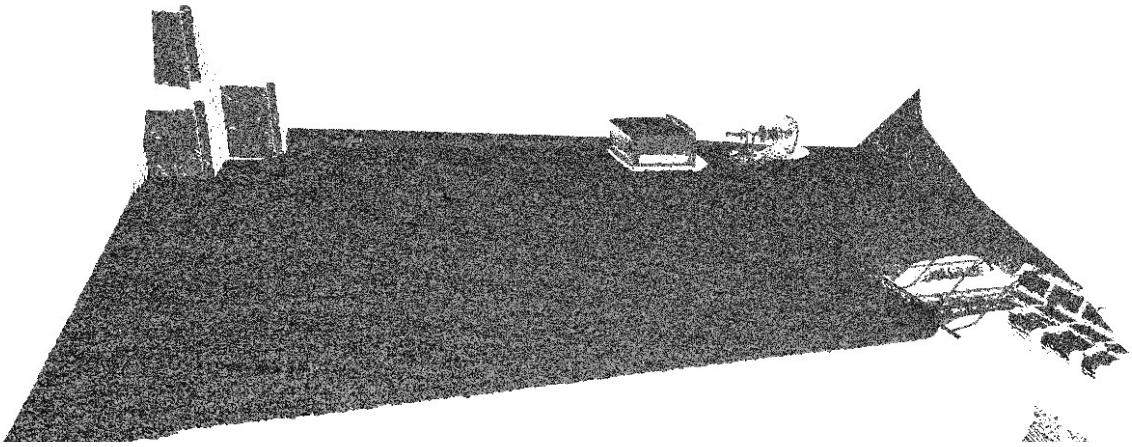
vrtulníku žádnou díru. Na druhou stranu ližiny připojil k trupu vrtulníku na nepatřičných místech a vrchní vrtule obsahuje díry. Tento model by opět potřeboval v obou případech ruční dodělávky.

5.2.3 Velká mračna

Posledním testovacím mračnem bylo mračno house. Toto mračno se od ostatních odlišuje především svou velikostí, má více jak půl milionu bodů. Mračno obsahuje v podstatě jen samé roviny. Po spuštění mým algoritmem s parametry d = 0.4 a p = 1.0 počítal program 2 minuty a 35.75 sekundy. Algoritmus greedy search byl spuštěn s parametrem SearchRadius = 1.0 a Mu = 7.5 a skončil v čase 9 minut a 13.17 sekundy. Výsledky jsou zobrazeny na obrázcích 5.22 a 5.23.



Obrázek 5.22: Trojúhelníková síť mračna house mojí metodou



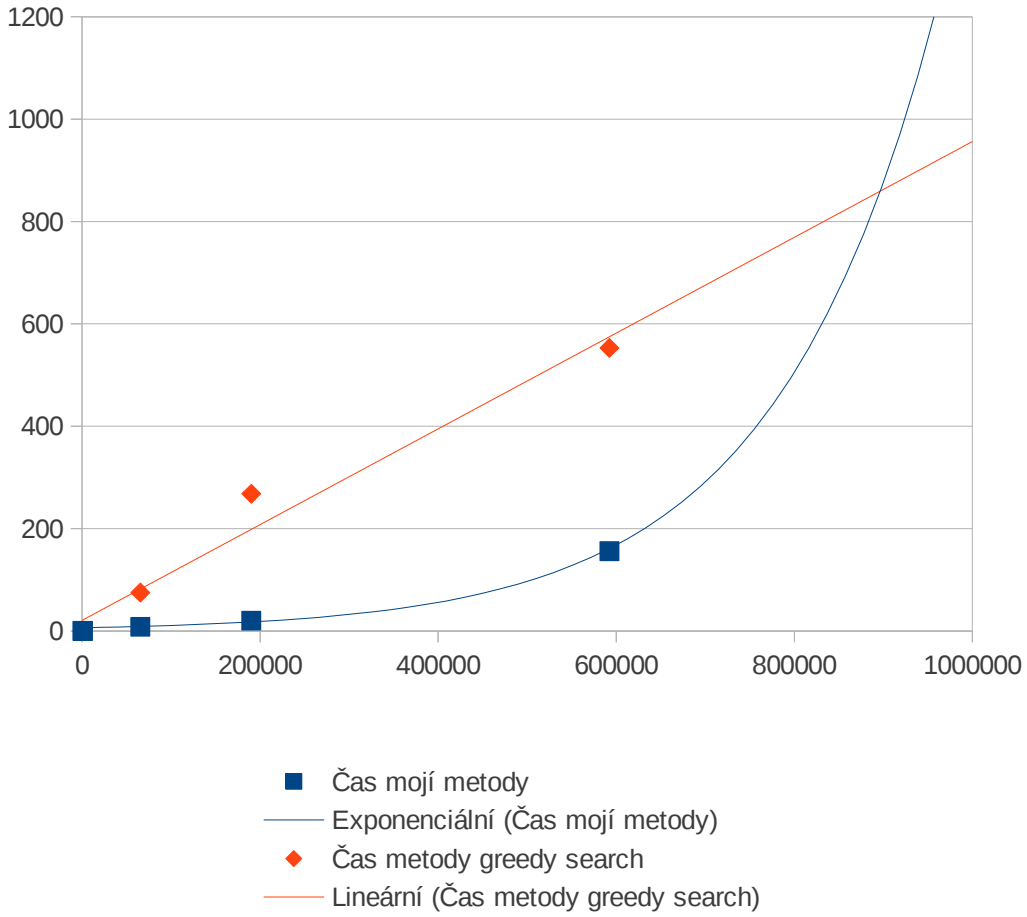
Obrázek 5.23: Trojúhelníková síť mračna house metodou greedy search

Na tomto mračnu je zajímavý především čas, který na něm oba algoritmy strávily. Zatímco u předchozích mračen měl můj algoritmus až desetinásobně rychlejší časy, zde má čas už jen čtvrtinový. Časy obou metod a velikosti všech mračen shrnuje tabulka 5.2.

Název	**Počet bodů**	**Čas strávený mojí metodou**	**Čas strávený metodou greedy search**
Ant	486	0.05 s	0.46 s
Chopper	1 066	0.13 s	0.75 s
Urban scene	65 553	7.95 s	1 m, 14.56 s
Ground	190 090	20.11 s	4m, 27.92 s
House	592 260	2 m, 35.75 s	9m, 13.17 s

Tabulka 5.2: Informace o času stráveném jednotlivými metoda na všech mračnech

Zatímco metoda greedy search má čase přibližně lineární ku počtu bodů, moje metoda má pro menší mračna časy výrazně kratší a pro větší mračna delší. Přehledně je to znázorněno na grafu 5.1.



Graf 5.1: Zobrazení času stráveného oběma metodami na všech testovacích mračnech

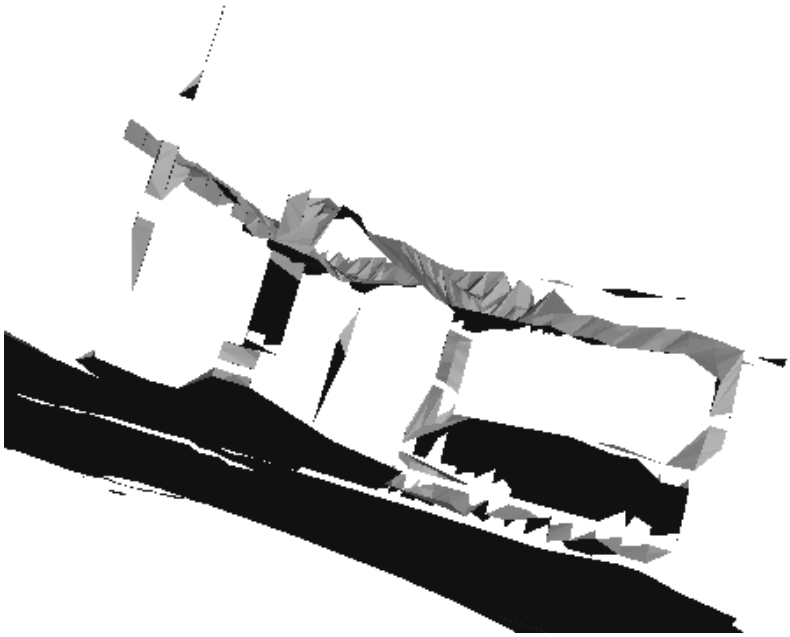
Z grafu vyplývá, že odhadovaný zlom, kdy začíná být greedy search rychlejší než moje metoda, nastává někde u 900 000 bodů. To znevýhodňuje můj algoritmus, protože většina skutečných mračen bude mít bodů více. Z grafu také vyplývá, že greedy search má pravděpodobně lineární složitost, zatímco můj algoritmus exponenciální, tudíž pro mračna o mnoha milionech bodů nebude můj program použitelný. Exponenciální složitost mého algoritmu je způsobena dvojitým cyklem ve funkci *save_plane_info*, který zjišťuje pozice nalezených bodů v rovině v původním mračnu obsahujícím všechny body.

5.3 Vliv hodnot jednotlivých parametrů

Jak již bylo řečeno dříve, uživatel může zadat přes příkazovou řádku čtyři různé parametry, které ovlivňují celkový výstup. Jak mohou jejich hodnoty ovlivňovat výsledek budeme demonstrovat na mračnu ground. Vynechám zde atribut minimálního počtu bodů pro rovinu, protože ten má na výslednou síť pouze marginální vliv. Při snižování jeho hodnoty začnou v síti přibývat malé roviny, při zvyšování naopak ubývat. Nemá smysl tomuto parametru dávat nižší hodnotu než 3.

5.3.1 Počet iterací metody RANSAC

Se zvyšujícím se počtem iterací metody RANSAC se zvyšuje pravděpodobnost, že nalezené roviny jsou doopravdy ty největší. Hodnota 100, která je dána implicitně, pro všechna testovaná mračna vracela správné výsledky. Při hodnotě iterací 10 u mračna ground vznikly stejné výsledky jako pro 100 iterací. Může se jednat o náhodu, ovšem u tohoto mračna to náhoda být nemusí, protože obsahuje pouze několik velkých rovin. Při hodnotě 3 se na první pohled zdálo, že jsme dostali opět stejný výsledek, ale při bližším pozorování šlo nalézt malé chyby, například jinak vytvořená stěna domu, jak je vidět na obrázku 5.24.



Obrázek 5.24: Dům z mračna ground při třech iteracích metody RANSAC

Tato změna vznikla pravděpodobně tím, že v některém cyklu nebyla nalezena některá větší rovina, a proto byla za největší považována jiná, která ve skutečnosti největší nebyla. Zvyšující se číslo parametru už k žádným změnám nevede. Snížení počtu iterací nijak výrazně nemění čas běhu

programu, tudíž hledáním metodou RANSAC stráví aplikace jen marginální čas oproti běhu celé aplikace.

5.3.2 Vzdálenost bodu od roviny

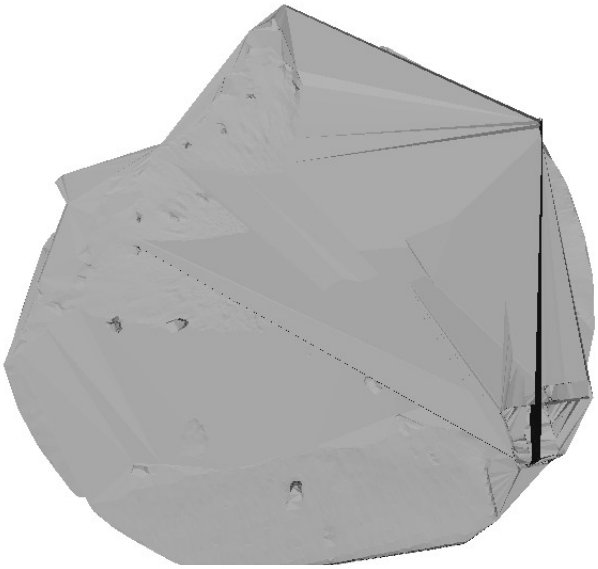
Dalším parametrem je vzdálenost bodu od roviny v metodě RANSAC. Tento parametr ovlivňuje především počet rovin. V případě, že zadáme hodnotu tohoto parametru 0.1 místo 0.4 některé roviny v mračnu se nám rozdělí na několik nových. Tento případ je zobrazen na obrázku 5.25, na kterém lze vidět jeden z okrajů mračna, kde se směrem do středu terén lehce svažuje.

Obrázek 5.25: Okraj mračna ground při parametru d = 0.1

Je též důležité si uvědomit, že změna tohoto parametru výrazně ovlivňuje strávený čas aplikace. Při nižší hodnotě atributu nalezne program více rovin a celkově stráví více času a to i přesto, že roviny jsou menší. Na tomto konkrétním mračnu se jedná o zvýšení časové zátěže asi o 10%.

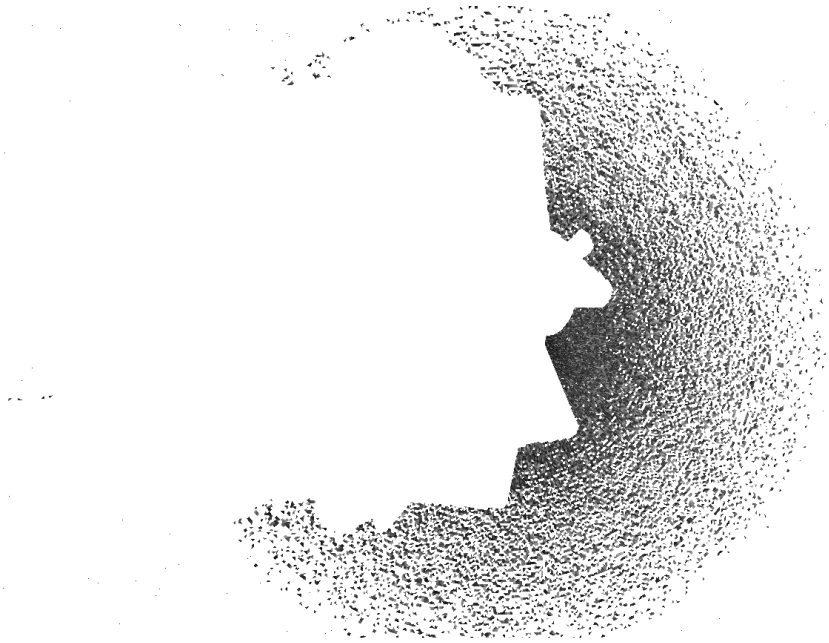
5.3.3 Maximální délka hrany

Tento parametr výrazně ovlivňuje výslednou strukturu sítě. Pokud je zadána příliš velká maximální délka hrany, z rovin se vytvoří jejich konvexní obálka, přijdeme o veškeré výřezy na krajích roviny a o veškeré díry uvnitř. Síť, která byla vytvořena s příliš velkým parametrem maximální délky hrany je znázorněna na obrázku 5.26.



Obrázek 5.26: Mračno ground s příliš vysokým parametrem p

Z výsledné sítě zmizely všechny díry a její okraje nejsou zdaleka tak členité jako předtím. Pokud naopak zadáme hodnotu tohoto parametru příliš malou, může nám ze sítě vymizet spousta trojúhelníku, které by do ní měly patřit, jak je zobrazeno na obrázku 5.27. Z původní sítě zůstal jen samotný střed a i v něm je spousta děr.



Obrázek 5.27: Mračno ground s příliš malým parametrem p

5.4 Shrnutí testů

Z provedených testů vyplývá, že kvalita dosažených výsledků je srovnatelná. Z tohoto důvodu se hlavním kritériem hodnocení stává čas. V tomto ohledu je moje metoda vhodnější pro mračna obshaující méně než 900 000 bodů, pro větší mračna je již vhodnější metoda greedy search.

6 Závěr

Cílem mé bakalářské práce bylo vytvořit aplikaci pro rekonstrukci povrchu z mračna bodů. Měl jsem vycházet z existujících metod a buď některou z nich implementovat, nebo vymyslet nějakou vlastní.

Výsledkem mé práce je samostatná aplikace implementující vlastní metodu, která je však založena na již existujících algoritmech. Aplikace vytváří z mračna bodů síť trojúhelníků, která může být dále zpracovávána v různých jiných programech. Snažil jsem se porovnat možnosti mé metody s již existující knihovní metodou.

Moje aplikace má jistě spoustu možností dalšího vývoje. Tou první je navazování rovin. V aktuálním stavu vznikají mezi jednotlivými rovinami díry, které nejsou nijak triangulovány. Tento stav by bylo dobré do budoucna změnit. Dalším vylepšením mé práce může být změna kritéria dle kterého se zahazují velké trojúhelníky. Nyní se jedná o absolutní délku nejdelší hrany, bylo by vhodné spojit tuto délku například i s relativními délkami stran oproti délkám v okolních trojúhelnících či s velikostí úhlů v trojúhelníku. Mohl bych též prozkoumat důvody vysoké časové náročnosti aplikace pro velká mračna a pokusit se tento nepříznivý stav změnit nějakou vhodnější implementací algoritmu.

Mimo změn samotného jádra práce by aplikace mohla mít grafické uživatelské rozhraní, které ale není nutné, protože k rekonstrukci mračen bodů se většinou dostanou pouze zkušenější uživatelé. Také by bylo možné přidat zobrazování mračen i vytvořených sítí, případně tvorbu samotných mračen z fotografií či videozáznamů. Možností je spoustu, tou nejlepší by dle mého názoru bylo přidat tento program jako zásuvný modul do nějaké jiné aplikace, která tohle všechno již umí.

Díky této práci jsem se seznámil s problematikou mračen bodů a s metodami jejich rekonstrukce. Taktéž jsem prohloubil své programovací znalosti jazyka C++, naučil jsem se pracovat s nestandardními knihovnami a generovat makefile v programu cmake. Získal jsem spoustu nových zkušeností, vytvořil jsem sám nějaký větší projekt a celkově pro mě byla tato práce velkým přínosem.

Literatura

[1] Point cloud. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-21]. Dostupné z: http://en.wikipedia.org/wiki/Point_cloud

[2] MeshLab [online]. 2005 [cit. 2013-04-21]. Dostupné z: http://meshlab.sourceforge.net/

[3] 3D scanner. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-21]. Dostupné z: http://en.wikipedia.org/wiki/3D_scanner

[4] Bundler [online]. 2010 [cit. 2013-04-21]. Dostupné z: http://phototour.cs.washington.edu/bundler/

[5] The PCD (Point Cloud Data) file format. PCL - Point Cloud Library [online]. [cit. 2013-04-21]. Dostupné z: http://pointclouds.org/documentation/tutorials/pcd_file_format.php

[6] PLY (file format). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-21]. Dostupné z: http://en.wikipedia.org/wiki/PLY_(file_format)

[7] Amenta, N.; Choi, S.; Kolluri, R. K.: The Power Crust. 2001.

[8] KAZHDAN, M.; BOLITHO, M.; HOPPE, H.: Poisson surface reconstruction. In Symposium on Geometry Processing 2006, s. 61–70.

[9] OHTAKE, Y.; BELYAEV, A.; ALEXA, M.; aj.: Multi-level partition of unity implicits. In SIGGRAPH ’03 ACM SIGGRAPH 2003 Papers, 2003, ISBN 1-58113-709-5, s. 463–470.

[10] MOBIUS, J.; KOBBELT, L.: OpenFlipper: an open source geometry processing and rendering framework. In Proceedings of the 7th international conference on Curves and Surfaces, 2010, ISBN 978-3-642-27412- 1, s. 488–500.

[11] FISCHLER, M. A.; BOLLES, R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In Communications of the ACM Volume 24 Issue 6, 1981, s. 381–395.

[12] Delaunay triangulation. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-24]. Dostupné z: http://en.wikipedia.org/wiki/Delaunay_triangulation

[13] 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.

[14] PIYUSH, Kumar. Triangle++. Apache HTTP Server Test Page powered by CentOS [online]. 2006 [cit. 2013-04-21]. Dostupné z: http://www.compgeom.com/~piyush/scripts/triangle/index.html

[15] Guennebaud, G.; Jacob, B.; aj.: Eigen v3. http://eigen.tuxfamily.org, 2010.

[16] Marton, Z. C.; Rusu, R. B.; Beetz, M.: On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, May 12-17 2009.

Seznam příloh

Příloha 1. CD se zdrojovými texty a dokumentací