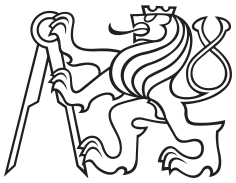
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra Kybernetiky

Plánování trajektorie bezpilotního prostředku pro inspekci 3D objektů

Martin Chloupek

Vedoucí: Ing. Milan Rollo, Ph.D.
Obor: Robotika
Studijní program: Kybernetika a robotika
Květen 2016

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Martin C h l o u p e k

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Plánování trajektorie bezpilotního prostředku pro inspekci 3D objektů

Pokyny pro vypracování:

1. Seznamte se se způsoby strojové reprezentace 3D objektů.
2. Zvolte vhodnou reprezentaci, která umožní matematické operace pro plánovací algoritmy.
3. Navrhněte algoritmus pro plánování letových trajektorií bezpilotních prostředků pro vizuální inspekci 3D objektů.
4. Navržený algoritmus implementujte a otestujte jeho vlastnosti v simulaci.
5. Ověřte algoritmus nasazením na reálný bezpilotní prostředek.

Seznam odborné literatury:

[1] Hallermann N., Morgenthal G.: Visual inspection strategies for large bridges using Unmanned Aerial Vehicles (UAV), IABMAS 2014, Shangai, 2014.
[2] Eschmann C., Kuo C.M., Kuo C.H., Boller, C.: Unmanned Aircraft Systems for Remote Building Inspection and Monitoring, 6th European Workshop on Structural Health Monitoring, Dresden, 2012.
[3] Máthé, K., Buşoniu L.: Vision and Control for UAVs: A Survey of General Methods and of Inexpensive Platforms for Infrastructure Inspection, In Sensors, 25;15(7), June 2015.

Vedoucí bakalářské práce: Ing. Milan Rollo, Ph.D.

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 15. 12. 2015

Poděkování

Děkuji svému vedoucímu Ing. Milanu Rollovy, Ph.D. za odborné vedení mé bakalářské práce, cenné připomínky a trpělivost.

Prohlášení

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

V Praze, 27. května 2016

Abstrakt

Cílem této práce je navrhnout a implementovat systém pro plánování trajektorií bezpilotního prostředku. Z mračna bodů, získaného zařízením LIDAR, vytvoříme polygonální model pomocí algoritmu BPA. Polygonální model použijeme pro vypočtení souřadnic bodů, ze kterých se bude skládat výsledná trajektorie. Poslední částí je vyřešit problém obchodního cestujícího pro vypočtené body. Implementovaným systémem je možné naplánovat inspekci jednoduchých tvarů (např. koule), ovšem u komplexních tvarů není zaručena inspekce celého objektu. Hlavním přínosem této práce je ucelený postup od mračna bodů až k vytvoření trajektorie pro bezpilotní prostředek.

Klíčová slova: UAV, bezpilotní prostředek, plánování trasy, LIDAR, mračno bodů, polygonální model, problém obchodního cestujícího, A* ve 3D, 2-opt

Vedoucí: Ing. Milan Rollo, Ph.D.

Abstract

The goal of this work is to design and implement a system for planning the trajectory of UAV. From the point cloud obtained by LIDAR device, we create a polygon mesh using the Ball-Pivoting algorithm. We use the polygon mesh for calculating the coordinates of points, which will be composed the result trajectory. The last part is to solve the traveling salesman problem for the calculated points. The implemented system can be use for planning inspection of simple shapes (eg.Balls). Entire inspection of complex shapes is not guaranteed. The main benefit of this work is a comprehensive procedure from point clouds to final trajectory for Unmanned Aerial Vehicle.

Keywords: UAV, trajectory panning, LIDAR, point cloud, polygonal mesh, travel salesman problem, A* in 3D, 2-opt

Title translation: UAV trajectory planning for visual inspection of 3D objects

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

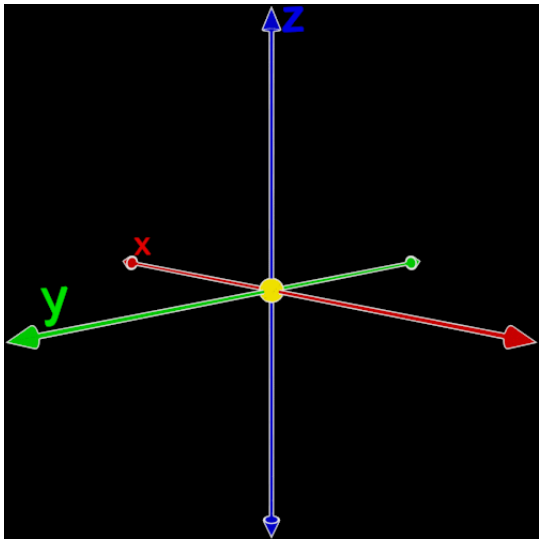
Využití bezpilotních prostředků se v dnešní době stále rozrůstá. Jedním z těchto využití je i provádění inspekcí nejrůznějších objektů. V dnešní době je k provedení inspekce zapotřebí, kromě bezpilotního prostředku s kamerou, také odborné obsluhy. Tato obsluha má na starosti ovládání bezpilotního prostředku a také ovládání kamery. To znamená, že tato obsluha musí mít zkušenosti s ovládáním bezpilotních prostředků. Cílem této práce je na základě mračna bodů, pořízeného zařízením LIDAR, provést *offline* naplánování trasy pro inspekci objektu z mračna bodů. Čímž docílíme zpřesnění, zlevnění a automatizace procesu celé inspekce.

Celá práce je rozdělena do šesti kapitol. Druhá kapitola se zabývá strojovou reprezentací prostorových objektů. Jsou zde popsány tři základní strojové reprezentace a jejich způsoby popisu. V závěru druhé kapitoly je proveden výběr způsobu reprezentace, který bude použit v rámci řešení této práce. V rámci třetí kapitoly je představen princip algoritmu BPA, který je použit pro vytvoření polygonálního modelu z mračna bodů. Jsou zde také popsány nedostatky tohoto algoritmu. Závěr této kapitoly se zabývá kvalitou výsledného polygonálního modelu v závislosti na velikosti mračna bodů, při použití algoritmu BPA. Čtvrtá kapitola se zabývá vypočtením polohy bodů, ze kterých se bude provádět inspekce. Jelikož těchto bodů je hodně je ve čtvrté kapitole popsán i princip na jehož základě dojde k redukci počtu těchto bodů. V rámci páté kapitoly jsou stanoveny podmínky pro automatické generování stavů stavového prostoru. Jelikož tyto podmínky dovolují za určitých okolností vygenerování nežádoucích stavů, tak je v této kapitole popsáno jedno manuální a jedno automatické řešení vzniklého problému. Šestá kapitola se zabývá vyhledáváním ve stavovém prostoru z páté kapitoly a řešením problému obchodního cestujícího pro body z kapitoly čtvrté. Sedmá kapitola se zabývá porovnáním naplánované trajektorie inspekce a trajektorie, kterou poté letělo UAV v simulačním programu AMP Planner.

Kapitola 2

Strojová reprezentace prostorových objektů

Základem trojrozměrné geometrie je kartézská soustava souřadnic. Soustava je pojmenována po francouzském filozofovi a matematikovy René Descartes, který ji v 17. století formuloval. V eukleidovském (trojrozměrném) prostoru má kartézská soustava souřadnic tři osy. Tyto osy jsou na sebe vzájemně kolmé a protínají se v jednom bodě, který označujeme jako střed soustavy se souřadnicemi (0,0,0). Bod v prostoru je potom definován vzdáleností od počátku na každé ze tří os.Vizuálně je kartézská soustava souřadnic zobrazena na obrázku 2.1.



Obrázek 2.1: Kartézská soustava souřadnic v eukleidovském prostoru. Žlutá koule označuje střed prostoru.Obrázek převzat z [16]

S využitím prostorové geometrie si v následujících kapitolách představíme tři základní strojové reprezentace prostorových objektů.

2.1 Hraniční reprezentace

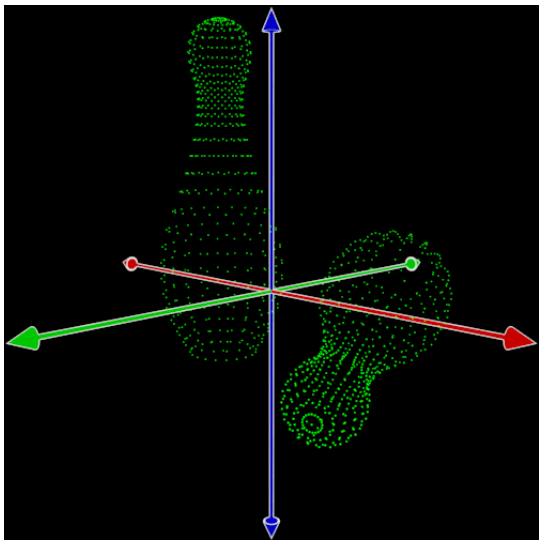
Jedná se o jednu z nejobvyklejších reprezentací těles [10]. Hraniční reprezentace popisuje pouze povrch tělesa, nikoli jeho vnitřní strukturu [16]. To

znamená, že výsledné modely jsou duté.

V následujícím textu si popíšeme některé způsoby popisu těles pomocí hraniční reprezentace.

2.1.1 Mračno bodů

Mračno bodů je množina bodů v souřadnicovém systému. Může být použita i kartézská soustava souřadnic.Mračna bodů se získávají pomocí 3D skenerů, jako je například LIDAR. Obecně nejsou mračna bodů používána ve většině 3D aplikací [8] jelikož jsou často konvertována na polygonální sítě či NURBS křivky. Na obrázku 2.2 jsou vidět dvě mračna bodů reprezentující bowlingové kuželky.



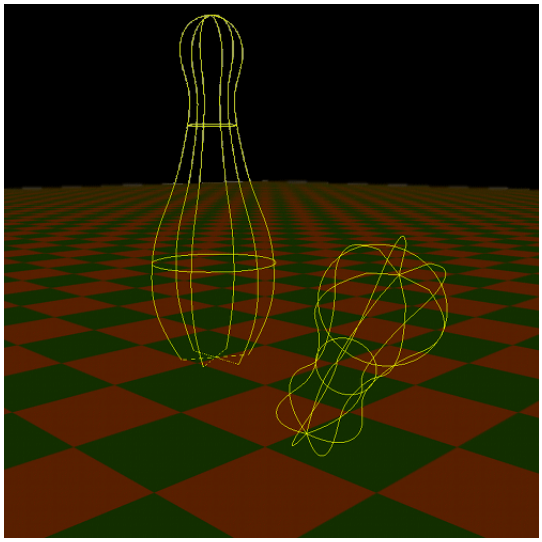
Obrázek 2.2: Mrak bodů složený z povrchových bodů tělesa. Obrázek převzat z [16].

2.1.2 Křivky NURBS

NURBS křivky jsou matematickým modelem běžně používaný v počítačové grafice pro generování a reprezentování křivek a ploch, které nabízejí velkou flexibilitu a přesnost při manipulaci jak s analytickými, tak s volnými tvary [6]. Křivka NURBS je definována pořadím, skupinou kontrolních bodů s různou váhou a uzlovými vektory. Na obrázku 2.3 jsou zobrazeny dvě bowlingové kuželky pomocí NURBS křivek.

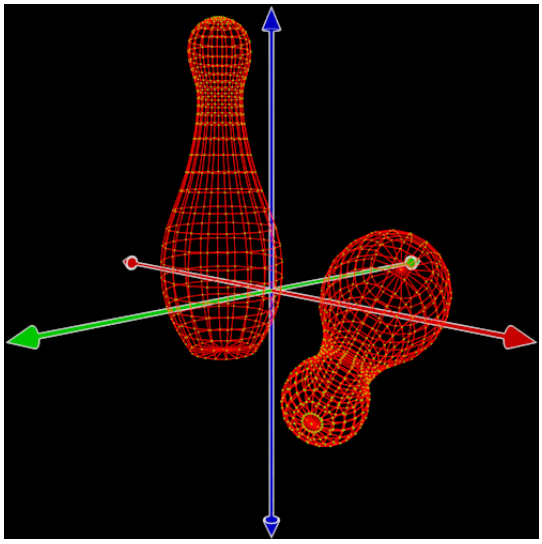
2.1.3 Polygonální síť

Při modelování objektů pomocí polygonální sítě, se snažíme aproximovat jejich povrch pomocí polygonů. Nejjednodušším polygonem je trojúhelník, neboť trojúhelník je nejjednodušším plošným útvarem. Tato reprezentace je rozšířená především v zábavní sféře (modely pro filmy či 3D počítačové hry),



Obrázek 2.3: Objekty definované pomocí křivek NURBS. Obrázek převzat z [16]

jelikož tyto modely mohou být akcelerovány současným široce dostupným hardwarem [16]. Nevýhodou této reprezentace je, že nedokáže přesně reprezentovat zakřivené plochy, které se tedy musí aproximovat velkým počtem polygonů aby se dosáhlo vizuálně uspokojivého výsledku. Proto při modelování komplexních povrchů dochází ke zpomalení. Na obrázku 2.4 jsou dvě polygonální sítě reprezentující dvě bowlingové kuželky.



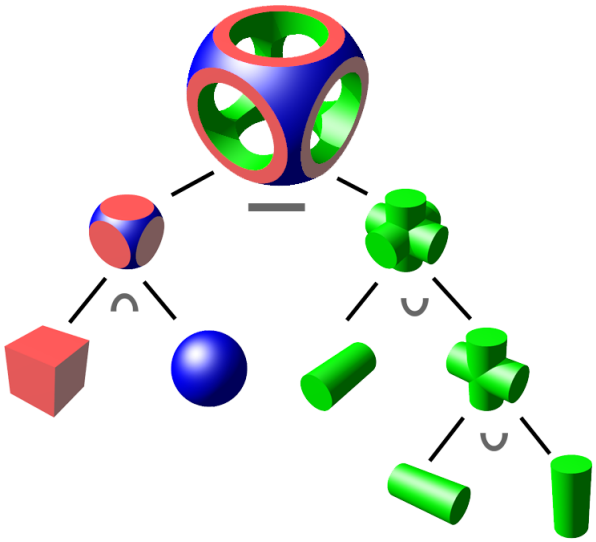
Obrázek 2.4: Polygonální sítě zobrazující dvě bowlingové kuželky. Obrázek převzat z [16].

2.2 Objemová reprezentace

U této reprezentace se kromě povrchu těles popisuje i jejich objem. Tato reprezentace pracuje s informací, zda je bod v prostoru součástí nějakého objektu či nikoliv. S objemovou reprezentací se setkáme především v technických oblastech a medicíně [16].

2.2.1 Konstruktivní geometrie těles (CSG)

Konstruktivní geometrie těles umožňuje vytvářet komplexní modely. Tyto modely jsou vytvářeny z jednodušších objektů pomocí Booleových množinových operací [3].Na obrázku 2.5 je znázorněno vytvoření složitějšího objektu z jednodušších spolu s vyznačenými použitými operacemi.



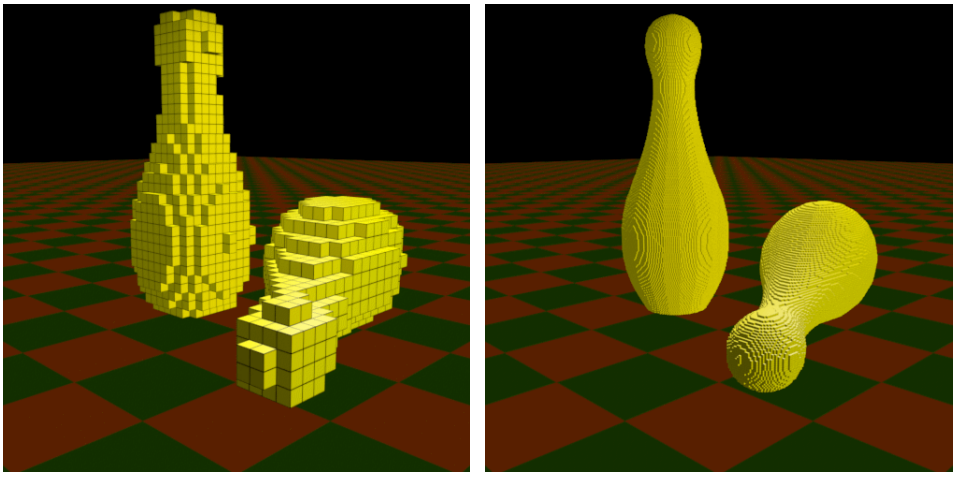
Obrázek 2.5: Vytváření objektů pomocí Booleových operací v CSG. Obrázek převzat z [3]

2.2.2 Vyčíslení objemu obsazeného tělesem

Tato reprezentace pracuje s tzv. voxely, což jsou objemové bloky v prostoru. Tato metoda se používá v medicíně při zobrazování dat získaných z přístrojů CT či magnetické rezonance [16]. Velikostí samotného voxelu se přímo určuje přesnost výsledného modelu. Na obrázcích 2.6 můžeme vidět rozdíl v přesnosti výsledného modelu při použití různých velikostí voxelů.

2.3 Procedurální reprezentace

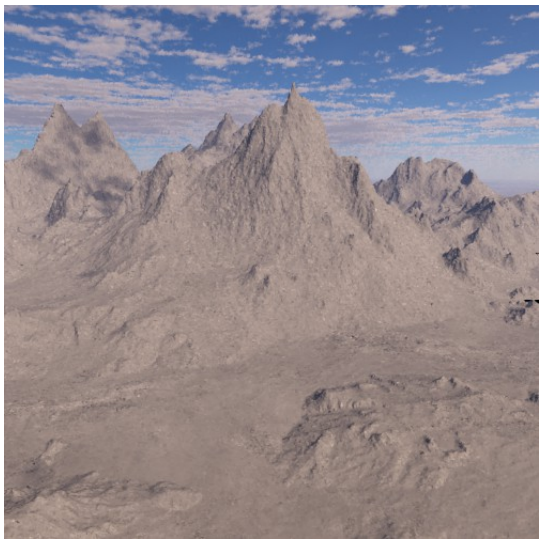
Tělesa u procedurální reprezentace jsou definována algoritmy. Po zadání parametrů algoritmům jsou objekty automaticky vytvořeny. Při generování



(a) : Aproximace velkými voxely **(b) :** Aproximace malými voxely

Obrázek 2.6: Reprezentace bowlingových koulí pomocí voxelů s různou velikostí. Obrázky převzaty z [16].

lze použít například fraktály, výsledné tvary tak mohou být velmi komplexní [16]. Model pohoří vytvořený pomocí procedurální reprezentace se nachází na obrázku 2.7.



Obrázek 2.7: Procedurálně reprezentovaný model pohoří. Obrázek převzat z [16].

2.4 Výběr prostorové reprezentace pro plánování trajektorie

Jelikož při plánování letové trasy vystačíme s informací o povrchu těles, budeme vybírat jednu z hraničních reprezentací. Nabízí se použít přímo mračno bodů, jelikož ho dostaneme jako výsledek po skenování LIDARem.

Avšak naším cílem je provést inspekci povrchu tělesa, musíme tedy mít tento povrch k dispozici. Povrch tělesa získáme z mračna bodů právě tím, že ho konvertujeme na polygonální síť či NURBS křivky. Pro náš účel je nejlepší použít polygonální síť a to z následujících důvodů:

1. Operace prováděné s polygonálními sítěmi jsou nejméně časově náročné.
2. Polygonální sítí můžeme zrekonstruovat povrch z mračna bodů.
3. Pro plánování trajektorie vystačíme s hrubou aproximací povrchů. Nepotřebujeme detailní model.

Kapitola 3

Získání polygonálního modelu objektu z mračna bodů

3.1 Získání mračna bodů

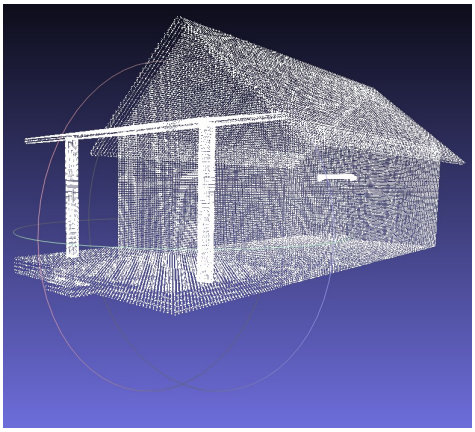
Než budeme moci provést inspekci objektu, je nejdříve nutné provést naskenování objektu samotného. K tomuto účelu nám poslouží metoda dálkového průzkumu. Tato metoda využívá pulzní laser k přesnému měření vzdálenosti. Zařízení LIDAR se skládá z laseru, snímače a speciálního GPS přijímače [15]. Výstupem ze zařízení LIDAR je soubor ve formátu .las, který obsahuje mračno bodů. Data, která může formát .las obsahovat jsou na obrázku 3.1. S mračnem bodů budeme dále pracovat v prostředí MeshLab, kde z mračna bodů vytvoříme polygonální síť. Bohužel MeshLab neumí s formátem .las pracovat, proto musíme formát .las převést na jiný formát. Pro náš účel stačí vědět pouze polohu (souřadnice x,y,z) každého bodu v mračnu. Použijeme program LasUtility, který převede soubor ve formátu .las na soubor ve formátu .xyz , kde jsou na každém řádku uvedeny souřadnice jednoho bodu. Tento formát již dokáže Meshlab otevřít a pracovat s ním.

Item	Format	Size	Required
X	long	4 bytes	*
Y	long	4 bytes	*
Z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits (bits 0, 1, 2)	3 bits	*
Number of Returns (given pulse)	3 bits (bits 3, 4, 5)	3 bits	*
Scan Direction Flag	1 bit (bit 6)	1 bit	*
Edge of Flight Line	1 bit (bit 7)	1 bit	*
Classification	unsigned char	1 byte	*
Scan Angle Rank (-90 to +90) – Left side	char	1 byte	*
User Data	unsigned char	1 byte	
Point Source ID	unsigned short	2 bytes	*

Obrázek 3.1: Přehled dat, které může soubor s příponou .las obsahovat. Tabulka převzata z [5]

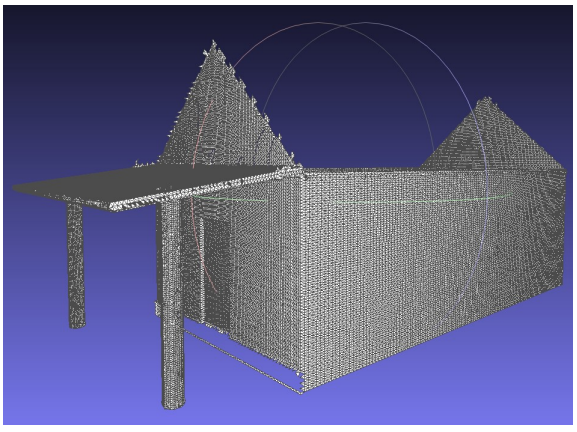
3.2 Převod mračna bodů na polygonální model

Program Meshlab nám vizuálně zobrazí mračno bodů a umožní nám s ním dále pracovat. V prostředí programu MeshLab vytvoříme polygonální model, který dále použijeme pro plánování trajektorie. Na obrázku 3.2 je mračno bodů reprezentující dům v prostředí MeshLab. Model domu z obrázku 3.2 stažen z [17].



Obrázek 3.2: Zobrazení mračna bodů o velikosti 107 622 bodů v prostředí MeshLab.

Pro rekonstrukci povrchu objektu použijeme algoritmus BPA, který jsi blíře představíme v sekci 3.3. Před spuštěním algoritmu se musí nastavit několik parametrů, které ovlivňují kvalitu výsledné sítě polygonů. Avšak prostředí MeshLab dokáže toto nastavení provést automaticky. Necháme MeshLab aby automaticky nastavil algoritmus BPA a podíváme se na výsledek rekonstrukce povrchu, ten je zobrazen na obrázku3.3.



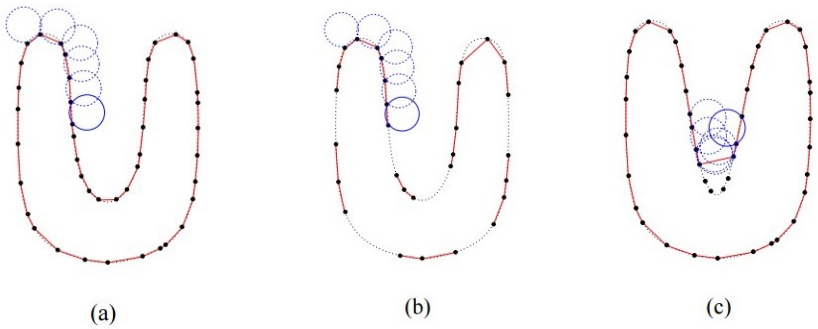
Obrázek 3.3: Neúplná polygonální síť domu.

Pokud porovnáme mračno bodů z obrázku 3.2 a z něj vzniklou polygonální síť z obrázku 3.3. Na první pohled vidíme, že domu chybí střecha a podlaha

verandy. Máme tedy polygonální model domu, ovšem je neúplný. Tato neúplnost je způsobena algoritmem BPA, který jsme použili pro rekonstrukci povrchu. V další kapitole 3.3 popíšeme jak algoritmus BPA pracuje a vyřešíme problém s neúplností modelu.

3.3 Princip použitého algoritmu pro vytvoření polygonálního modelu

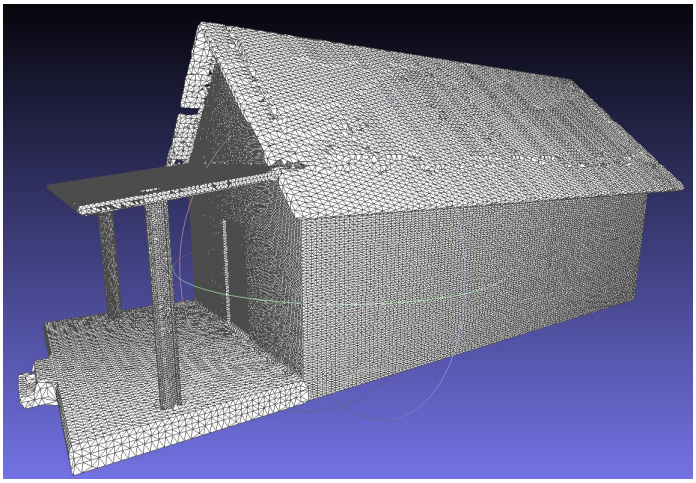
Algoritmus počítá trojúhelníkovou síť interpolací daného mračna bodů. Princip BPA je jednoduchý: Tři body vytvoří trojúhelník, jestliže se koule o poloměru k (který definuje uživatel) dotkne bodů a přitom koule nebude obsahovat žádný jiný bod. Algoritmus začíná se *seed triangle*, což je první trojúhelník, od kterého se začíná vytvářet síť. Poté co je *seed triangle* nalezen, jsou okolo jeho hran vytvářeny koule (koule se musí dotýkat právě dvou bodů již existujícího trojúhelníku) dokud se nedotkne dalšího bodu, čímž dojde k vytvoření trojúhelníku. Tento proces pokračuje dokud nebyly vyzkoušeny všechny hrany. Poté začne znovu od jiného *seed triangle*, dokud nebudou všechny body vyzkoušeny. Výhodou tohoto algoritmu je malá paměťová náročnost a rychlost zpracování. Algoritmus pracuje nejlépe, jsou-li body ekvidistantní [18].



Obrázek 3.4: Algoritmus BPA ve 2D. Obrázek převzat z [18]

Na obrázku 3.4a vidíme výsledek rekonstrukce povrchu, pokud je správně zvolen poloměr k kružnice a vzdálenost sousedních bodů není větší něž průměr kružnice. Obrázek 3.4b znázorňuje případ, kdy vzdálenost mezi sousedními body je větší než průměr kružnice. To má za následek, že ve výsledném zrekonstruovaném povrchu jsou díry. Na posledním obrázku 3.4c je případ kdy, zakřivení modelu je větší než $1/k$, což má za příčinu, že některé body budou při rekonstrukci zcela vynechány. To má za následek, že výsledný zrekonstruovaný povrch je méně přesný.

Důvod, proč v modelu z obrázku 3.3 chybí střecha a část verandy je stejný jako na obrázku 3.4b kde zapříčinil díry ve zrekonstruovaném povrchu. Tedy vzdálenost sousedních bodů na střeše a části verandy je jiná než na zbytku domu. Jelikož MeshLab dovoluje jednotlivé výsledky algoritmu BPA slučovat, stačí pouštět algoritmus BPA opakovaně s rozdílným poloměrem k.



Obrázek 3.5: Úplná polygonální síť domu.

Již úplná polygonální síť domu je na obrázku 3.5.

3.4 Otestování kvality výsledného modelu v závislosti na velikosti mračna bodů

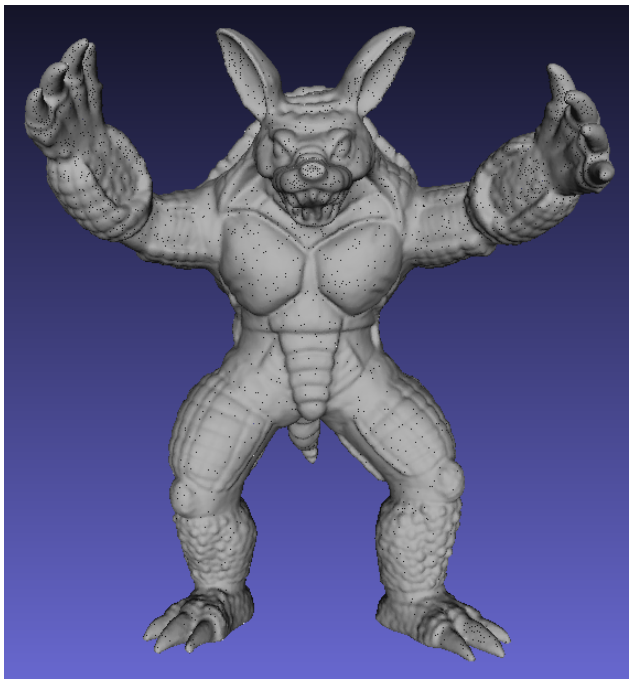
Otestujme nyní algoritmus BPA a sledujme časovou náročnost a kvalitu zrekonstruovaného povrchu. Provedeme sérii rekonstrukcí povrchu s proměnným počtem bodů. Výchozí polygonální model 3.6 byl stažen z [11]. Tento polygonální model se skládá z 345 944 trojúhelníků, které jsou tvořeny 172 974 body. Jednodušší modely (s menším počtem bodů) vytvoříme tak, že použijeme *Monte Carlo sampling*, u kterého můžeme zadat výsledný počet vzorků. Začněme tedy nejprve se 172 974 vzorky a budeme postupně počet vzorků snižovat o dvě třetiny. Jelikož výsledná rekonstrukce je závislá na zvoleném poloměru k, tak jako výsledek vezmeme sjednocení prvních třech k s největším počtem zrekonstruovaných trojúhelníků. Naměřené hodnoty se nachází v tabulce 3.1 spolu s referencemi na příslušné obrázky. Algoritmus byl pouštěn na jednom vlákně procesoru Intel Core i5-3750k.

Počet bodů	Počet trojúhelníků	Časová náročnost BPA [s]	Obrázky
172 974	317 310	925.3	3.7
57 658	105 652	121.4	3.8
19 219	35 143	22.4	3.9
6 406	11 601	6.8	3.10
2 135	3 750	3.9	3.11

Tabulka 3.1: Časová náročnost algoritmu BPA

Z tabulky 3.1 vidíme, že při první rekonstrukci, která měla stejný počet bodů jako originál, jsme přišli o přibližně 8% trojúhelníků tvořících povrch modelu. Porovnáme-li obrázky 3.6b a 3.7b vidíme, že tento úbytek je znatelný,

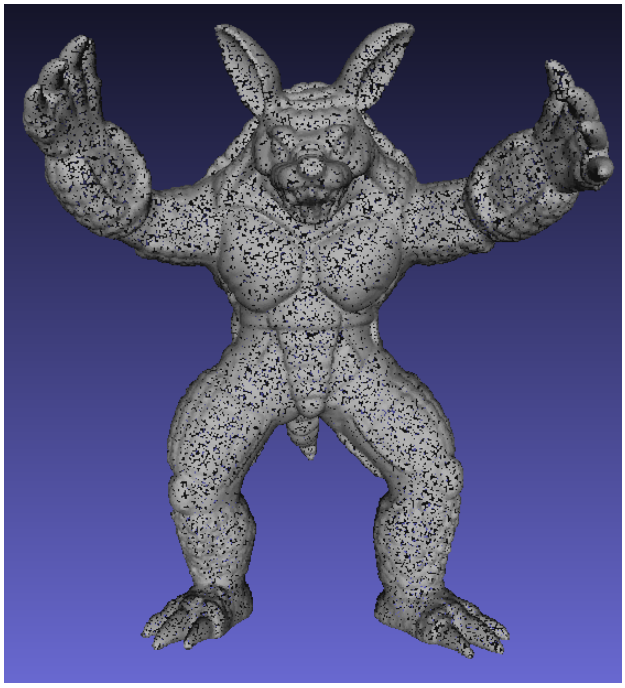
(a) : Mračno bodů



(b) : Polygonální síť

Obrázek 3.6: Originální model stažený z [11]

avšak detaily zůstávají zachovány. Další rekonstrukce byla z 57 658 bodů, výsledný zrekonstruovaný model, který je na obrázku 3.8b stále zachovává některé detaily, avšak v oblasti hlavy jsou již znatelné nepřesnosti. Zlomovou rekonstrukcí je ta ze 6 406 bodů, která je na obrázku 3.10. Detaily již

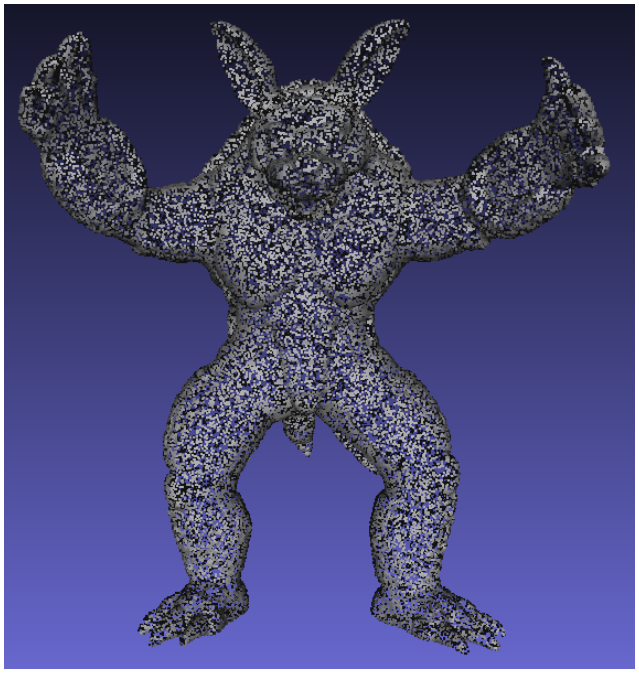


(a) : Mračno bodů

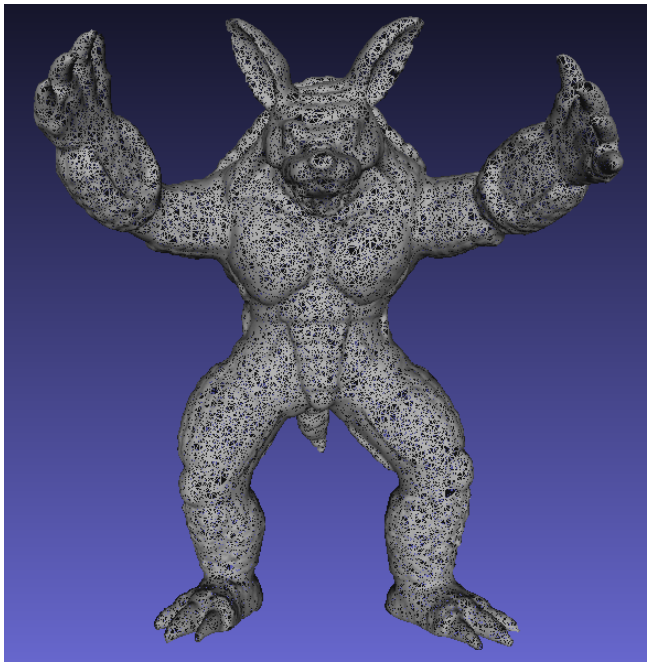
(b) : Polygonální síť

Obrázek 3.7: Rekonstrukce ze 172 974 bodů

rozhodně zachovány nejsou, ovšem obrys je modelu je stále zachován. Při poslední rekonstrukci ze 2 135 bodů došlo již ke značným nepřesnostem, které jsou obzvláště vidět v oblasti končetin. Tato rekonstrukce je na obrázku 3.11.

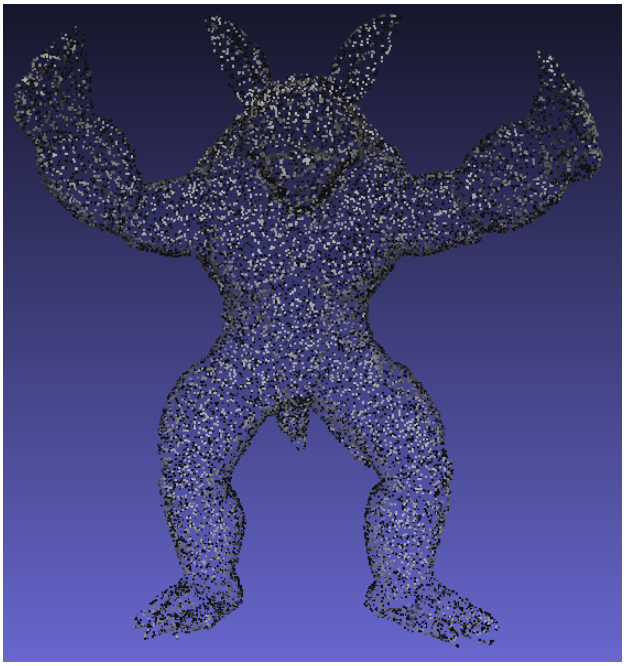


(a) : Mračno bodů

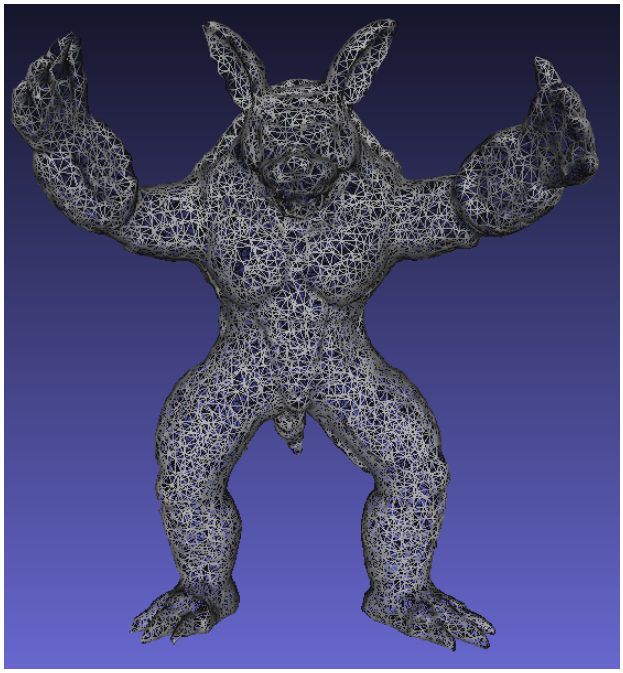


(b) : Polygonální síť

Obrázek 3.8: Rekonstrukce z 57 658 bodů

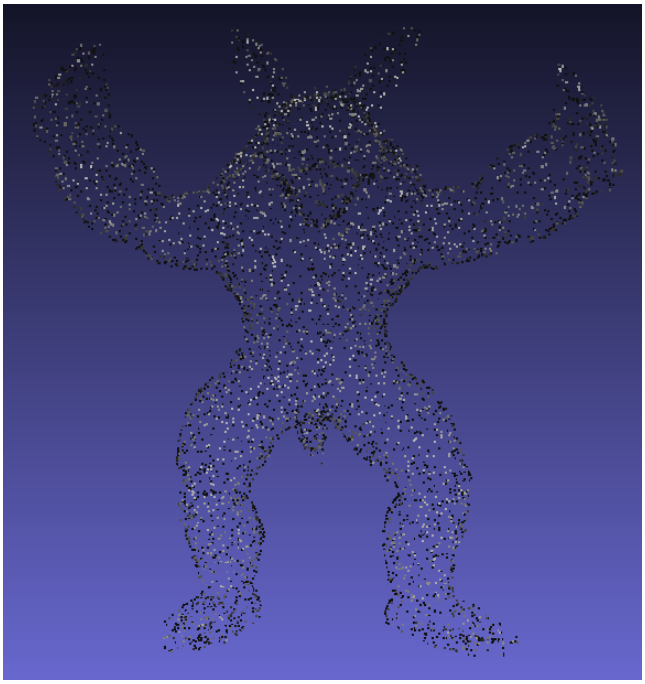


(a) : Mračno bodů

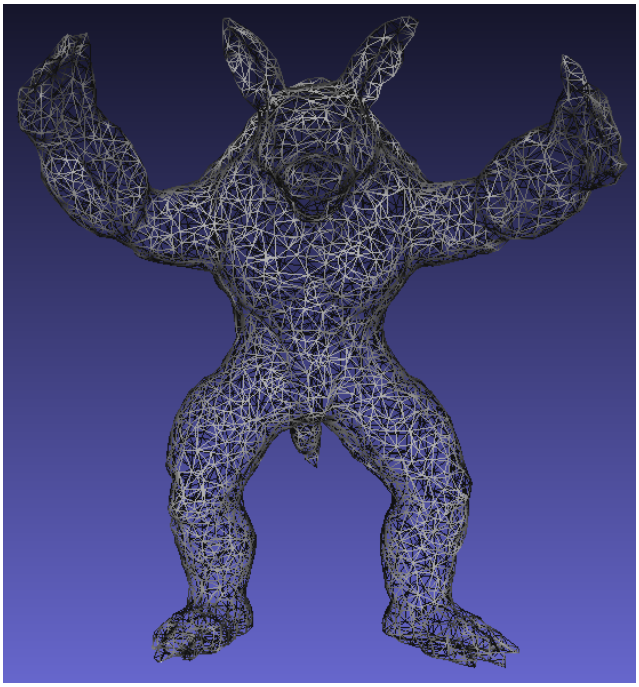


(b) : Polygonální síť

Obrázek 3.9: Rekonstrukce z 19 219 bodů

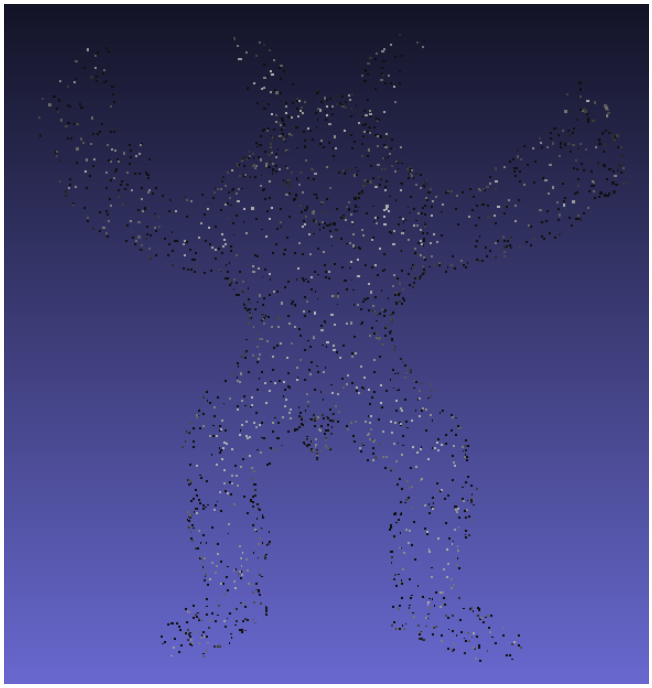


(a) : Mračno bodů

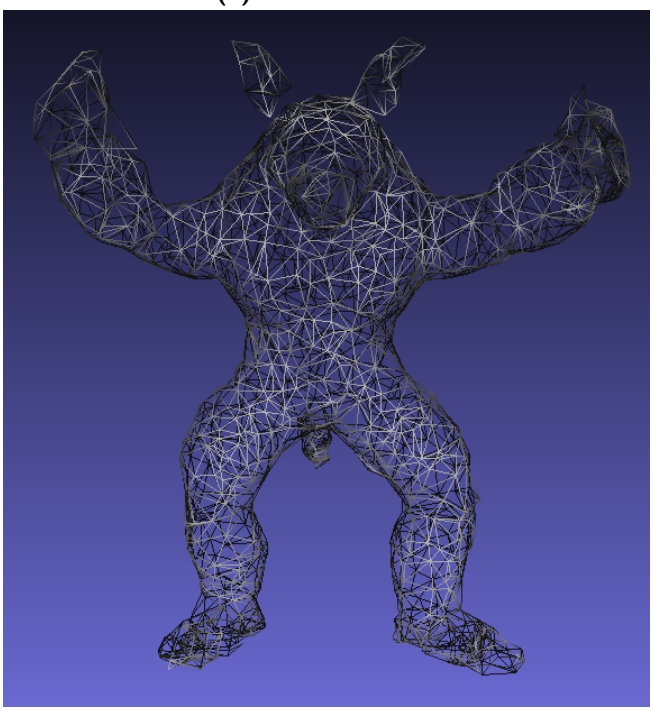


(b) : Polygonální síť

Obrázek 3.10: Rekonstrukce z 6 406 bodů



(a) : Mračno bodů



(b) : Polygonální síť

Obrázek 3.11: Rekonstrukce z 2 135 bodů

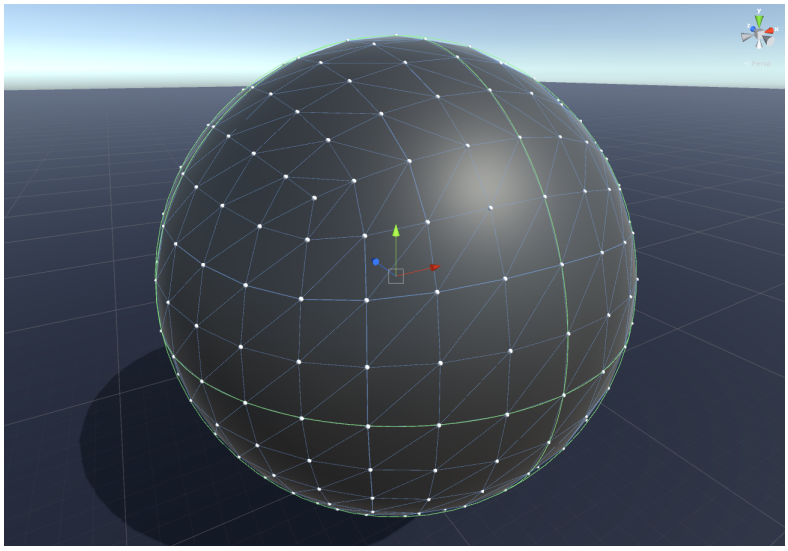
Kapitola 4

Výpočet polohy inspekčních bodů

Než budeme schopni sestavit letovou trasu pro UAV, musíme nejdříve vypočítat polohu bodů v prostoru, ze kterých se bude inspekce provádět. Žádný software, který jsme do této doby použili nám něco takového neumožňuje. Pro samotné plánovaní letové trasy použijeme herní engine Unity 3D [12]. Unity 3D nám umožní jednoduše vizuálně zobrazovat naše výsledky a implementovat systém pro plánování trajektorií. Můžeme také do prostředí Unity importovat již vytvořené modely a dále s nimi pracovat. Při tomto nahrávání je důležité nechat Unity vytvořit *Collider*. *Collider* je komponenta, která definuje tvar modelu pro fyzikální účely, více o *Collider* v dokumentaci zde [13].

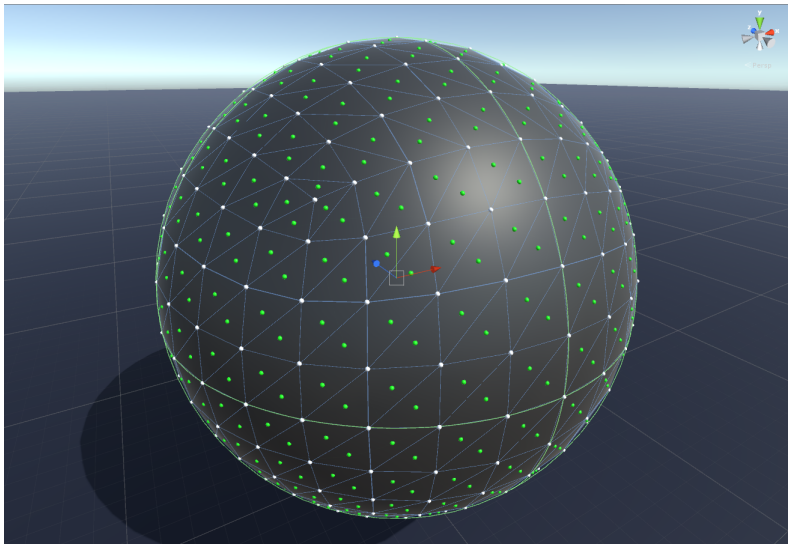
4.1 Výpočet polohy inspekčních bodů z modelu objektu

Z těchto bodů se bude skládat trasa, kterou UAV poletí při inspekci objektu. Musíme tedy zajistit dva hlavní požadavky. Za prvé musí být zajištěno, že tyto body budou v dostatečné vzdálenosti od modelu, jinak by mohlo dojít ke kolizi. Za druhé tyto body musejí být rozmístěny tak, aby bylo možné provést inspekci celého modelu. Víme, že povrch modelu je tvořen jednotlivými trojúhelníky, můžeme tedy využít tyto trojúhelníky abychom zajistili inspekci celého modelu. Na příkladu koule 4.1 vidíme, že celý je povrch je rozdělen na trojúhelníky se zhruba stejnou plochou.



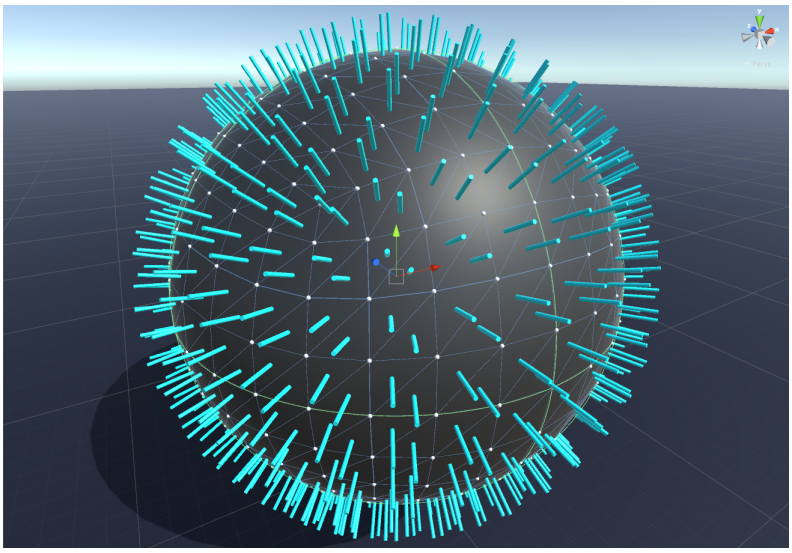
Obrázek 4.1: Koule o poloměru libovolném poloměru s vyznačenými trojúhelníky a vrcholy (bílé body).

Pokud tedy zajistíme inspekci každého trojúhelníku, bude zajištěna i inspekce celého modelu. Pomocí trojúhelníků můžeme také zajistit náš první požadavek, jelikož trojúhelník je tvořen třemi body viz 4.1. Tři body nám stačí k určení roviny a také jejího normálového vektoru. Abychom věděli, ke kterému trojúhelníku patří který normálový vektor, zvolíme jako počáteční bod vektoru těžiště příslušného trojúhelníku. Na obrázku 4.2 vidíme předešlý příklad koule z 4.1, nyní již s vypočtenými těžišti pro jednotlivé trojúhelníky.



Obrázek 4.2: Koule libovolném poloměru s vyznačenými trojúhelníky ,vrcholy (bílé body) a těžišti (zelené body).

Abychom mohli zobrazit normálový vektor, je třeba ještě definovat koncový bod. Nechť je koncový bod vektoru ve směru příslušné normály a ve vzdálenosti x od příslušného těžiště, kde x kladné číslo. Takové definici vyhovují právě dva body, mi si ovšem vybereme vždy ten, kdy normálový vektor neprochází modelem. Na obrázku 4.3 vidíme normály příslušných trojúhelníků, jejichž vzdálenost od povrchu modelu (koule) je 1 metr.



Obrázek 4.3: Koule o poloměru 15 metrů s vyznačenými trojúhelníky ,vrcholy (bíle kuličky) a normálami (tyrkysové válce)

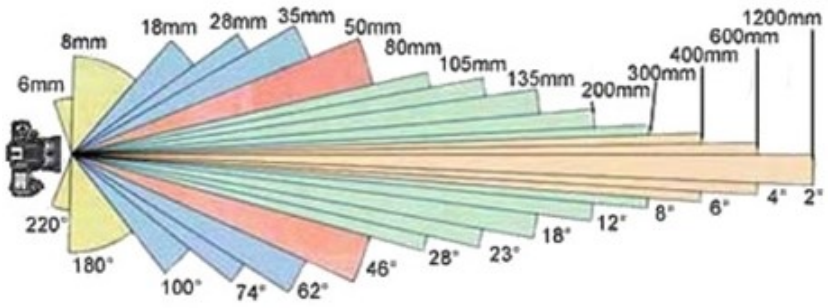
Z obrázku 4.3 vidíme, že koncové body normálového vektoru vyhovují našim dvou požadavkům, které musí splňovat body inspekce. Jelikož vzdálenost koncových bodů od povrchu modelu je nastavitelná, určíme tím přímo minimální vzdálenost (bezpečnostní vzdálenost) mezi modelem a UAV. Dále pokud směr natočení kamery UAV bude opačný, než je směr příslušného normálového vektoru, zajistíme tím i inspekci celého modelu. Musí ovšem být zajištěno, že v zorném poli kamery bude celá plocha příslušného trojúhelníku.

4.2 Redukování počtu inspekčních bodů pomocí kamery

Způsob nalezení bodů inspekce představený v sekci 4.1 je jednoduchý, ovšem jeho výsledkem je velký počet bodů, ze kterých by se měla skládat výsledná trajektorie. Kdybychom chtěli provést inspekci koule z obrázku 4.3 znamenalo by to navštívit 768 bodů, což ještě není nereálné. Ovšem v případě, že bychom chtěli provést inspekci modelu z obrázku 3.8b, znamenalo by to navštívit 57 658 bodů. Jak se dále ukáže, znamenalo by to i řešit problém obchodního cestujícího pro tento počet bodů, což by již bylo dosti časově náročné. Potřebujeme tedy najít způsob, jak zredukovat počet bodů a zároveň

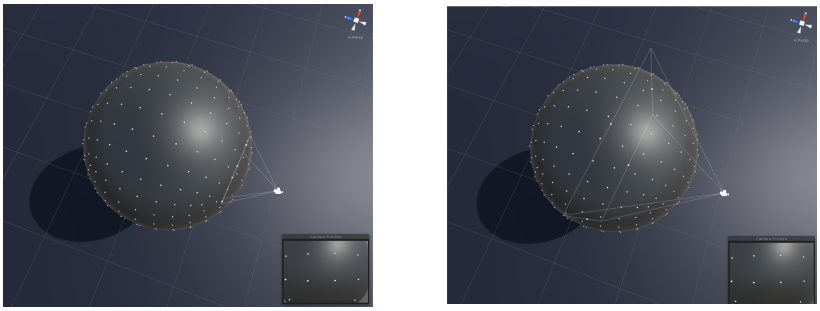
být schopni zaručit, že bude provedena inspekce celého modelu.

Vrátíme-li se ke způsobu nalezení inspekčních bodů ze sekce 4.1. Předpokládali jsme, že z jednoho inspekčního bodu uvidíme jen plochu příslušného trojúhelníku, což nemusí být vždy pravda. Jelikož vzdáleností inspekčního bodu od modelu také přímo určujeme vzdálenost mezi UAV a objektem. Je tedy nutné aby se tento inspekční bod nacházel v určité bezpečné vzdálenosti, abychom předešli kolizi UAV s objektem. Řekněme, že vzdálenost mezi inspekčním bodem od modelem bude y, kde y je kladné číslo. Čím větší bude y tím pravděpodobnější bude, že kamera zaznamená celý povrch více trojúhelníků. Co všechno zaznamenáme kamerou z jednoho bodu, také závisí na velikosti zorného pole kamery. Na obrázku níže 4.4 můžeme vidět závislost velikosti zorného pole na ohniskové vzdálenosti.



Obrázek 4.4: Přehled velikosti zorných polí v závislosti na ohniskové vzdálenosti. Obrázek převzat z [7].

Abychom mohly kameru simulovat v Unity, potřebujeme specifikovat ještě jeden parametr *Far*. Tento parametr udává vzdálenost, na kterou kamera vidí. To znamená, že objekty, které jsou dál nejsou kamerou zobrazovány. Jelikož v Unity můžeme detekovat zda je objekt v zorném poli či nikoliv, využijeme toho následovně: Řekneme, že celá plocha trojúhelníku je v zorném poli kamery, právě tehdy když jsou všechny tři vrcholy trojúhelníku v zorném poli kamery. Musíme jsi tedy dát pozor abychom tento parametr nenastavili špatně.Rozdíl mezi špatným a správným nastavením parametru *Far* je patrný z obrázku 4.5.



(a) : Správné nastavení **(b) :** Špatné nastavení

Obrázek 4.5: Nastavení parametru *Far*

Z obrázku 4.5a je vidět, že pokud budou všechny tři body trojúhelníku v zorném poli kamery, tak je celá plocha trojúhelníku viděna kamerou. Algoritmus, který použijeme pro redukci počtu bude následující:

1. Ke každému inspekčnímu bodu zapiš, jaké trojúhelníky jsou kamerou viděny.

2. Přidej všechny inspekční do seznamu *open*.

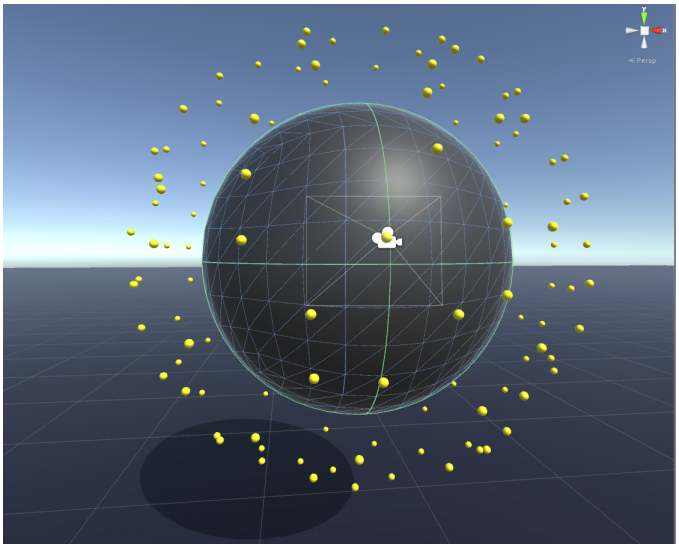
3. Vyber inspekční bod ze seznamu *open*, z kterého je vidět nejvíce trojúhelník. Tento inspekční bod přidej do seznamu *closed* a jeho trojúhelníky přidej do seznamu *records*.

4. Každému inspekčnímu bodu v seznamu *open* smaž trojúhelníky, které jsou již v seznamu *records*.

5. Pokud se v seznamu *open* nachází inspekční bod, který nemá žádné trojúhelníky, tak tento inspekční bod smaž.

6. Opakuj body 3. až 6 dokud nebude seznam *open* prázdný.

Ilustrujme funkci algoritmu na příkladu s koulí. Nastavíme poloměr koule, u které budeme chtít provést inspekci na 15 metrů. Dále vzdálenost bodu od povrchu koule nastavíme na tři metry. Parametr *Far* kamery nastavíme na 3 metry a 30 centimetrů, úhel zorného pole kamery nastavíme na 60°. Výsledek tohoto nastavení je zobrazen na obrázku 4.6.

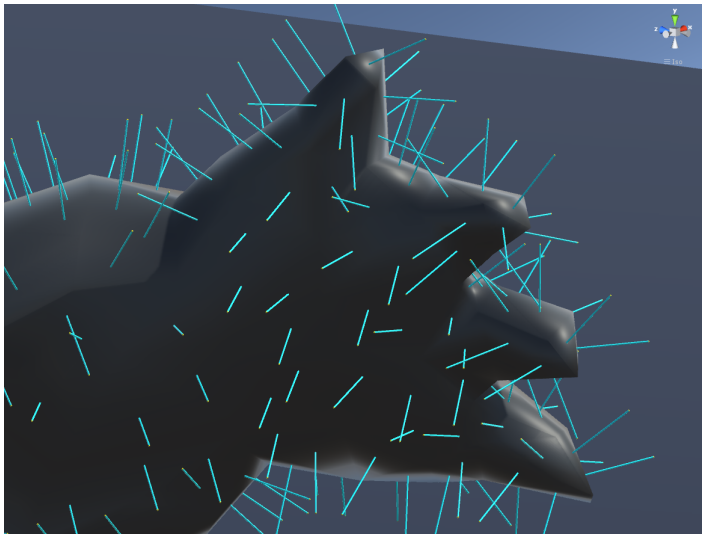


Obrázek 4.6: Redukce bodů inspekce za pomocí kamery.

Z obrázku 4.6 je patrné, že došlo k redukci bodů pro inspekci. Konkrétně jsme pomocí kamery a nastavení popsaného výše dokázali zredukovat původních 768 bodů na 148 bodů. Zredukovali jsme tedy počet bodů o 80% bodů přičemž jsme dodrželi, že bude provedena inspekce celého modelu.

4.3 Detekce překážky v blízkosti inspekčního bodu

Musíme zajistit, že v blízkosti inspekčního bodu není překážka, do které by mohlo UAV narazit. U jednoduchého modelu jako je samotná koule, se nám to stát nemůže. Ovšem v případě složitějšího modelu jako je na obrázku 4.7 se může stát, že inspekční bod bude v blízkosti překážky (prostor mezi prsty).



Obrázek 4.7: Inspekční body, které jsou blízko modelu

Skutečnost, že inspekční bod bude v blízkosti modelu ještě není ta nejhorší varianta. Jelikož se také může stát, že inspekční bod bude uvnitř modelu. Musíme tedy u každého inspekčního bodu zkontrolovat dvě podmínky. První podmínkou je, že inspekční bod nesmí být uvnitř modelu. Druhou podmínkou je, že vzdálenost mezi inspekčním bodem a modelem musí být vetší nebo rovna bezpečnostní vzdálenost. Pokud inspekční bod nesplní některou podmínku, musíme ho smazat. To má za následek, že není zaručeno provedení inspekce celého modelu.

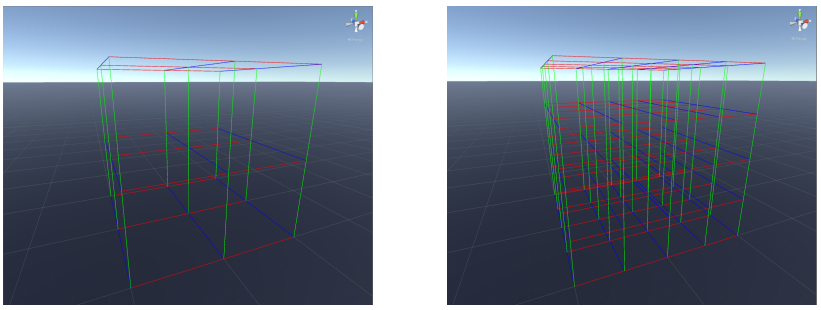
Kapitola 5

Vytvoření stavového prostoru ve 3D

Díky nalezení inspekčních bodů víme, kam se potřebujeme dostat. Ovšem nemůžeme trajektorii UAV naplánovat jako přímé spojení jednotlivých inspekčních bodů. Následkem takového řešení by velmi pravděpodobně byla kolize UAV s objektem. Je tedy potřeba jednoznačně určit kudy může v prostoru vést trajektorie UAV.

5.1 Rozdělení vymezeného prostoru na jednotlivé krychle

Jelikož velikosti objektů, pro které budeme chtít provádět inspekce, budou proměnlivé. Bude tedy i velikost prostoru, ve kterém se bude nacházet objekt, proměnlivá. Budeme se vždy snažit aby tento vymezený prostor byl co nejmenší, ale zároveň aby se do něho vešel celý objekt. Tento vymezený prostor rozdělíme na jednotlivé buňky. Buňka je krychle, která ve vymezeném prostoru zabírá předem stanovený objem. Jelikož velikost stavového prostoru je proměnná, je i velikost objemu buňky proměnná. Rozdělení vymezeného prostoru je vizuálně zobrazeno na obrázku 5.1.

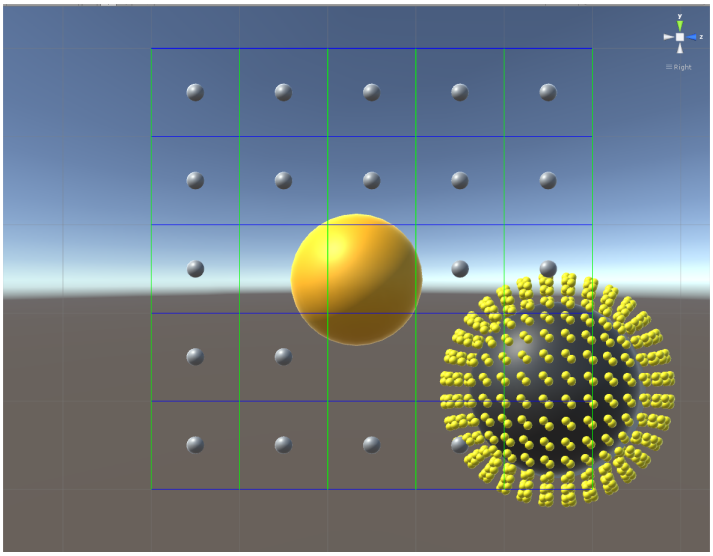


(a) : Velké buňky. **(b) :** Malé buňky

Obrázek 5.1: Rozdělení vymezeného prostoru na jednotlivé buňky.

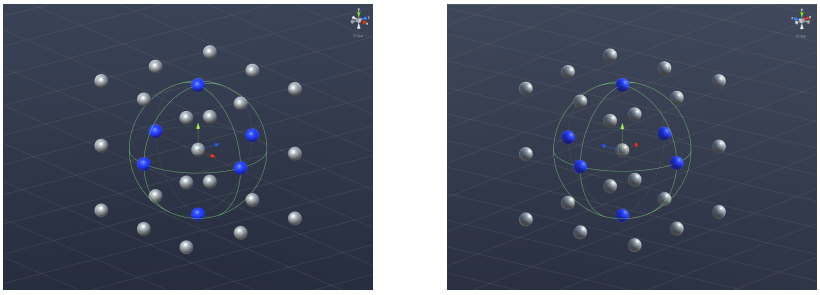
5.2 Vygenerování stavů stavového prostoru

Máme již prostor rozdělený na jednotlivé buňky. Dále v prostředku buňky vytvoříme uzel bude-li splněna podmínka, že vzdálenost mezi uzlem a jakýmkoliv objektem (náš model či překážka) bude větší, než vzdálenost mezi inspekčním bodem a modelem. Tato podmínka nám zajistí, že u výsledné trajektorie bude dodržena bezpečná vzdálenost od modelu i jiných překážek. Zda je podmínka splněna zjistíme pomocí funkce *OverlapSphere* [14], která se nachází v knihovně *UnityEngine*.Tato funkce zjistí zda se koule v zadaném místě a o zadané velikosti dotýká nějakého objektu či do něj přesahuje. Funkce pracuje s *Colliders*, proto model a překážky musejí mít *Collider*. Na obrázku 5.2 vidíme uzly (šedé body ve středu buňky) vygenerované za podmínek popsaných výše.



Obrázek 5.2: Vygenerované uzly (šedé body) v prostoru s překážkou.

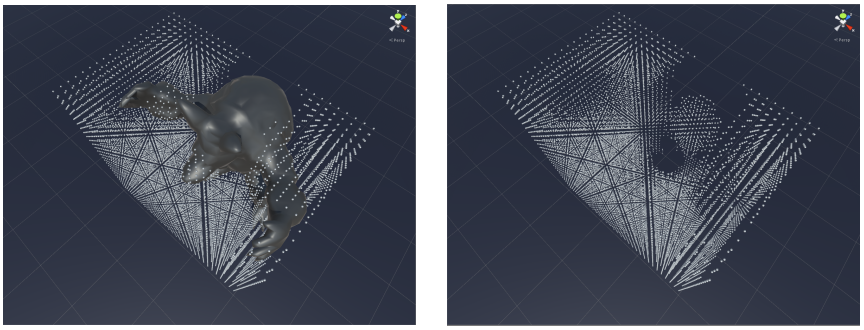
Takto vytvořené uzly budou tvořit množinu stavů stavového prostoru, který budeme později prohledávat. Dále budeme potřebovat množinu cílových stavů. Cílovými stavy jsou pro nás inspekční body, proto na jejich souřadnicích vytvoříme uzly. Abychom mohly ve stavovém prostotu vyhledávat potřebujeme také znát sousední uzly ke každému uzlu. Sousední uzel definujeme následovně : uzel B je sousedem uzlu A, jestliže vzdálenost mezi uzlem A a uzlem B je menší nebo rovna délce hrany buňky a zároveň přímka mezi těmito uzly neprochází žádnou překážkou. Sousední uzly jsou podle této definice vyznačeny modře na obrázku 5.3.



Obrázek 5.3: Zobrazení sousedních uzlů (modré body).

5.3 Odstranění uzlů z vnitřku modelu

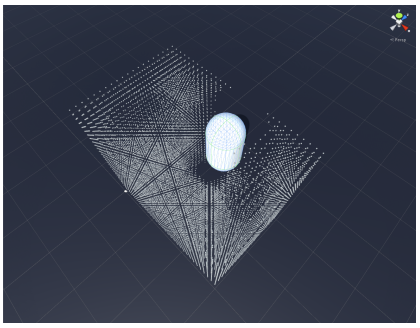
Jelikož je náš model dutý, tak dochází ke generování uzlů právě uvnitř modelu. Tato skutečnost je demonstrována obrázkem 5.4. Na obrázku 5.4a je zobrazen model, okolo kterého jsme vygenerovali *Nodes*. Vnitřek modelu je poté zachycen na obrázku 5.4b z něhož můžeme vidět, že opravdu dochází ke generování uzlu i ve vnitřku modelu.



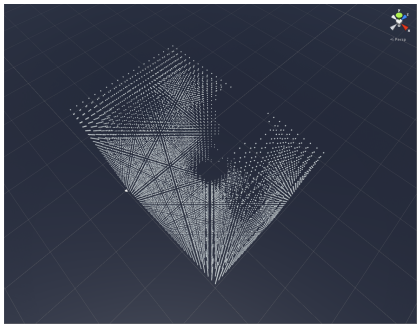
(a) : S modelem **(b) :** Bez modelu

Obrázek 5.4: Zobrazení vygenerovaných uzlů.

Tento problém můžeme vyřešit dvěma způsoby. První řešení je jednoduché a účinné, spočívá v tom, že na místa kde se vygenerovaly nechtěné uzly umístíme překážku (např. krychli). Tím zařídíme, že se tyto uzly příště již nevygenerují, což jsi můžeme ověřit na obrázku 5.5.



(a) : Umístění překážky.

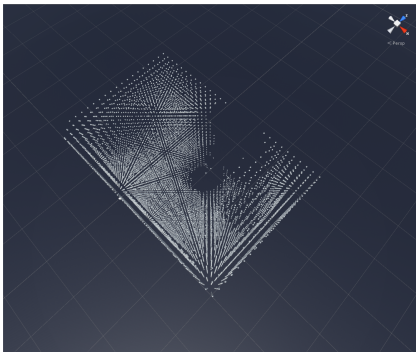


(b) : Po následujícím vygenerování.

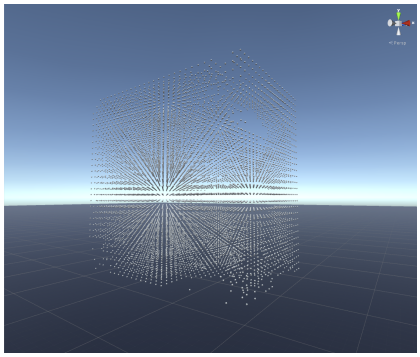
Obrázek 5.5: Odstranění nežádoucích uzlů pomocí překážky.

Tento přístup může být ovšem dosti časově náročný a hlavně je třeba kontrolovat aby nedošlo k nechtěnému odstranění uzlů, které nejsou uvnitř modelu.

Druhým řešením tohoto problému je napsat script, který odstraní uzly z vnitřku modelu za nás. Potřebujeme tedy najít způsob jak rozlišit uzly, které jsou uvnitř modelu a které nikoliv. Bohužel Unity nemá žádnou funkci, která by nám toto přímo umožnila. Řešení je následující: uzel je uvnitř modelu jestliže na přímce mezí uzlem a nejbližším vrcholem modelu, který je posunut ve směru k uzlu, není žádná překážka. Toto řešení má ovšem jeden háček, nestanovuje o jakou vzdálenost se má výsledný vektor posunout. Po sérii testů s různým nastavením vzdáleností se ukázalo, že pro model z obrázku 5.4a se dosáhne nejlepšího výsledku jeli vzdálenost posunutí tři metry. Dodejme, že velikost tohoto modelu je 30 metrů x 40 metrů x 25 metrů. Výsledek tohoto řešení je na obrázku 5.6.



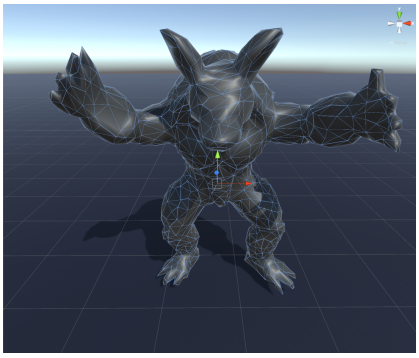
(a) : Odstranění vnitřních uzlů.



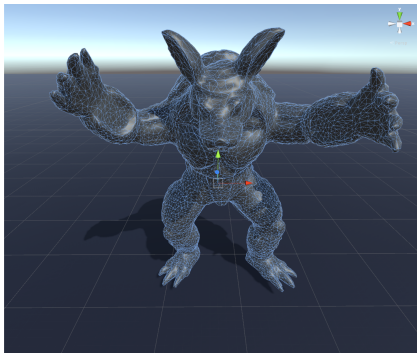
(b) : Nechtěné odstranění uzlů.

Obrázek 5.6: Odstranění nežádoucích uzlů pomocí scriptu.

Z prvního obrázku 5.6a vidíme, že došlo k odstranění většiny nežádoucích uzlů. Bohužel byli odstraněny i některé vnější uzly, což je vidět z obrázku 5.6b. Ovšem dalším testováním se ukázalo, že toto řešení je také závislé na kvalitě (počtu trojúhelníků) modelu. Dosud používaný model je tvořen 2 702 trojúhelníky 5.7a, použijeme kvalitnější model tvořený 21 620 trojúhelníky 5.7b.



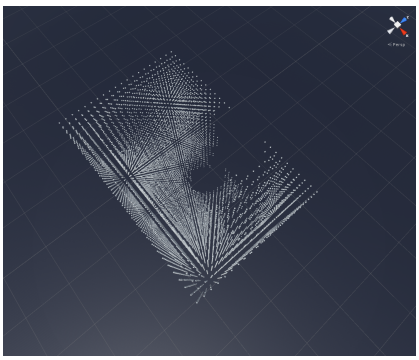
(a) : Model tvořen 2 702 trojúhelníky.



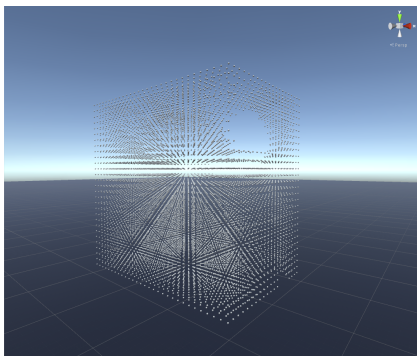
(b) : Model tvořen 21 620 trojúhelníky.

Obrázek 5.7: Porovnání modelů s různými počty trojúhelníků

Podívejme se tedy na obrázek 5.8, kde jsou zobrazeny výsledky při použití kvalitnějšího modelu. Dodejme, že všechna nastavení a i velikosti zůstali nezměněny.



(a) : Odstranění vnitřních uzlů.



(b) : Nechtěné odstranění uzlů.

Obrázek 5.8: Odstranění nežádoucích uzlů pomocí scriptu.

Je vidět výrazné zlepšení hlavně u nechtěného odstranění vnějších uzlů. Pokud jde o odstranění vnitřních uzlů nebyl nalezen jediný, který by zůstal uvnitř.

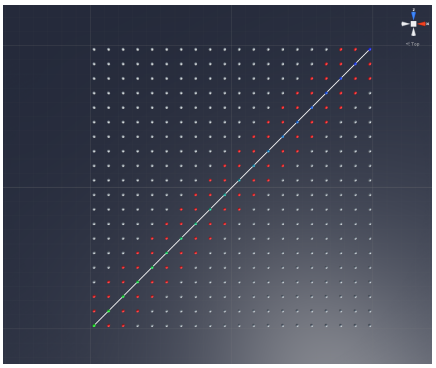
Kapitola 6

Nalezení optimální trajektorie pro inspekci trojrozměrného objektu

Než budeme moci přistoupit k sestavení výsledné trajektorie, která bude procházet všemi inspekčními body. Musíme nejdříve zvolit algoritmus, který bude schopen vyhledávat optimální cesty ve vygenerovaném stavovém prostoru. Jelikož požadujeme nalezení optimální cesty, tak zvolíme algoritmus A*.

6.1 Otestování heuristických funkcí ve vytvořeném stavovém prostoru

Časová složitost algoritmu A* je závislá na použité heuristické funkci [2]. Proto provedeme sérii testů v nichž porovnáme různé heuristiky. Naším cílem bude optimalizovat výslednou délku cesty. Porovnáme následující heuristiky : Eukleidovská metrika, Manhattanská metrika, Čebyševova metrika, Octicle metrika [4]. V rámci tohoto testování budeme uvažovat UAV o velikosti bodu, aby získané výsledky nebyli ovlivněny. V prvním testu budeme hledat cestu napříč stavovým prostorem, jelikož každá metrika nalezla stejnou cestu není nutné pro každou zvlášť mít obrázek. Tato cesta je na obrázku 6.1, vyznačena bílou linkou, červeně jsou pak označeny expandované uzly, které nejsou součástí cesty.



Obrázek 6.1: Cesta (bílá linka) nalezená všemi metrikami.

Z tabulky 6.1 vidíme, že nejrychlejší metrikou je Octicle, zhruba stejně

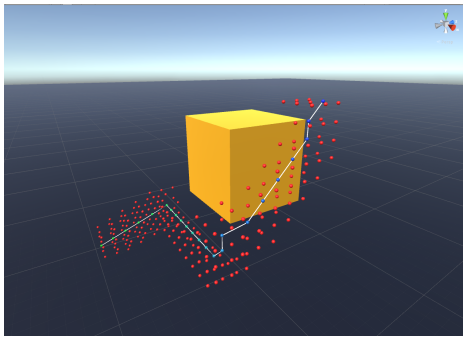
Jméno metriky	Čas [s]	Délka cesty [m]	Počet expandovaných uzlů
Eukleidovská	0.0019531251	26.87005	130
Manhattanská	0.0006332397	26.87005	130
Čebyševova	0.0006408691	26.87005	130
Octile	0.0005874634	26.87005	130

Tabulka 6.1: Výsledky metrik z prvního testu

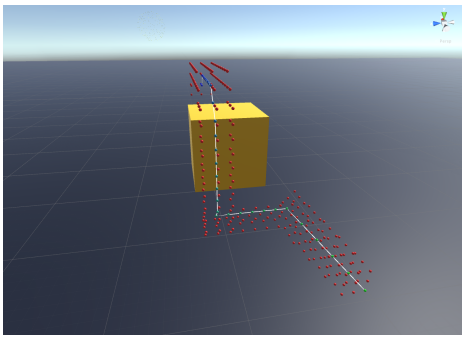
Jméno metriky	Čas [s]	Délka cesty [m]	Počet expandovaných uzlů
Eukleidovská	0.001289368	28.87758	397
Manhattanská	0.001831055	35.31711	526
Čebyševova	0.006958008	27.70601	1071
Octile	0.001770021	33.90289	508

Tabulka 6.2: Výsledky metrik z druhého testu

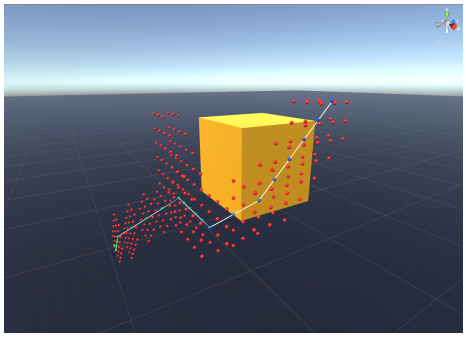
rychlé jsou metriky Manhattanská a Čebyševova. Nejpomalejší metrikou je Eukleidovská, což je hlavně způsobeno tím, že se při jejím výpočtu používá odmocnina. V druhém testu je již překážka mezi počátečním a cílovým bodem, nalezené cesty jednotlivými metrikami jsou na obrázku 6.2.



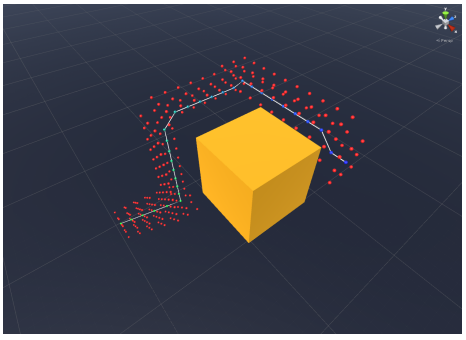
(a) : Eukleidovská metrika.



(b) : Manhattanská metrika



(c) : Čebyševova metrika



(d) : Octicle metrika

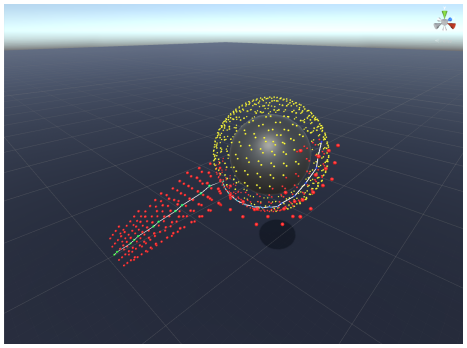
Obrázek 6.2: Cesty nalezené jednotlivými metrikami v druhém testu.

Výsledky druhého testu jsou v tabulce 6.2. Tento test nám přinesl zajímavý

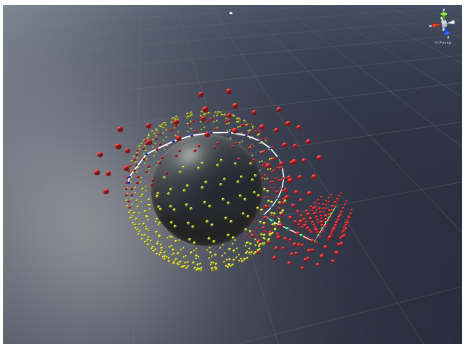
Jméno metriky	Čas [s]	Délka cesty [m]	Počet expandovaných uzlu
Eukleidovská	0.004180908	26.67566	1091
Manhattanská	0.003479004	28.60426	988
Čebyševova	0.002227783	25.65294	701
Octile	0.004882813	30.15134	1181

Tabulka 6.3: Výsledky metrik z třetího testu

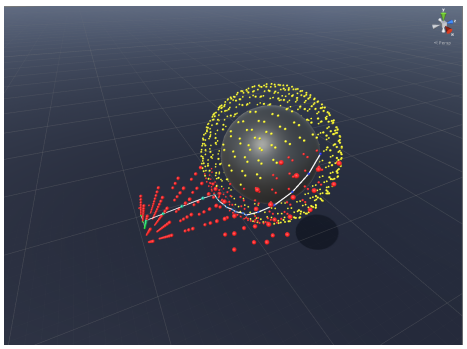
výsledek, jelikož nejrychlejší metrikou byla Eukleidovská a to hlavně proto, že expandovala nejméně uzlů. Mi se ovšem primárně zajímáme o délku nalezené cesty. Nejkratší cesta byla nalezena za použití Čebyševovy metriky. U tohoto testu a předchozího byli cílové uzly umístěny v mřížce. Proto u následující testu bude cílovým uzlem inspekční bod. Výsledné cesty najdeme na obrázku 6.3 a naměřené hodnoty v tabulce 6.4.



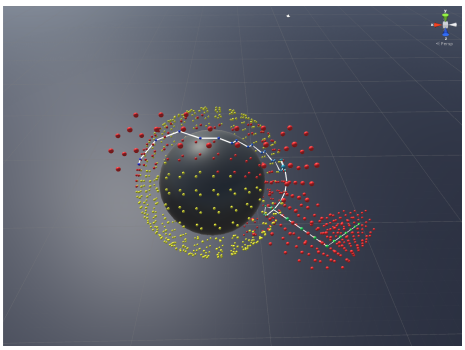
(a) : Eukleidovská metrika.



(b) : Manhattanská metrika



(c) : Čebyševova metrika



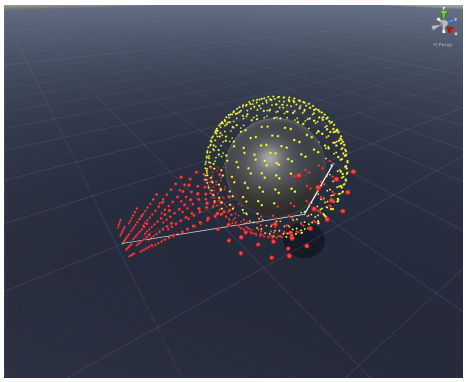
(d) : Octicle metrika

Obrázek 6.3: Cesty nalezené jednotlivými metrikami ve třetím testu.

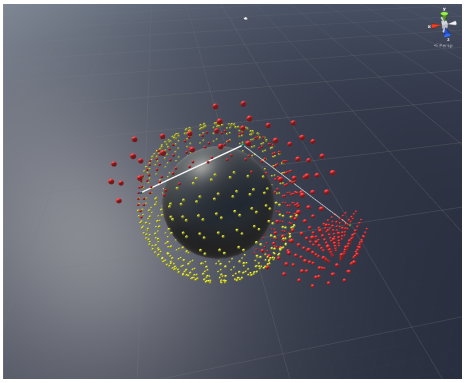
Z tabulky 6.4 vyčteme, že nejkratší cesta byla nalezena opět pomocí Čebyševovy metriky. Navíc při použití Čebyševovy metriky byla cesta nalezena nejrychleji a také expandovala nejméně uzlů. Z provedených testů plyne, že bychom měli při výběru heuristiky upřednostňovat Čebyševovu metriku, ovšem není nijak zaručeno, že Čebyševova metrika najde vždy nejkratší cestu.

6.2 Vyhlazování nalezených cest

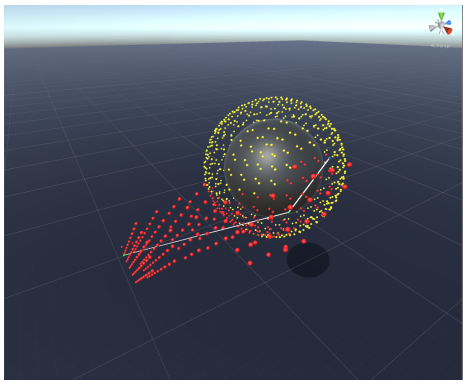
Na obrázcích 6.2 a 6.3 vidíme, že cesty nalezené algoritmem A* a příslušnou metrikou můžou být vyhlazeny. Vyhlazení cesty bude mít za následek její zkrácení a zjednodušení (bude se skládat z méně nodů).Vyhlazením cesty de-facto vytvoříme cestu novou. Takže musí být zaručeno, že na nové vyhlazené cestě nebudou žádné překážky. To dokážeme zajistit pomocí funkce *SphereCast* z knihovny Unity. Podívejme se jaký dopad bude mít vyhlazení cest z obrázku 6.3, jejich vyhlazené varianty jsou na obrázku 6.4. Nezapomeňme, že v sekci 6.1 jsme uvažovali UAV o velikosti bodu, musíme tedy tuto úvahu zachovat i při následném vyhlazování.



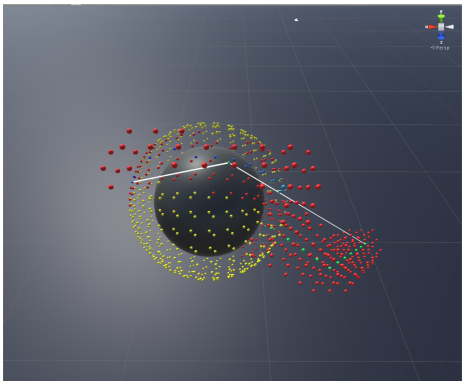
(a) : Eukleidovská metrika.



(b) : Manhattanská metrika



(c) : Čebyševova metrika



(d) : Octicle metrika

Obrázek 6.4: Vyhlazené cesty z obrázku 6.3.

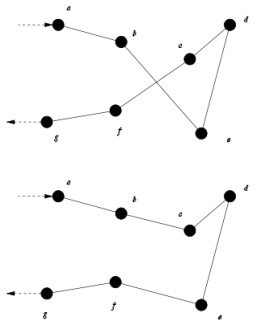
Z obrázku 6.4 vidíme, že došlo k zjednodušení cest. Porovnáme-li vzdálenosti z tabulek 6.2 a 6.4 shledáme, že nejkratší cesta z 6.2 byla vyhlazováním zkrácena přibližně o dva metry.

Jméno metriky	Čas [s]	Délka cesty [m]	Počet expandovaných uzlů
Eukleidovská	0.004180908	23.49276	1091
Manhattanská	0.003479004	25.43348	988
Čebyševova	0.002227783	23.54231	701
Octile	0.004882813	25.53794	1181

Tabulka 6.4: Výsledky metrik po vyhlazení cesty

6.3 Problém obchodního cestujícího

Problém obchodního cestujícího vyřešíme pomocí jednoduchého lokálního vyhledávacího algoritmu 2-opt. Základní myšlenka tohoto algoritmu je najít cesty, které se překřižují a přeuspořádat je tak aby se již nepřekřižovaly. Tato myšlenka je znázorněna na obrázku 6.5.



Obrázek 6.5: Znázornění základní myšlenky algoritmu 2-opt. Obrázek převzat z [1].

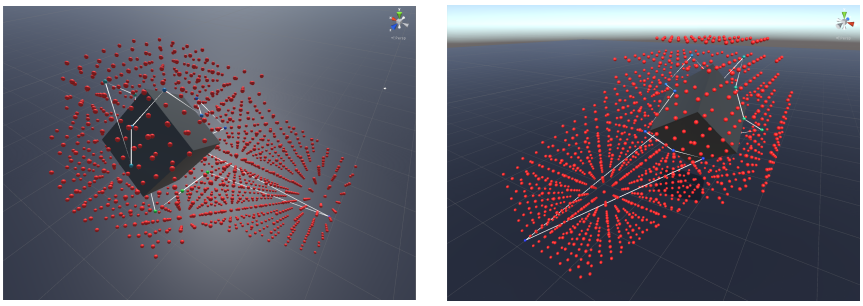
Pseudokód algoritmu 2-opt, jež byl převzat z 6.5 :

```
repeat until no improvement is made {
    start_again:
    best_distance = calculateTotalDistance(existing_route)
    for (i = 0; i < number of nodes eligible to be swapped - 1; i++) {
      for (k = i + 1; k < number of nodes eligible to be swapped; k++) {
          new_route = 2optSwap(existing_route, i, k)
          new_distance = calculateTotalDistance(new_route)
          if (new_distance < best_distance) {
          existing_route = new_route
          goto start_again
          }
      }
    }
}

2optSwap(route, i, k) {
  1. take route[1] to route[i-1] and add them in order to new_route
```

```
    2. take route[i] to route[k] and add them in reverse order
          to new_route
    3. take route[k+1] to end and add them in order to new_route
    return new_route;
}
```

Výše v pseudokódu vidíme, že při každé iteraci dochází k vytvoření dvou nových cest. To pro nás znamená, že se v každé iteraci bude dvakrát pouštět algoritmus A* aby tyto nové cesty vyhledal. Což má značný dopad na časovou náročnost algoritmu 2-opt.Algoritmus vyzkoušíme na malém počtu inspekčních bodů. Pro tento účel nám poslouží model krychle. Při tomto testu byla strana krychle osm metrů, inspekční body ve vzdálenosti dva metry, byla použita Čebyševova metrika, UAV bylo aproximováno jako koule o poloměru 1,25 m a výsledná trasa byla vyhlazena. Výsledek tohoto testu je zobrazen na obrázku 6.6.



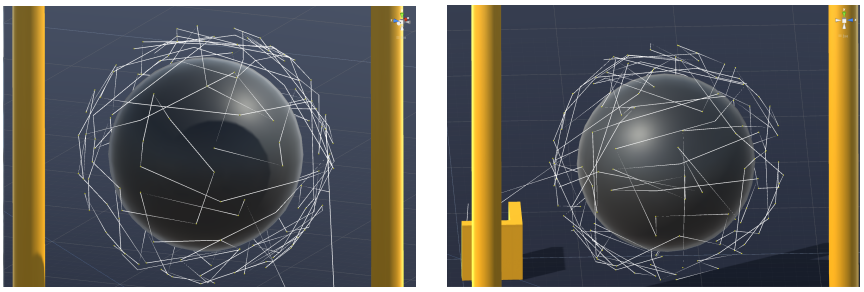
Obrázek 6.6: Naplánovaná trajektorie (bílá linka) pro bezpilotní prostředek.

Naplánovaná trajektorie na obrázku 6.6, je zvýrazněna bílou linkou. Posloupnost, kterou se prochází jednotlivé uzly na trajektorii poznáme podle jejich zbarvení. Počáteční uzel je zelený, následující uzly přecházejí pozvolně do modré barvy. Jelikož koncový i počáteční uzel je stejný, tak počáteční uzel je přebarven na modro. Původní náhodně vygenerovaná trajektorie měřila 182 metrů a díky algoritmu 2-opt a vyhlazování byla nalezena trajektorie o délce 132 metrů.

Kapitola 7

Simulace

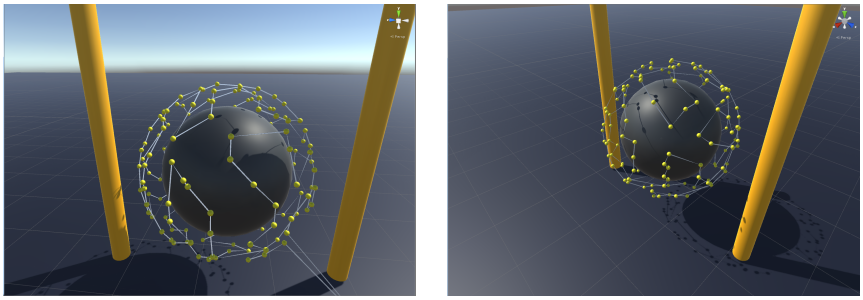
Vzhledem k časové náročnosti náročnosti algoritmu 2-opt, jsme se rozhodli provést simulaci letu UAV při inspekci koule. Pro tuto simulaci jsme zvolili následující hodnoty parametrů: velikost UAV jsme aproximovali jako kouli o poloměru 1.25 metru, úhel zorného pole kamery byl nastaven na 60∘, parametr *Far* kamery byl nastaven na 3 metry, vzdálenost inspekčních bodů od povrchu modelu byla nastavena na 2.5 metru, použili jsme metriku Octicle a výslednou trasu jsme samozřejmě nechali vyhladit. S tímto nastavením jsme zredukovali původních 768 inspekčních bodů na 153. Poprvé jsme pustili algoritmus 2-opt s časovým limitem 45 minut. Počáteční náhodně vygenerovaná trajektorie mela délku 1568 metrů, po 45 minutách byla trajektorie snížena na 946 metrů. Tato trajektorie je zobrazena na obrázku 7.1.



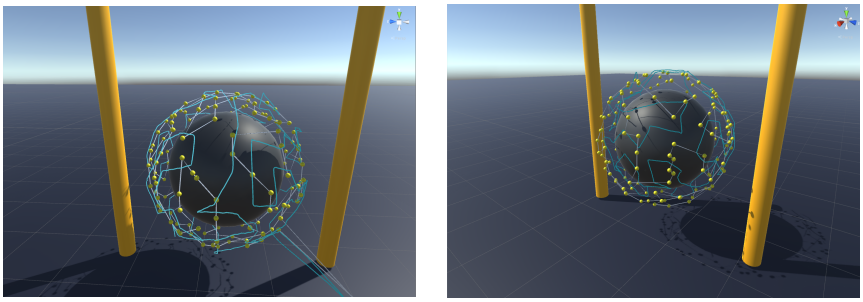
Obrázek 7.1: Trajektorie nalezená algoritmem 2-pot po 45 minutách.

Již na první pohled na 7.1 vidíme, že takováto trajektorie je nepoužitelná. Není to způsobeno žádnou chybou, jen to dokazuje časovou náročnost algoritmu 2-pot. Po druhé jsme pustili algoritmus 2-opt s časovým limitem 12 hodin. Bohužel i po této době byl algoritmus 2-opt zastaven časovým limitem. Výsledná trajektorie je na obrázku 7.2. Délka této trajektorie je 429 metrů, je tedy zhruba o polovičku kratší něž trajektorie nalezená po 45 minutách.

Vidíme, že u trajektorie z obrázku 7.2 došlo k drastickému zlepšení oproti trajektorii z obrázku 7.1. Trajektorii nalezenou po 12 hodinách použijeme dále. Polohy bodů, ze kterých se skládá trajektorie 7.2 uložíme do souboru abychom je mohly použít pro simulaci letu UAV v programu AMP Planner. Trajektorii získanou touto simulací jsme zpětně převedli do Unity. Na obrázku 7.3 vidíme porovnání trajektorii získanou simulací v programu AMP Planner (tyrkysová) a naplánované trajektorie (bílá).



Obrázek 7.2: Trajektorie nalezená algoritmem 2-pot po 12 hodinách.



Obrázek 7.3: Porovnání naplánované trajektorie (bílá) a odsimulované trajektorie (tyrkysová).

Z obrázku 7.3 vidíme, že zvolená bezpečnostní vzdálenost je dostačující, jelikož nedošlo ke kolizi UAV a objektu. Vidíme také, že vzdálenost mezi UAV a objektem se podařilo udržet. Dále můžeme vypozorovat, i určitou podobnost mezi odsimulovanou trajektorií a trajektorií naplánovanou. Připomeňme, že naše řešení uvažuje pouze snímání objektu z inspekčního bodu. Ovšem ve skutečnosti může snímání probíhat po celou dobu letu. To znamená, že naplánovaná trajektorie nemusí být zcela přesně proletěna UAV, aby byla provedena inspekce celého objektu. Na obrázku 7.3 je vidět, že odsimulovaná trajektorie vede okolo celého modelu.

Kapitola 8

Závěr

Cílem bakalářské práce bylo navrhnout a implementovat systém pro plánování trajektorií bezpilotního prostředku. Na začátku našeho řešení předpokládáme, že již máme mračno bodů. Toto mračno bodů bylo nasnímáno zařízením LIDAR a je tedy uloženo v souboru ve formátu .las. Pomocí programu LasUtility jsme převedli soubor z formátu .las do formátu .xyz. Tento formát již bylo možné otevřít programem MeshLab. V tomto programu jsme použili algoritmus BPA, který z mračna bodů dokáže zrekonstruovat povrch naskenovaného objektu. Tím jsme získali polygonální model, který jsme mohly nahrát do prostředí Unity 3D. V Unity 3D jsme začali s implementací systému pro plánování trajektorie. Začali jsme s výpočtem polohy inspekčním bodů, což jsou body, ze kterých se bude skládat finální trajektorie. Na tyto body jsme kladli dva požadavky. Prvním požadavkem bylo, že budou v dostatečné vzdálenosti od modelu, aby se předešlo kolizi UAV s objektem. Druhý požadavek byl, že tyto body musejí být rozmístěny tak, aby byla provedena inspekce celého modelu. To nás vedlo ke zjištění, že pokud provedeme inspekci každého trojúhelníku, ze kterých je tvořen model, tak bude provedena inspekce i celého modelu. Vypočítali jsme tedy inspekční bod pro každý trojúhelník, jako bod ležící na normále příslušného trojúhelníku ve vzdálenosti x od povrchu modelu. Toto řešení, ovšem přineslo velké množství inspekčních bodů, které bylo nutné je zredukovat. Redukci jsme provedli na základě tvrzení, že čím dále od povrchu modelu bude inspekční bod, tím vetší část modelu bude v zorném poli kamery. Využili jsme této skutečnosti a implementovali algoritmus, který zredukoval počet inspekčním bodů a zároveň zachoval, že bude provedena inspekce celého modelu. Zjistili jsme, že u komplexních tvarů modelu může dojít k tomu, že inspekční bod bude v blízkosti modelu, či dokonce uvnitř. Museli jsme tyto inspekční body smazat, abychom zabránili možné kolizi UAV a modelu. To mělo za následek, že nebudeme moci provést inspekci celého objektu. Následujícím krokem řešení, bylo implementovat automatické generování stavů stavového prostoru. Za tímto účelem jsme rozdělily vymezený trojrozměrný prostor na jednotlivé buňky. Dále jsme stanovily, že uprostřed buňky vytvoříme uzel, jestliže mezi uzlem a nejbližším objektem bude větší, než vzdálenost mezi inspekčním bodem a modelem. Toto nám zajistilo, že u výsledné trajektorie bude dodržena bezpečná vzdálenost od modelu i jiných překážek. Dále jsme také na pozici každého inspekčního bodu

vytvořili uzel. Tyto uzly jsou stavy stavového prostoru. Aby bylo možné ve stavovém prostoru vyhledávat, definovali jsme sousední uzly. Zjistili jsme, že u komplexních modelů dochází ke generování nežádoucích uzlů uvnitř modelu. Pro tento problém jsme našili dvě různá řešení. První řešení spočívá v umístění překážky (např. koule) do místa kde se nechtěné uzly generují. Toto řešení je jednoduché a účinné. Druhé řešení spočívá v automatickém nalezení takovýchto uzlů a jejich vymazání. Toto řešení je funkční nicméně není stoprocentní. Dalším krokem bylo implementovat algoritmus A*, který bude vyhledávat ve stavovém prostoru, který jsme automaticky vygenerovali. Abychom docílili zjednodušení (cesta se bude skládat z menšího počtu uzlů) a zkrácení cest implementovali jsme vyhlazování cest. Poslední částí řešení bylo vyřešit problém obchodního cestujícího pro inspekční body. Za tímto účelem jsme implementovali algoritmus 2-opt. Na závěr této práce jsme provedli porovnání naplánované trajektorie a trajektorie, po které letělo UAV v simulačním programu AMP Planner. Čímž jsme zjistili, že UAV obletělo daný model aniž by došlo ke kolizi. Systém implementovaný v rámci této bakalářské práce umožňuje naplánovat trajektorii pro inspekci jednoduchých objektů (např. koule). Ovšem u komplexních tvarů není zaručena inspekce celého objektu.

Literatura

[1] 2-opt - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/2-opt. (Accessed on 05/26/2016).

[2] A* – wikipedie. https://cs.wikipedia.org/wiki/A*. (Accessed on 05/26/2016).

[3] Constructive solid geometry - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Constructive_solid_geometry. (Accessed on 05/23/2016).

[4] Heuristics. http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html. (Accessed on 05/26/2016).

[5] Las_1_3_r11.pdf. http://www.asprs.org/wp-content/uploads/2010/12/LAS_1_3_r11.pdf. (Accessed on 05/21/2016).

[6] Nurbs – wikipedie. https://cs.wikipedia.org/wiki/NURBS. (Accessed on 05/21/2016).

[7] Objektiv ip kamery | securityguide. https://www.securityguide.cz/security/viewArticle/objektiv-IP-kamery. (Accessed on 05/24/2016).

[8] Point cloud - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Point_cloud. (Accessed on 05/21/2016).

[9] Polygonal modeling - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Polygonal_modeling. (Accessed on 05/21/2016).

[10] Počítačová 3d grafika – wikipedie. https://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8Dov%C3%A1_3D_grafika. (Accessed on 05/21/2016).

[11] The stanford 3d scanning repository. http://graphics.stanford.edu/data/3Dscanrep/. (Accessed on 05/23/2016).

[12] Unity - game engine. https://unity3d.com/. (Accessed on 05/24/2016).

[13] Unity - manual: Colliders. http://docs.unity3d.com/Manual/CollidersOverview.html. (Accessed on 05/27/2016).

[14] Unity - scripting api: Physics.overlapsphere. http://docs.unity3d.com/ScriptReference/Physics.OverlapSphere.html. (Accessed on 05/26/2016).

[15] What is lidar? http://oceanservice.noaa.gov/facts/lidar.html. (Accessed on 05/21/2016).

[16] Základy reprezentace trojrozměrného prostoru v počítači – wikisofia. https://wikisofia.cz/index.php/Z%C3%A1klady_reprezentace_trojrozm%C4%9Brn%C3%A9ho_prostoru_v_po%C4%8D%C3%ADta%C4%8Di. (Accessed on 05/21/2016).

[17] animatedheaven. Old farm house - 3d model - .obj, .mb, .fbx. http://tf3dm.com/3d-model/old-farm-house-91130.html. (Accessed on 05/22/2016).

[18] Holly Rushmeier Claudio Silva Gabriel Taubin Fausto Bernardini, Joshua Mittleman. bpa.dvi. http://www.research.ibm.com/vistechnology/pdf/bpa_tvcg.pdf. (Accessed on 05/22/2016).