F3

Fakulta elektrotechnická
Katedra kybernetiky

Bakalářská práce

Integrace simulátoru dronů v Unreal Engine 5

Jan Hrnčíř

24. května 2024

Vedoucí práce: Ing. Robert Pěnička, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hrnčíř** Jméno: **Jan** Osobní číslo: **483140**

Fakulta/ústav: **Fakulta elektrotechnická**

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

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Integrace simulátoru dronů v Unreal Enginu 5

Název bakalářské práce anglicky:

Drone simulator integration in Unreal Engine 5

Pokyny pro vypracování:

1) Seznamte se s Unreal Engine 5 (UE5), se systémem skupiny Multi-Robot Systems (MRS) pro simulaci autonomní dronů (UAV) a dalšími nástroji používanými pro simulaci UAV.
2) Navrhněte propojení UE5 s MRS UAV simulací pro získávání senzorických dat z navržených prostředí v UE5.
3) Vhodně zvolenými metodami procedurálního generování vytvořte minimálně dvě různá prostředí pro účely testování autonomního letu v prostředí s překážkami. Student sám vybere vhodná prostředí jako například les či jeskyně.
4) Kriticky zhodnoťte dosažené výsledky s ohledem na budoucí využití. Porovnejte rychlost získávání sensorických dat v navržených prostředích pro jedno a více UAV. Srovnejte vlastnosti implementovaných metod a generované výstupy z hlediska vizuální kvality a složitosti generovaného modelu.

Seznam doporučené literatury:

[1] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, M. Saska, “The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles,” J Intell Robot Syst 102 (26), 2021.
[2] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, D. Scaramuzza, “Flightmare: A flexible quadrotor simulator,” In Conference on Robot Learning 2021, pp. 1147-1157.
[3] Shah, Shital, et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles." Field and Service Robotics: Results of the 11th International Conference, 2018.
[4] W.Lorensen, H.Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” Computer Graphics, 21 (4): 163-169, 1987.
[5] Smelik, Ruben M., et al. "A survey on procedural modelling for virtual worlds." Computer Graphics Forum, 33 (6), 2014.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Robert Pěnička, Ph.D. Multirobotické systémy FEL

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

Datum zadání bakalářské práce: **16.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

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

Poděkování / Prohlášení

Chtěl bych poděkovat celé své rodině za nekonečnou podporu při studiích nejen finanční a za to, že mohu dělat, co mě baví.

V neposlední řadě děkuji Ing. Robertu Pěničkovi., Ph.D. za vedení a cenné rady při vypravování mé bakalářské práce.

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

V Praze dne 24. 5. 2024

.....

Abstrakt / Abstract

Bakalářská práce se zabývá spojením plně autonomního simulačního systému dronů skupiny Multirobotických systémů (MRS) s herním enginem Unreal Engine 5 (UE5) a navržením virtuálních prostředí vhodných pro testování algoritmů autonomního letu dronů.

Virtuální prostředí byla vytvořena procedurálním generováním a zahrnují nekonečný les a uzavřené prostředí jeskyně. Pro tvorbu terénu lesa byl použit generátor šumu Perlin noise a procedurální generace obsahu v UE5 (Procedural Content Generation) pro umístění fotogrammetrických modelů stromů. Jeskyně byla vytvořena pomocí algoritmu Perlin worms a techniky Marching cubes pro vytvoření polygonálních povrchů jeskyně.

Na závěr práce byly provedeny testy zaměřené na ověření výkonnosti a použitelnosti navrženého spojení systémů. Výsledky ukazují, že navržený systém je schopen realisticky simulovat a testovat algoritmy autonomního letu dronů v různých prostředích.

Klíčová slova: Plně autonomní simulace dronů, Unreal Engine 5, UAV, Procedurální generování, Perlin Noise, Perlin Worms, Marching Cubes

The bachelor thesis deals with the integration of a fully autonomous drone simulation system of the Multi–robotic Systems Group with the Unreal Engine 5 (UE5) game engine and the design of virtual environments suitable for testing algorithms for autonomous drone flight.

The virtual environments were created using procedural generation and include an infinite forest and a closed cave environment. The Perlin noise generator was used to create the forest terrain and Procedural Content Generation in UE5 was used to place photogrammetric models of trees. The cave was created using the Perlin worms algorithm and the Marching cubes technique to create polygonal surfaces.

At the end of the work, we performed test to verify the performance and usability of the proposed system coupling. The results show that the proposed system is able to realistically simulate and test autonomous drone flight algorithms in different environments.

Keywords: Drone simulation, Unreal Engine 5, Fully Autonomous Drones, UAV, Procedural Content Generation, Perlin Noise, Perlin Worms, Marching Cubes

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Použití robotických simulátorů hraje zásadní roli nejen pří vývoji autonomních dronů neboli UAV (Unmanned aerial vehicle), ale i v robotice obecně. Testování a ladění nejrůznějších parametrů navržených metod pomocí skutečných robotů by bylo příliš nákladné a nepraktické z finančního i časového hlediska. Velkou výhodou je, že vzniklé chyby při simulaci nejsou fatální ve srovnání se selháním skutečných robotů.

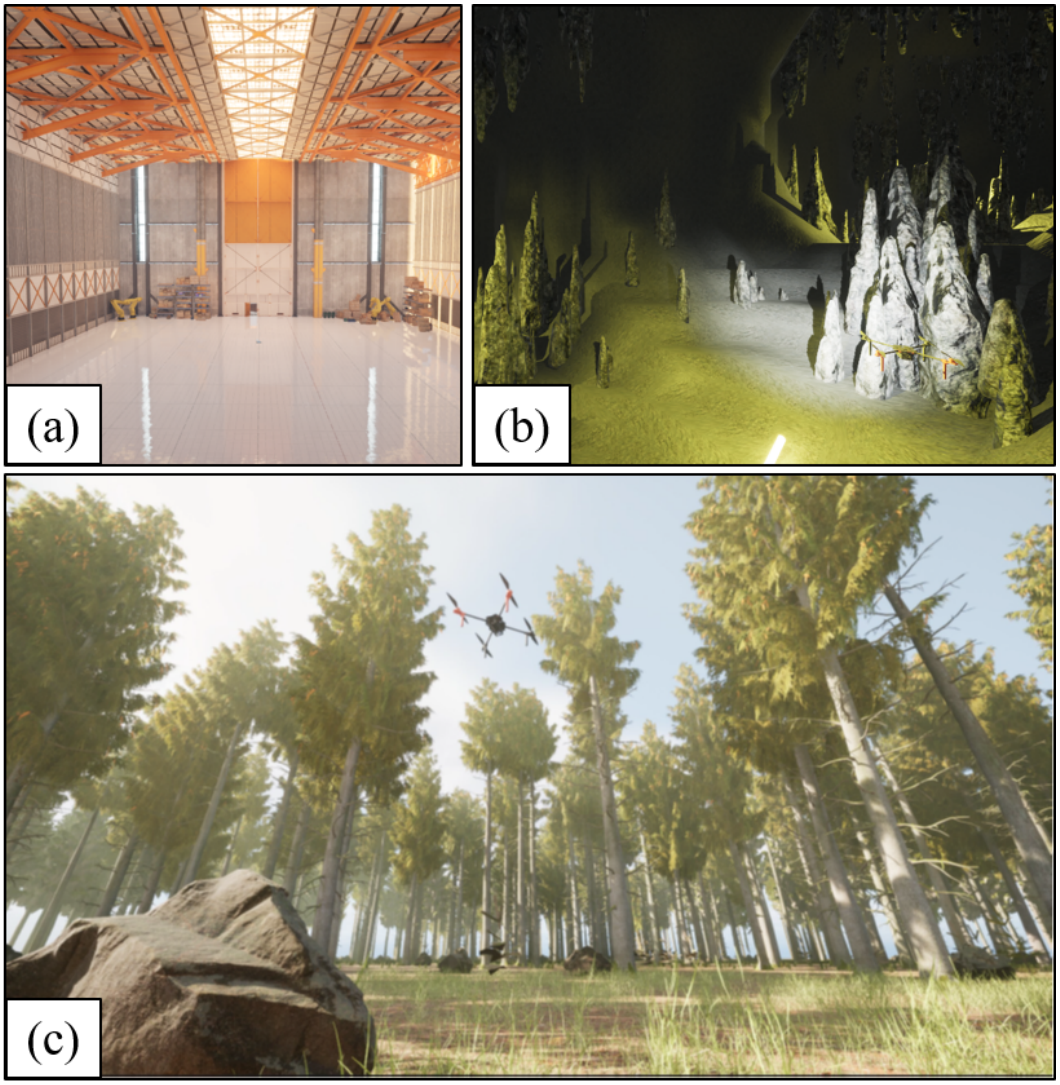
Nedávné pokroky v metodách učení umělé inteligence v robotice jako například autonomní závodní drony [1] dosahující úrovně světových šampionů nebo naučení pohybů kráčejících robotů [2] byl umožněn díky věrným fyzikálním simulacím a shromažďovaní dat ze simulovaného virtuálního prostředí.

I přesto neexistuje žádný simulátor zahrnující plně autonomní dron schopný autonomního letu založeném na získaných datech ze senzorů v cela neznámém nepřehledném prostředí. Většina simulátorů [3–5] zmíněních ve druhé kapitole se často používá ve spojení s přístupy založenými na učení jako je například Reinforcement Learning, kde se vyhodnocuje mnoho tréninkových pokusů (úspěšných či neúspěšných) k naučení správné trajektorie pohybu dronu. Tyto simulátory na rozdíl od tradičních robotických simulátorů, jako je například Gazebo [6], nejsou určeny pro tradiční plnohodnotné robotické simulace v reálném čase, kde software pro navigaci dronů komunikuje přímo s fyzikálně simulovaným robotem a prostředím v reálném čase, zatímco přistupuje k široké škále simulovaných senzorů.

Současně roste potřeba poskytovat vysoce věrné a fotorealistické simulace. S rostoucím využitím přístupů založených na počítačovém vidění (např. detekce překážek) je potřeba mít minimální rozdíl mezi virtuálním a reálným prostředím. Fotorealičnosti lze dosáhnout pomocí herních enginů (např. Unity [7] a Unreal Engine 5 [8]), které využívají pokroku ve vývoji počítačové grafiky a grafických karet, a umožňují pokročilé funkce vykreslování, jako je např. ray tracing.

Dostupné simulátory s vysokou mírou fotorealizmu postrádají vysoký stupeň autonomních navigačních funkcí, jako je generování trajektorií, plánování cest, mapování a lokalizace. Vzhledem k chybějícím funkcím se uživatelé simulátorů často zaměřují na vývoj těchto funkcí, spíše než na řešení úloh, které jsou skutečným předmětem jejich výzkumu.

Z toho důvodu jsou cíle bakalářské práce spojení plně autonomního simulátoru dronů skupiny Multirobotických systémů (MRS) [9–11] s herním enginem Unreal Engine 5 a navržení virtuálních prostředí s překážkami, které budou sloužit pro testování algoritmů autonomního letu. Hlavními vytvořenými prostředími jsou les a jeskyně (viz Obrázek 1.1). Obě prostředí jsou plně procedurálně generovaná, což umožňuje jednoduchou přípravu scénářů pro testování algoritmů autonomie dronů. Dále je vytvořeno jedno statické prostředí skladu, které díky použitým modelům dosahuje velké míry fotorealičnosti. V závěru práce jsou provedeny testy s cílem ověření výkonnosti a použitelnosti navrženého spojení systémů ve vytvořených prostředích i v porovnání s konkurenčním simulátorem.



Obrázek 1.1. Vytvořená prostředí v UE5: (a) Sklad (b) Jeskyně (c) Les

Kapitola 2
Současné robotické simulátory

V této kapitole představíme několik existujících open-source simulátorů, které jsou využívány při výzkumu a vývoji robotiky a strojového učení. Zaměříme se zejména na simulátory dronů a ukážeme jejich klíčové vlastnosti, výhody a případná omezení.

Autonomní systémy dronů se často spoléhají na informace získané ze senzorů, jako jsou LiDAR, kamery, inerciální měřicí jednotky (IMU) a globální polohovací systém (GPS). Tyto senzory jsou nezbytné pro simultánní lokalizaci a mapování (SLAM) a vizuální navigaci, které umožňují autonomní let. Bez těchto senzorů by dron neměl způsob, jak se v prostoru navigovat. Proto by měl simulátor dronů obsahovat širokou škálu simulovaných senzorů, aby mohl realisticky napodobit podmínky letu a umožnit komplexní testování algoritmů autonomie.

Již zmíněné *Gazebo* [6] je oblíbený open-source simulátor pro různé druhy robotů. Renderovací engine je založen na OpenGL, což je API pro grafický hardware umožňující vykreslování 2D a 3D grafiky. Dynamika dronu je řešena široce využívaným open-source fyzikálním enginem, Open Dynamics Engine. Lze simulovat RGBD kameru, IMU, LiDAR a GPS. Použití Gazeba společně s Robot Operating System (ROS) je běžnou praxí při vývoji robotických aplikací. Hlavní nevýhodou Gazeba jsou omezené vykreslovací schopnosti ve srovnání dále uvedenými simulátory.

Flightmare[3] je open-source flexibilní modulární simulátor dnonů. Jeho architektura je založená na oddělení renderovacího a fyzikálního enginu, což přináší výhodu uživateli rozhoudnout se, zda potřebuje oba enginy. Autoři ukazují možnost extrémně rychlé simulace: 230Hz při zapnutém renderovacím enginu a až 200 000 Hz při samostatné fyzikální simulaci. Renderovací engine je vytvořen v popularním 3D herním enginu Unity. Flightmare nabízí širokou škálu prostředí od jednoduchého skladiště po komplexní scénu lesa. Nové prostředí jsou snadno dostupná z Unity Store buď placené variantě nebo některé assety lze nálezt i volně dostupné. Samotné rozšíření prostředí nebo jejich výměna je pro uživatele přímočará a vyžaduje jen minimální znalosti enginu Unity. V simulátoru jsou dostupné tři varianty dynamiky, implementace senzorů IMU a RGBD kamery s možností nastavení velikosti záběru ohniska a dokonce simulace efektu *motion blur*. Ačkoli není implementován LIDAR, je umožno přes aplikační programové rozhraní (API) extrahovat mračno bodů daného prostředí. Simulátor je vhodný pro aplikace hlubokého učení a reinforcement learningu [12].

AirSim [4] je open-source simulátor vyvíjený společností Microsoft. Je postavený na herním enginu Unreal Engine 4, který autoři využívají k docílení realistické grafiky a výpočtu kolizí, a vlatní realtime fyzikální simulací s frekvencí 1000Hz. Simulátor podporuje širokou škálu senzorů, včetně IMU, magnetometru, GPS, barometru a RGBD kamer. Airsim integruje hardware ovladač PX4 umožnující HITL (hardware-in-the--loop) simulace. Airsim je využitelný ve výzkumu a nasazení strojového učení a zpracování obrazu pro autonomní vozidla i drony. Dále nabízí možnost dynamického nastavení efektu počasí. [12].

FlightGoggles [5] je open-source simulátor postavený na platformě Unity. Jeho hlavním konceptem je využití fotogrammetrie - přístupu k vytváření 3D objektů a prostředí z fotografií. Tímto způsobem lze naskenovat reálná prostředí, zpracovat je do virtuálního prostředí a použít je k simulačním účelům. Podobně jako Flightmare má modulární architekturu, což umožňuje běh vykreslování a detekce kolizí samostatně. FlightGoggles obsahuje RGBD kameru, IMU a LiDAR.

UAV ToolBox MATLAB-Simulink [13]. MATLAB se Simulinkem vytváří univerzální simulační prostředí se sadou toolboxů a algoritmů. UAV Toolbox poskytuje nástroje pro návrh, simulaci a testování algoritmů pro autonomní drony. Můžeme navrhovat algoritmy pro autonomní let a ověřovat navržené postupy v HITL simulaci. UAV Toolbox stejně jako Airsim podporuje hardware ovladač PX4. Lze simulovat výstupy kamer, LiDAR, IMU a GPS senzorů ve fotorealistickém 3D prostředí založeném na Unreal Enginu nebo ve zjednodušeném simulačním prostředí přímo v MATLABu, který je složený pouze z jednoduchých polygonálních objektů. Simulátor je pevně integrován do ekosystému MATLAB a neposkytuje univerzální API.

Kapitola 3
Integrace UEds s MRS UAV simalací

Z předchozí kapitoly po seznámení s vlastnostmi existujících simulátorů se ukazuje být výhodné a praktické oddělení renderovacího (UEds)[14] a fyzikálního enginu (MRS UAV simalace)[15] pro různé typy simulací. Například simulace, kdy simulovaný čas je daleko rychlejší než reálný čas, jsou využívány pro sběr dat ke strojovému učení, kde je potřeba co nejrychleji sbírat senzorická data z dronů, která jsou na sobě nezávislá. Naopak, při simulacích v reálném čase s více drony je klíčové zajistit plynulý a realistický běh simulace. Oddělení dynamiky je také výhodné pro navrh a testování metod automatického řízení. Tento přístup umožňuje lépe se přizpůsobit specifickým potřebám konkrétního projektu.

S touto myšlenkou bude navrženo spojení těchto dvou systémů. Oba systémy vznikají ve skupině Multirobotické systémy (MRS). Na začátku kapitoly nejdříve představíme oba systémy a na konec bude rozebráno navržené řešení spojení.

3.1 Unreal Engine simulátor dronů

Simulátor dronů v herním enginu Unreal Enginu 5 (UEds) vznikl v důsledku požadavku skupiny MRS na vlastní renderovací systém. Skupina nechce být závislá na systémech třetích stran, jejíž podpora není často dostatečná s porovnání obrovské komunity Unreal Engine 5.

Tento engine je v herním průmyslu standardem pro pohlcující a fotorealistická prostředí. Unreal Engine 5 obsahuje pokročilé algoritmy pro renderování počítačové grafiky, fyzikální simulace a přesné detekce kolizí objektů.

Hlavním požadavkem na vytvořený simulátor bylo využití fotorealistické grafiky, detekce kolizí a simulace senzorů jako je LIDAR, kamera a IMU jednotka. UEds umožňuje simulovat paralelní běh několika dronů současně, což je výhodné pro trénování a vyhodnocení algoritmů reinforcement learning, protože simulace může běžet daleko rychleji než v reálném čase.

Architektura simulátoru UEds je rozdělaná do tří částí: herní režim, prostředí a samotné drony.

Herní režim

- Hlavní komponenta v Unreal Enginu, která řídí celou logiku simulátoru, resp. hry.
- Udržuje si referenci na všechny drony ve scéně a manipuluje s nimi.
- Umožňuje vytváření a odstraňování dronů.
- Případně může být využit ke změně prostředí, manipulaci se světlem nebo přidávání/odebírání jiných objektů (např. překážek).

Prostředí

- Obsahuje abstrakci světa, mapy, osvětlení a fyziky.
- Drží reference na objekty v prostředí (např. zdi, stromy, terén, drony, světlo) a udržuje si hierarchii transformací mezi nimi.
- Umožňuje objektům mezi sebou interagovat, jako například řešení kolizí nebo vypočítání stínů.

Dron

- Reprezentace dronu z reálného světa v Unreal Engine 5.
- Obsahuje grafický model (mesh) dronu (mesh), jeho rám, vrtule a motory.
- K modelu jsou připojeny senzory LIDAR a kamer. Jejich poloha je vzhledem ke dronu plně konfigurovatelná.
- Dron se může pohybovat ve všech šesti stupních volnosti. Lze mu nastavit polohu i rotaci.
- Při změně polohy jsou detekovány kolize s prostředím. (V případě kolize při přesunu z jedné polohy do druhé je nastavena nová poloha dronu v místě kolize.)

Komunikace se UEds simulátorem je řešena pomocí TCP/IP protokolu. Herní režim a objekty dronů si drží reference na vlastní TCP server, což umožňuje větší komunikační propustnost ve srovnání s jedním společným serverem. To odpovídá reálnému světu, kde má každý dron pro ovládání svůj vlastní ovladač. Nejdříve je spuštěn server herního režimu, který čeká na požadavky. Simulátor nepřidává žádné drony při spuštění, a tak je ze začátku server herního režimu jediným běžícím serverem. Po požadavku na vytvoření dronu herní režim vytvoří instanci dronu, spustí server dronu a udržuje referenci na dron pro případné pozdější odstranění.

Pro účely komunikace byli vytvořeny dvě klientské aplikace (*UEds-game-mode--controller* a *UEds-drone-controller*) poskytující API rozhraní (Application Programming Interface) pro server herního režimu a servery dronů. Oba klienti jsou směrovány k příslušným serverům a využívají serializace zpráv do binárního formátu, jehož výhodou je rychlejší přenos menšího objemu dat než u klasických formátů jako např. XML nebo JSON.

Simulátor lze integrovat do jakéhokoliv prostředí v Unreal Engine. Pro zprovoznění UEds v novém prostředí je nutné provést dva kroky. Nejprve je zapotřebí importovat kódy modulu samotného simulátoru UEds a modulu pro serializaci zpráv. Nakonec stačí jen nastavit výchozí herní režim na režim poskytovaný simulátorem UEds.

Zkrácený seznam metod API rozhraní nutných pro potřeby spojení s MRS UAV simulací je v Tabulce 3.1 a Tabulce 3.2.

Metoda	Popis metody
Ping	test funkčnosti
GetDrones	vrátí seznam všech dronů
SpawnDrones	vytvoří nový dron
RemoveDrones	odstraní dron

Tabulka 3.1. Seznam metod API rozhraní klienta herního režimu

Metoda	Popis metody
Ping	test funkčnosti
SetLocationAndRotation	nastaví novou polohu dronu
GetLidarData	vrátí získaná data z LIDARu (orientaci a délku paprsků)
GetCameraData	vrátí získaná data obrázky z kameru v JPG binárním formátu

Tabulka 3.2. Seznam metod API rozhraní klienta dronu

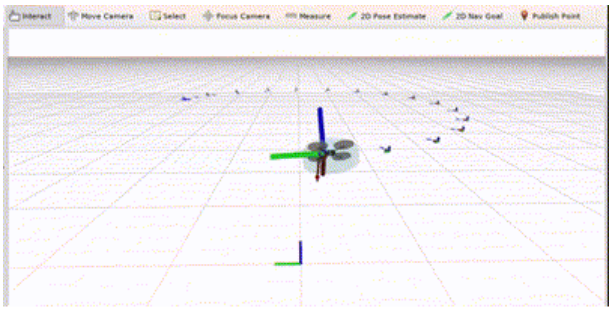
3.2 MRS UAV simulace

Skupina MRS se zaměřuje na autonomní multi-rotorové drony neboli UAV (unmanned aerial vehicle), pro které vyvíjí systémy automatického řízení, stavových pozorovatelů a simulací. MRS UAV systém je navržený tak, aby umožňoval bezpečné ověření navržených postupů plánování trajektorií, mapování prostředí a lokalizaci, počítačového vidění, a to vše před samotným provedením experimentů v reálném prostředí, kde vznik případné chyby může být při nejmenším hodně drahý.

MRS UAV systém je vyvíjený ve frameworku Robot Operetion Sytem (ROS)[16] ve formě jednotlivých systémových balíků (packages). ROS je prostředník mezi operačním systémem a jednotlivými spuštěnými procesy v ROSu, tzv. uzly (Nodes). Uzly mezi sebou komunikují prostřednictvím zpráv v definovaných informačních kanálech (Topic) a díky ROSu je zajištěno, že probíhá komunikace mezi všemi běžícími procesy asynchronně. Například můžeme vytvořit ovladač senzoru jako uzel, který bude odesílat data ze senzoru do příslušného kanálu. Zprávy z tohoto kanálu můžou být dále použity dalšími uzly pro potřeby logování, filtraci dat, až po zpracování vyššími systémy např. pro plánování a navigaci.

V současné době skupina pro simulace využívá Gazebo simulátor s nedostatečným vykreslováním prostředí a vlastní MRS Simulátor s jednoduchou vizualizací v Rviz [17] (viz Obrázek 3.1). Rviz je jednoduchý ROS vizualizační nástroj pro zobrazení dat z ROSu, jako jsou data ze senzorů a souřadnicové systémy robotů a podobně.

Právě rozšířením MRS Simulátoru bude vytvořeno napojení na renderovací simulátor UEds.



Obrázek 3.1. Rviz vizualizace fyzikální MRS Simulace letu po kruhové trajektorii [15]

MRS Simulátor se nachází v balíčku `mrs_multirotor_simulator`[1] a má tyto vlastnosti:

- Implementuje pouze v hlavičkových souborech C++ dynamiku dronu s řídicími zpětnovazebními regulátory. Tuto knihovnu lze použít v jakémkoli jiném projektu.

[1] `https://github.com/ctu-mrs/mrs_multirotor_simulator`

- Kaskáda zpětnovazebních regulátorů umožňuje řízení dronu od individuálních motorů až po požadovanou pozici a natočení vůči ose z, tzv. heading.
- Obsahuje ROS wrapper knihovny řešící dynamiku jednoho dronu.
- Celý simulátor je vytvořený jako minimalistický ROS uzel simulující vícero dronů s možností řešení vzájemných kolizí a kolizí se zemí (rovina v počátku).

Knihovna dynamiky řeší dynamický model dronu daný soustavu pohybových diferenciálních rovnic (1), (2) a (3).

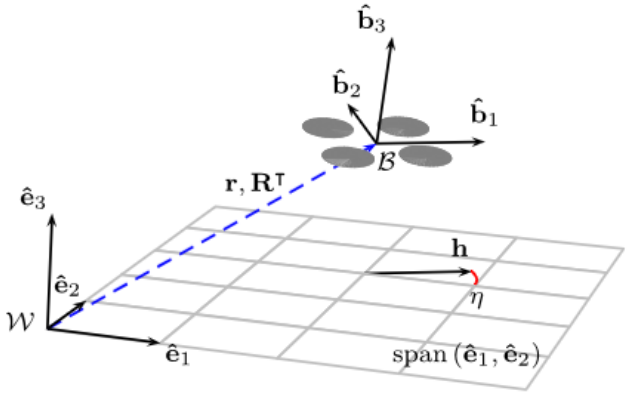
$$m\,\ddot{\boldsymbol{r}} = f\,\boldsymbol{R}\,\mathbf{e_3} - m\,g\,\mathbf{e_3} \tag{1}$$

$$\boldsymbol{J}\dot{\boldsymbol{\omega}} = \boldsymbol{\tau} - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} \tag{2}$$

$$\dot{\boldsymbol{R}} = \boldsymbol{R}\,\Omega \tag{3}$$

Kde proměnné značí: m hmotnost, $\boldsymbol{r} \in \mathbb{R}^3$ vektor pozice středu hmotnosti dronu v souřadnicovém systému světa, $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ rotační matici transformace ze souřadnicového systému dronu do souřadnicového systému světa (viz Obrázek 3.2), $\boldsymbol{\omega}$ vektor úhlových rychlostí, $\boldsymbol{J}$ matici momentu setrvačnosti, Ω tenzor úhlové rychlosti splňující podmínku $\Omega\boldsymbol{v} = \boldsymbol{\omega} \times \boldsymbol{v}, \forall \boldsymbol{v} \in \mathbb{R}^3$, f síla vykonaná motory, neboli tah, $\boldsymbol{\tau}$ moment síly působící na dron, a g konstantu gravitačního zrychlení.

Numerickou integrací obyčejných diferenciálních rovnic získáme nový stav dronu, který je určený vektorem $\boldsymbol{r}$ a maticí $\boldsymbol{R}$, v každém kroku simulace.



Obrázek 3.2. Vztah mezi transformacemi souřadnicových systémů dronu $\mathcal{B}$ a světa $\mathcal{W}$ [15]

3.3 Spojení MRS Simulátoru a UEds

Myšlenkou spojení je umožnění MRS Simulátoru využívat virtuální prostředí v UEds. K dosažení tohoto cíle je nezbytné zajistit efektivní komunikaci mezi oběma systémy. Hlavní a klíčovou vlastností je získání dat ze senzorů (LiDAR, kamera) pro aktuální stav pozice a orientace dronu. Komunikace je realizována pomocí API UEds, které je popsáno v Tabulce 3.1. Navrhované spojení musí vyřešit problém rozdílného používání pravotočivého (ROS) a levotočivého (Unreal Engine) souřadnicového systému. Dále je nezbytné vytvořit transformaci mezi MRS simulátorem a UEds, tedy umístit světový souřadnicový systém MRS simulace do UEds. V neposlední řadě je nutné definovat strukturu zpráv v ROS pro přenos dat ze senzorů.

Spojení je realizováno ve třídě `DroneControllerRos`, která obaluje API rozhraní klienta dronu v ROSu a vytváří tak prostředníka mezi oběma systémy. Třída si udržuje referenci na objekt dynamické simulace jednoho dronu, ze kterého získává aktuální stav dronu, a informuje tento objekt, když nastane kolize s prostředím UEds. Dynamická simulace následně simuluje pád dronu. Ve třídě jsou definovány metody zprostředkující komunikaci mezi MRS Simulátorem a UEds:

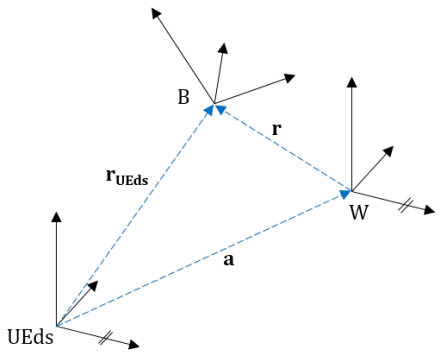
- nastavení pozice a orientace dronu v UEds,
- získání dat ze senzorů z UEds,
- nekonečná smyčka opakující předchozí metody (ROS Timer).

3.3.1 Transformace pozice a orientace dronu

Protože počátek souřadnic v Unreal Enginu se může nacházet kdekoliv ve vytvořeném virtuálním prostředí, které může být rozsáhlé, je nejprve nutné definovat umístění souřadnicového systému MRS Simulátoru v UEds. Instance objektu dronu se v UEds vytvoří na předem definovaných souřadnicích. V tomto bodě bude námi uvažovaný počátek světových souřadnic MRS simulátoru (viz Obrázek 3.2).

Nyní díky znalosti transformace mezi souřadnicovými systémy UEds a MRS Simulátoru můžeme vyjádřit pozici dronu získanou z dynamické simulace v souřadném sytému UEds. Jelikož se jedná pouze o translaci, lze transformaci vyjádřit vektorovým součtem (viz Rovnice (4)), kde $\boldsymbol{a}$ je rádiusvektor počátku MRS Simulace v souřadnicovém systému UEds, $\boldsymbol{r}$ je rádiusvektor počátku dronu v souřadnicovém systému MRS Simulace a $\boldsymbol{r}_{UEds}$ je rádiusvektor počátku dronu v souřadnicovém systému UEds.

$$\boldsymbol{r}_{UEds} = \boldsymbol{a} + \boldsymbol{r} \tag{4}$$



Obrázek 3.3. Transformace mezi souřadnicovými systémy

K nastavení orientace dronu v UEds potřebujeme znát úhly roll($\mathcal{R}$), pitch($\mathcal{P}$) a yaw($\mathcal{Y}$). Tyto úhly popisují neznámé elementární rotace okolo os x, y a z. Rotace o daný úhel kolem dané osy je vyjádřena rotačními maticemi $\boldsymbol{R}_x$, $\boldsymbol{R}_y$ a $\boldsymbol{R}_z$ (viz Rovnice (5),(6) a (7)).

$$\boldsymbol{R}_z(\mathcal{Y}) = \begin{pmatrix} \cos\mathcal{Y} & -\sin\mathcal{Y} & 0 \\ \sin\mathcal{Y} & \cos\mathcal{Y} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$\boldsymbol{R}_y(\mathcal{P}) = \begin{pmatrix} \cos\mathcal{P} & 0 & \sin\mathcal{P} \\ 0 & 1 & 0 \\ -\sin\mathcal{P} & 0 & \cos\mathcal{P} \end{pmatrix} \tag{6}$$

$$\boldsymbol{R}_x(\mathcal{R}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\mathcal{R} & -\sin\mathcal{R} \\ 0 & \sin\mathcal{R} & \cos\mathcal{R} \end{pmatrix} \tag{7}$$

Násobením těchto matic získáme potřebné vztahy pro neznámé úhly ze znalosti rotační matice $\boldsymbol{R}$ (viz Rovnice (8)) z dynamické simulace.

$$\boldsymbol{R} = \boldsymbol{R}_z(\mathcal{Y})\boldsymbol{R}_y(\mathcal{P})\boldsymbol{R}_x(\mathcal{R}) =$$

$$\begin{pmatrix} \cos\mathcal{Y}\cos\mathcal{P} & \cos\mathcal{Y}\sin\mathcal{P}\sin\mathcal{R} - \sin\mathcal{Y}\cos\mathcal{R} & \cos\mathcal{Y}\sin\mathcal{P}\cos\mathcal{R} + \sin\mathcal{Y}\sin\mathcal{R} \\ \sin\mathcal{Y}\cos\mathcal{P} & \sin\mathcal{Y}\sin\mathcal{P}\sin\mathcal{R} + \cos\mathcal{Y}\cos\mathcal{R} & \sin\mathcal{Y}\sin\mathcal{P}\cos\mathcal{R} - \cos\mathcal{Y}\sin\mathcal{R} \\ -\sin\mathcal{P} & \cos\mathcal{P}\sin\mathcal{R} & \cos\mathcal{P}\cos\mathcal{R} \end{pmatrix} \tag{8}$$

Z prvních dvou prvků matice v prvním sloupci můžeme vyjádřit $\cos\mathcal{P}$ a ze třetího prvku $\sin\mathcal{P}$.

$$cos^2\mathcal{P} = {r_{11}}^2 + {r_{21}}^2 \rightarrow cos\mathcal{P} = \pm\sqrt{{r_{11}}^2 + {r_{21}}^2} \tag{9}$$

$$sin\mathcal{P} = -r_{31} \tag{10}$$

Nyní můžeme vyjádřit vztah pro výpočet prvního úhlu pitch.

$$\mathcal{P} = atan2(-r_{31}/\sqrt{{r_{11}}^2 + {r_{21}}^2}) \tag{11}$$

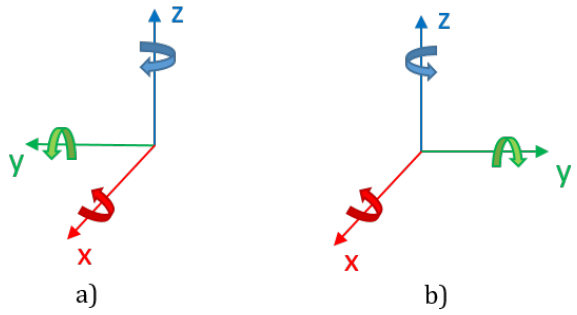
Obdobným způsobem z prvků v prvním sloupci a posledním řádku matice vyjádříme vztahy pro zbylé úhly, yaw a roll.

$$\mathcal{Y} = atan2(r_{21}/r_{11}) \tag{12}$$

$$\mathcal{R} = atan2(r_{32}/r_{33}) \tag{13}$$

Nakonec musíme vyřešit správný převod získaných souřadnic polohy a orientace vyjádřených v pravotočivém souřadnicovém systému (MRS Simulace) do levotočivého (UEds) k zajištění shodného pohybu dronu v obou simulacích. Rozdíl v levotočivém a pravotočivém systému je, že invertuje osu y viz obr. 3.4. Je tedy potřeba pouze invertovat y-ové hodnoty polohy.

$$y_{UEds} = -\,y_{MRSSimulace} \tag{14}$$



Obrázek 3.4. Souřadnicové systémy: (a) levotočivý používaný v UE5 (b) pravotočivý používaný v MRS Simulaci

Rotace kolem os v pravotočivém systému jsou kladné po směru hodinových ručiček a v levotočivé naopak v proti směru. Avšak v Unreal Enginu jsou kladné rotace definovány jinak oproti běžné konvenci.

Rotace kolem os x a y zůstávají kladné ve směru hodinových ručiček. Pouze rotace kolem osy z je definována kladně proti směru hodinových ručiček. Můžeme napsat vztahy:

$$\mathcal{R}_{UEds} = \mathcal{R}_{MRSSimulace} \tag{15}$$

$$\mathcal{P}_{UEds} = -\mathcal{P}_{MRSSimulace} \tag{16}$$

$$\mathcal{Y}_{UEds} = -\mathcal{Y}_{MRSSimulace} \tag{17}$$

Úhel rotace kolem osy x zůstává shodný v obou systémech. Zbývající úhly rotací kolem os y a z musíme invertovat z důvodu invertované osy y a protichůdných směrů rotací v případě osy z, jak si můžeme všimnou z Obrázku 3.4 .

3.3.2 Získání dat ze senzorů z UEds

Pro získaná data z API rozhraní klienta dronu byli vytvořeny dva informační kanály (tzv. ROS Topic), do kterých jsou příchozí data odesílána a můžou být následně zpracovány pro potřeby vizualizace v nástroji Rviz nebo pro další zpracování algoritmy autonomního letu.

Informační kanály jsou pojmenovány podle jedinečného označení dronu v objektu dynamické simulace. Tímto je třída `DroneControllerRos` připravená i na vytvoření více instancí v případě simulace více dronů.

ROS Topic pro data z RGB kamery je pojmenován takto:

```
"/" + unikatni oznaceni uav + "/rgbd/image_raw/compressed"
```

Data obrazu z RGB jsou získána API rozhraním v kompresní binární formě ve formátu `jpeg`. Pro přenos obrazu z kamery poskytuje ROS předdefinovanou strukturu zprávy `CompressedImage Message`. Po předchozím úspěšném přijetí dat jsou data přiřazena do zprávy a odesláno do příslušného kanálu. Hlavička zprávy může být doplněna o čas odeslání zprávy, případně souřadnicový systém kamery pro další zpracování v ROSu.

```
sensor_msgs/CompressedImage Message

----------------------
std_msgs/Header header
string format
uint8[] data
```

ROS Topic pro data z LiDARu je pojmenován takto:

```
"/" + unikatni oznaceni uav + "/lidar/points"
```

Obdobně jako v případě dat z kamery je pro ukládaní dat z LiDARu tzv. mračna bodů (PointCloud) v ROSu široce užívaná struktura zprávy `PointCloud2`.

```
sensor_msgs/PointCloud2

----------------------
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Hlavička zprávy je stejná jako v případě kamery. Zde musíme přiřadit souřadnicový systém MRS Simulátoru, ve kterém bude mračno bodů interpretováno. Parametry `height` a `width` odpovídají vertikálnímu, resp. horizontálnímu rozlišení LiDARu neboli počtu vyslaných paprsků ve vertikálním, resp. horizontálním směru. Ve struktuře `PointField` definujeme strukturu jednoho bodu z mračna, která obsahuje pozici bodu a délku paprsku. Body jsou ve zprávě uloženy v 1D poli (data) pro větší efektivitu, a je nutné definovat posun (point_step) v bytech mezi jednotlivými body. Každý bod obsahuje čtyři hodnoty typu float (x, y, z, délka paprsku), což znamená, že posun je 16 bytů. Celková velikost 1D pole v bytech je dána celkovým počtem bodů násobeným posunem (row_step = width * height * point_step).

Z API rozhraní dronu získáme pozici LiDARu, orientaci a délku paprsků v souřadnicovém systému UEds, ze kterých vypočítáme pozice bodů mračna. Zde není potřeba provádět žádné transformace jako v případě pozice a orientace dronu, protože orientace paprsků LiDARu jsou již vyjádřeny v souřadnicovém systému dronu. Pro získání pozice i-tého bodu mračna už jen přičteme délku d_i i-tého paprsku ve směru jeho orientace $\mathbf{o_i}$.

$$\mathbf{point_i} = d_i \cdot \mathbf{o_i} \tag{18}$$

Po naplnění struktury je zpráva odesláno do informačního kanálu.

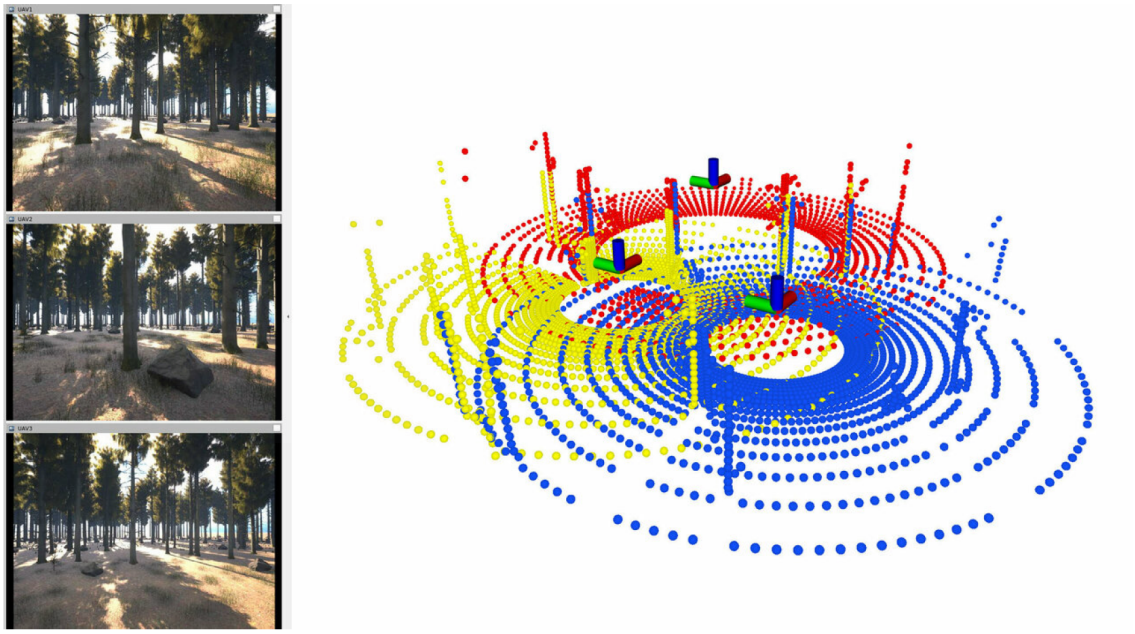
3.3.3 Integrace do MRS Simulátoru

Uzel MRS Simulátoru je implementován jako Nodelet, což je v rámci ROSu výkonný nástroj, který při zasílání zpráv mezi uzly v rámci jednoho Nodelet manažera nevytváří kopie zpráv, ale komunikace probíhá pomocí ukazatelů (pointerů) na zprávy. Takto efektivní správa paměti v rámci Nodeletu je zejména výhodná při zasílaní velkého objemu dat, jako je v našem případě mračno bodů, a nezatěžuje tak komunikaci jako při zasílání zpráv přes protokol TCP/IP, používaný standardně v ROSu. Nodelet má také vlastního správce vláken umožňující efektivní paralelní zpracování zpráv (callbacků).

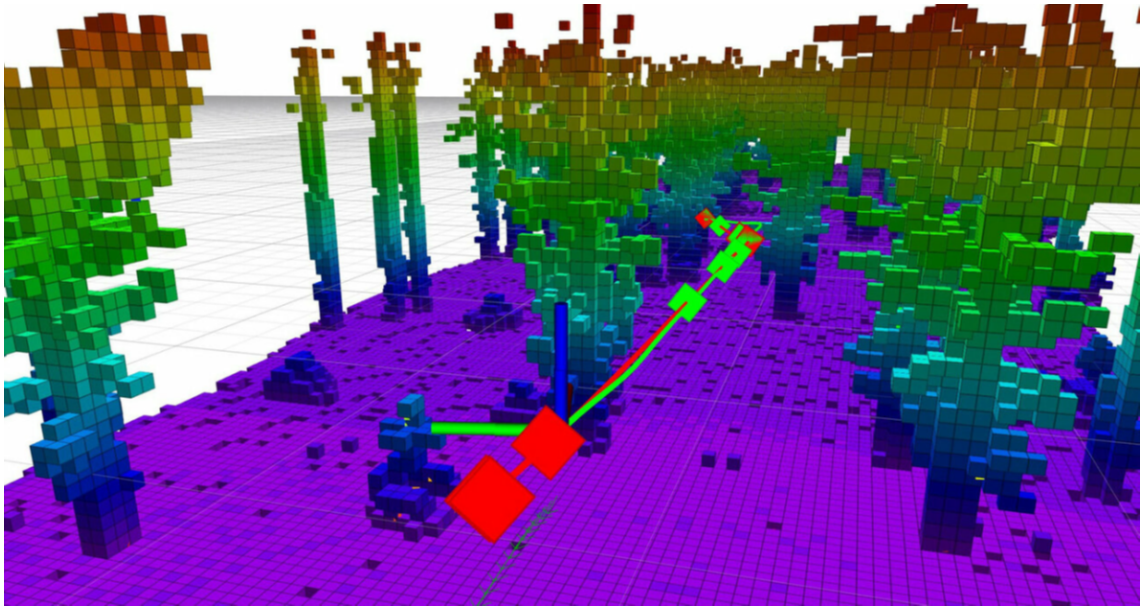
K zajištění cyklického volání metod se používá ROS Timer. Tímto nástrojem můžeme metodám nastavit, s jakou periodou se spouští. V MRS Simulátoru je ROS Timer používán k periodickému krokování dynamické simulace. Stejným způsobem je ve třídě `DroneControllerRos` implementována periodická aktualizace polohy dronu a čtení dat ze senzorů.

Při inicializaci MRS Simulátoru se pro každý simulovaný dron vytvoří jedna instance třídy `DroneControllerRos`. Díky tomu, že každá instance třídy `DroneControllerRos` vytváří svůj vlastní Timer, který běží nezávisle na ostatních instancích, je komunikace mezi systémy výrazně efektivnější.

Na obrázcích 3.5 a 3.6 můžeme vidět MRS Simulátor s integrací UEds v jednom z vytvořeným prostředích, o kterých pojednává následující kapitola. Na obrázku 3.5 je zobrazena simulace tří dronů v reálném čase se simulovanými senzory LiDARu a kamery. Na obrázku 3.6 je zobrazena vizualizace lokalizace, mapování a plánování trajektorie využívající senzorická data z LiDARu.



Obrázek 3.5. Simulace tří UAVs s RGB kamerou a LiDARem



Obrázek 3.6. Vizualizace lokalizace a mapování v MRS UAV Systému

Kapitola 4
Tvorba virtuálních prostředí

V této kapitole se zaměříme na proces tvorby scén: skladu, lesa a jeskyně. Virtuální prostředí budou sloužit jako simulační platforma pro testování a vývoj metod řídících algoritmů, umělé inteligence a rozpoznávání obrazu používaných při autonomním letu.

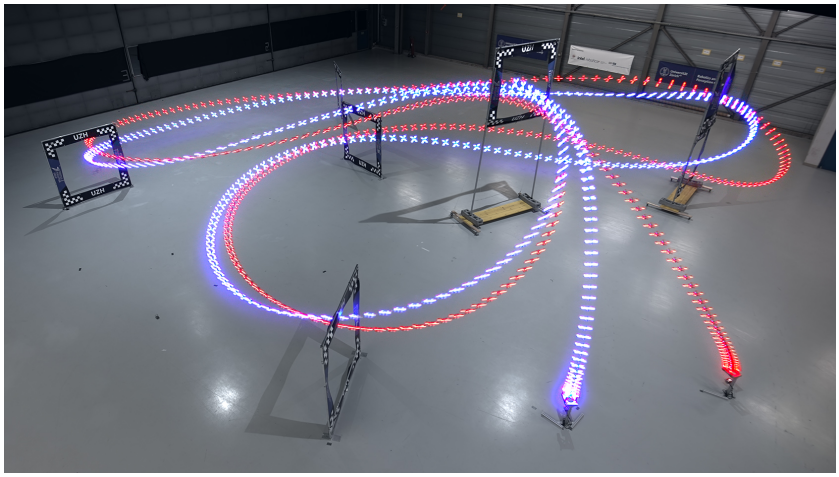
První prostředí bylo vytvořeno ručně, což je časově náročný a zdlouhavý proces. Pro zbylé prostředí jsme zvolili efektivnější přístup a využili metodu procedurálního generování. Tato technika umožňuje vytváření rozmanitých a rozsáhlých až nekonečných scén bez ručního umisťování každého detailu. Procedurálně generované scény vznikají algoritmicky na základě počátečních parametrů generátoru šumu a tzv. *seed*. Díky tomu, že generátoru poskytneme stejný *seed*, vygeneruje deterministicky stejnou sérii náhodných čísel, což zaručuje konzistentní vzhled prostředí při každém opakovaném generování, a navíc jednoduchou změnou *seedu* vygenerujeme nové rozdílné prostředí.

4.1 Manuální tvorba - Sklad

První scéna je vytvořena kombinací modelů dostupných na Unreal Engine Marketplace [18], kde komunita sdílí své vytvořené modely, textury, případně celá prostředí. Získané modely byly pečlivě vybrány tak, aby vytvořily realistické prostředí skladu.

Obrázek 4.1. Ručně vytvořené prostředí skladu

Toto prostředí může být využito například k učení agresivního letu závodních dronů. Výzkumníci švýcarské univerzity v Curychu trénovali v podobném simulačním prostření svoji umělou inteligenci „Swift“ [19], která vyhrála v závodě dronů nad světovými šampiony. Tento úspěch představuje další milník v oblasti umělé inteligence, když autonomní systém porazil lidské šampiony ve fyzickém sportu. Možnost rychlého autonomního letu



Obrázek 4.2. Autonomní dron (modrá) vs lidský pilot (červená) [19]

má potenciál využití v oblastech jako monitorování životního prostředí (např. sledování kůrovce v lesích) nebo záchrana lidí a jejich lokalizace v rozsáhlých komplexů budov.

4.2 Nekonečný procedurálně generovaný les

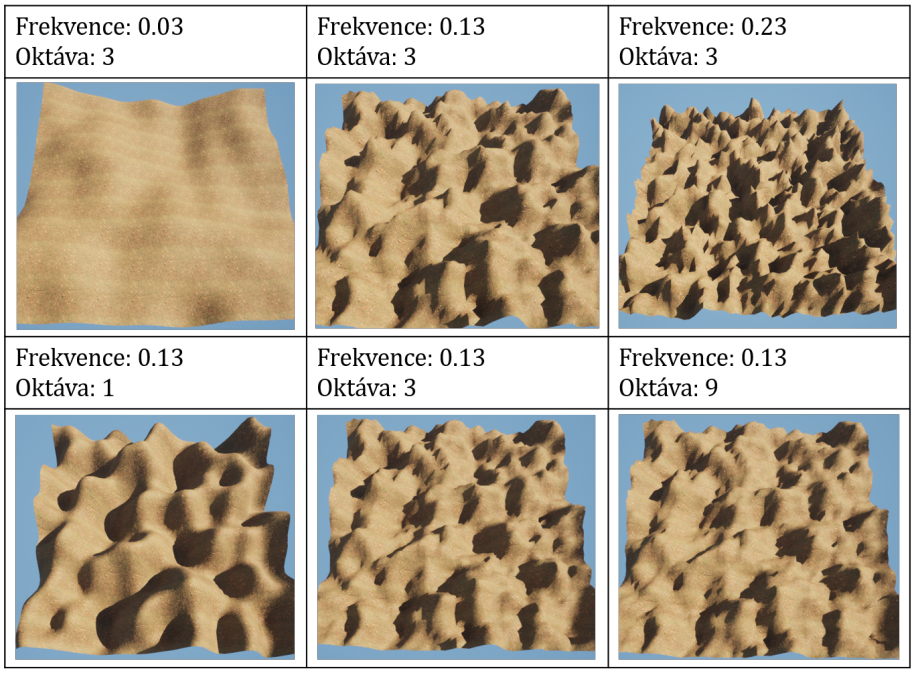
Základem pro tvorbu lesního prostředí jsou fotogrammetrické modely stromů, kamenů a nízké vegetace, dostupné v UE Marketplace, které jsou rozmisťovány do terénu lesa. Pro vytvoření terénního reliéfu byl zvolen deterministický generátor náhodných čísel *Perlin Noise* [20–21] a pro procedurální umístění vegetace jsme využili v UE5 komponentu *Procedural Content Generation* (PCG) [22].

4.2.1 Algoritmus Perlin noise

Perlinův šum byl vytvořen Kena Perlinem v reakci na nedostatek realistických vizuálních efektů v počítačové grafice v 80. letech 20. století. Jeho algoritmus umožňuje generovat textury s přirozeně vypadajícím šumem. Algoritmus nachází uplatnění v mnoha různých oblastech, jako je například animace efektů kouře, simulace vln na hladině vody, tvorba oblaků nebo generování terénu.

Perlinův šum je generován na základě vstupních parametrů. Základními parametry jsou:

- Amplituda – Určuje rozsah hodnot, který šum může nabývat.
- Frekvence – Ovlivňuje hustotu pruhů nebo vln v šumu. Vyšší frekvence znamená více detailů, zatímco nižší frekvence produkují hladší šum.
- Oktáva – Oktávy umožňují kombinovat více vrstev šumu s různými frekvencemi a amplitudami. Čím vyšší je počet oktáv, tím více detailů a složitosti může mít výsledný šum.



Obrázek 4.3. Perlin Noise: Porovnání různých hodnot frekvencí a oktáv

Na obrázku 4.3 jsou zobrazeny terénní reliéfy vygenerované s různým nastavením parametrů. Pro vytvoření prostředí lesa byl zvolen terén s nejpřirozenějším vzhledem, kde byl parametr frekvence nastaveny na 0.03 a použity byly tři oktávy.

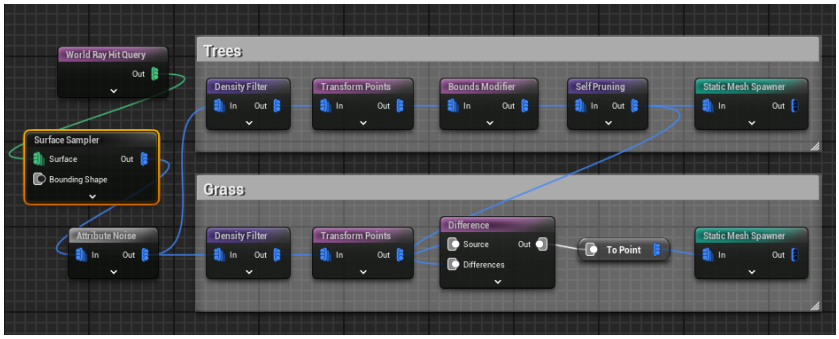
4.2.2 Komponenta procedurálního generování v UE5

PCG komponenta v Unreal Engine je flexibilní nástroj pro procedurální generování obsahu, jako je například vegetace, budovy měst a další detailní prvky, které dotvářejí scénu. Tato komponenta umožňuje tvůrcům vytvářet dynamické a rozmanité prostředí na základě definovaných pravidel, což umožňuje snadnou modifikaci a rozšiřování prostředí bez potřeby ručního vytváření každého detailu.

Pravidla pro generování prostředí jsou definována v PCG Blueprintu, který je reprezentován uzlovým grafem (viz Obrázek 4.4). Terén prostředí je vzorkován v uzlu *Surface Sampler*, kde jsou generovány náhodné body na povrchu terénu. Tyto body jsou následně zpracovány dalšími uzly, které postupně filtrují jejich hustotu, aby byla dosažena požadovaná koncentrace stromů. Díky vhodnému nastavení filtrů můžeme snadno upravit hustotu zalesnění.

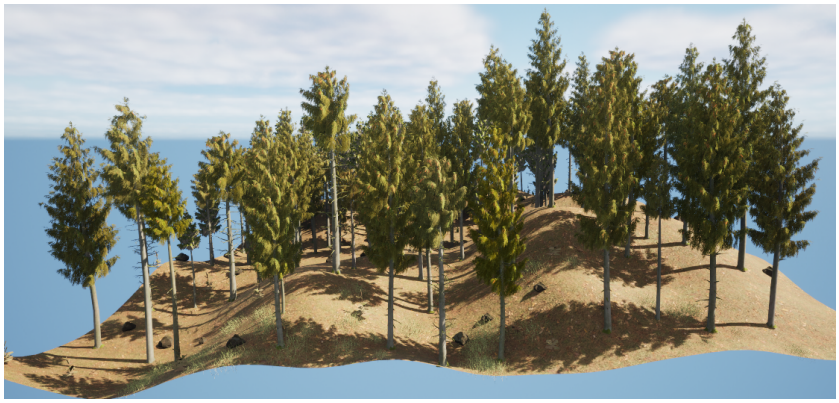
Dále jsou náhodně měněny orientace bodů v rozmezí od 0 do 360 stupňů. Zároveň je jejich velikost, a tím i velikost generovaného modelu v daném bodě, náhodně upravena v rozsahu od 0.5 do 1. Tato úprava přispívá k větší diverzitě a přirozenosti prostředí. V uzlu *Bounds Modifier* jsou každému vygenerovanému bodu přiřazeny hranice, do kterých by se měl model stromu vejít. V případě, že dochází k průniku hranic, jsou příslušné body odfiltrovány v uzlu *Self Pruning*, aby se zabránilo nepřirozenému překrývání modelů stromů.

Nakonec je na pozici každého bodu umístěn náhodně vybraný model stromu. Pro generaci nízké vegetace je použit obdobný postup s vyšší hustotou bodů.



Obrázek 4.4. PCG Blueprint

Obrázek 4.5 ukazuje část vygenerovaného světa pomocí Perlinova šumu a PCG komponenty.



Obrázek 4.5. Procedurálně generovaná část lesního prostředí

4.2.3 Rozdělení světa a implementace

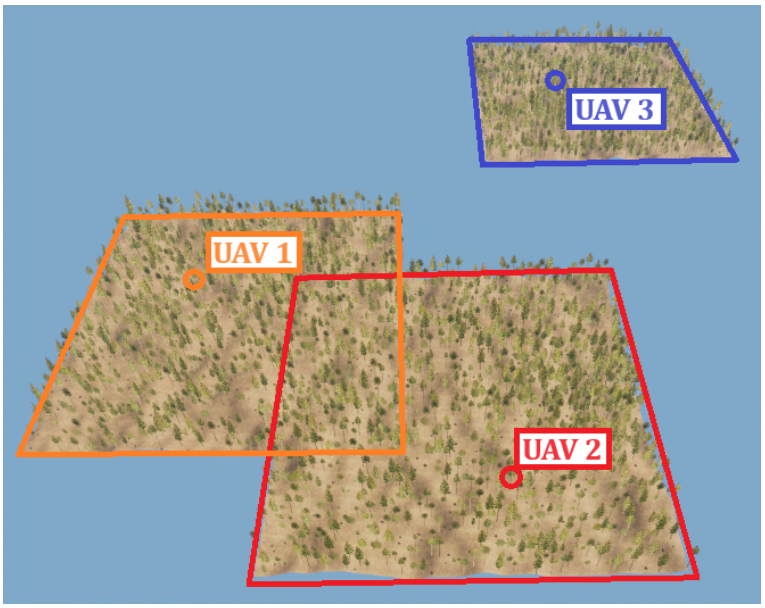
Pro dosažení nekonečného lesního prostředí byl vyvinut manažer světa ve třídě *ChunkManager*, který má za úkol rozdělit svět do mřížky jednotlivých kusů (chunks) a aktualizovat prostředí podle aktuální polohy dronů získaných z herního režimu simulátoru UEds. Nové kusy prostředí jsou vytvořeny pouze v okolí dronu, který je právě v nastavené viditelné vzdálenosti od dronu. Pro zajištění efektivní správy paměti je implementován mechanismus, který odstraňuje instance kusů prostředí, které již nejsou viditelné z žádného směru (x, y) ani od žádného dronu ve scéně. Tento přístup minimalizuje nadbytečné zatížení paměti a umožňuje optimalizované generování nekonečného prostředí. Manažer si udržuje seznam již vytvořených kusů lesa, které jsou jednoznačně určeny polohou svého krajního bodu. Díky znalosti velikosti vytvářených kusů můžeme na základě celočíselného dělení určit, ve které oblasti se dron nachází, a reagovat tak na generování nových kusů prostředí. Pro úplnost je na Obrázku 4.7 zobrazen diagram tříd implementovaného generátoru lesa.

Pro vytváření terénu v UE5 je použita třída ProceduralMeshComponent [23], která umožňuje dynamické vytváření a úpravu meshů za běhu aplikace. Práce s touto třídou je podobná práci se strukturou Vertex Buffer Object v OpenGL. Je nutné vyplnit pozice vrcholů meshe, velikosti normál, UV souřadnice pro korektní texturování a definovat pořadí vrcholů tvořící jednotlivé trojúhelníky meshe.

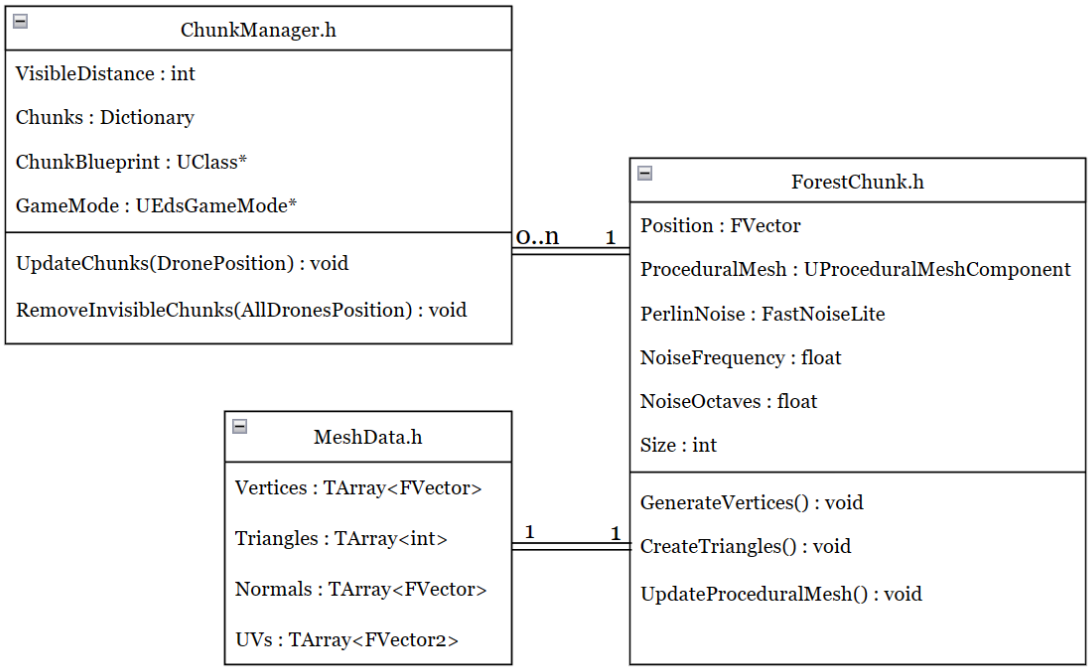
Terén v třídě ForestChunk je rozdělen do mřížky, kde každý vrchol má svou souřadnici xy a odpovídající hodnotu z, která je generována pomocí zmíněného Perlin Noise. Velikosti normál jsou vypočteny pomocí dostupné metody v UE5, konkrétně UKismetProceduralMeshLibrary::CalculateTangentsForMesh [24]. Nakonec po vygenerování meshe je aplikována PCG komponenta generující vegetaci.

Díku použití *seedu* v obou komponentách procedurálního generování zůstává svět konzistentní i při opětovném generování již zaniklých kusů světa, což je důležité pro reálné ztvárnění světa i pro algoritmy lokalizace a mapování autonomního letu dronů.

Na Obrázku 4.6 vidíme proces generování nekonečného lesního prostředí při spuštěné simulaci tří dronů.



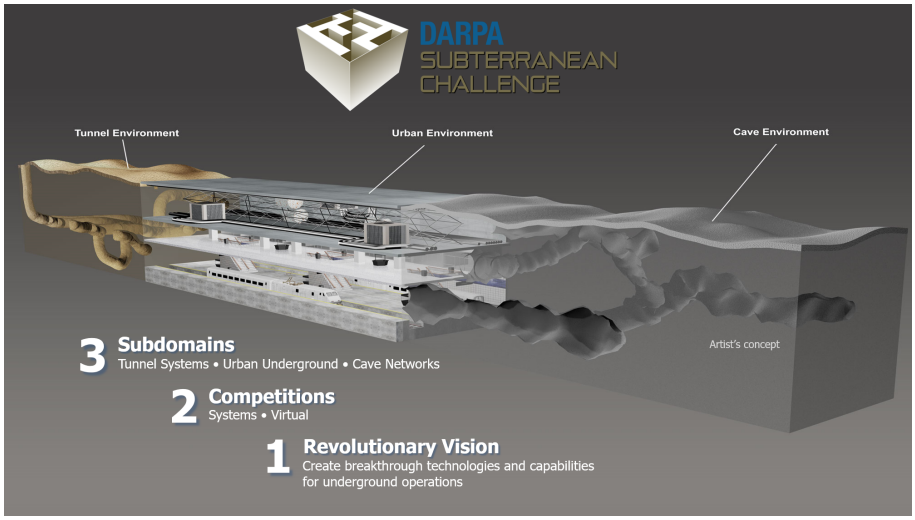
Obrázek 4.6. Generování nekonečného prostředí



Obrázek 4.7. Diagram tříd implementace nekonečného lesa

4.3 Procedurálně generované jeskynní prostředí

Inspirací pro vytvoření uzavřeného jeskynního prostředí je *DARPA Subterranean Challenge* [25], soutěž zaměřená na vývoj autonomních vozidel financovaná *Defense Advanced Research Projects Agency - DARPA*, nejvýznamnější výzkumnou organizací Ministerstva obrany Spojených států. Tato soutěž sloužila jako motivace k nalezení nových přístupů rychlého mapování, navigace a vyhledávání v podzemních prostředích, včetně jeskynních systémů. Tyto prostředí představují výzvu i pro zkušené záchranáře, navíc terén se může v průběhu času měnit, což z něj dělá příliš riskantní prostor pro vstup osob.



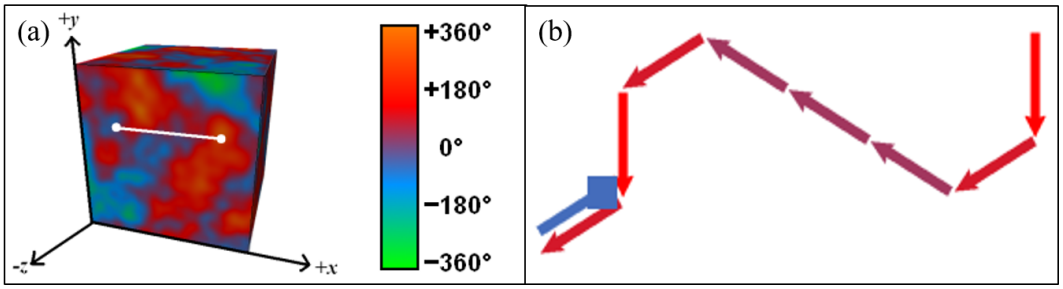
Obrázek 4.8. DARPA Subterranean Challenge [25]

Procedurální generovaní jeskyně využívá dva hlavní algoritmy. První je Perlin Worms [26–27], který je použitý na vytvoření struktur jeskyní. Druhý algoritmus Marching Cubes [28] slouží k vytvoření polygonálních ploch, které reprezentují povrch jeskyně. Déle je použit přístup vzorkování povrchu jeskyně pro doplnění detailů jako jsou například krápníky.

4.3.1 Generování souřadnic jeskyně - Algoritmus Perlin Worms

Perlin Worms je algoritmus využívající Perlin noise k generování vinutých struktur, které vizuálně připomínají červy. Tento algoritmus nachází své uplatnění v procedurálním generování přirozeně vypadajících struktur jako například jeskyně, řeky nebo silnice.

Generování struktur pomocí algoritmu Perlin Worms je založeno na náhodném vytváření segmentů, které vytvářejí zakřivené linie. Tento proces začíná určením pozice startovního segmentu, pro který se získávají hodnoty Perlin noise. Na základě těchto hodnot jsou vypočítány úhly, které určují směr a polohu následujícího segmentu. Každý nový segment je generován na základě předchozího, čímž vzniká postupně se rozvíjející struktura.



Obrázek 4.9. (a) Interpretace šumu na úhel (b) Vytvoření segmentů „červa“ [26]

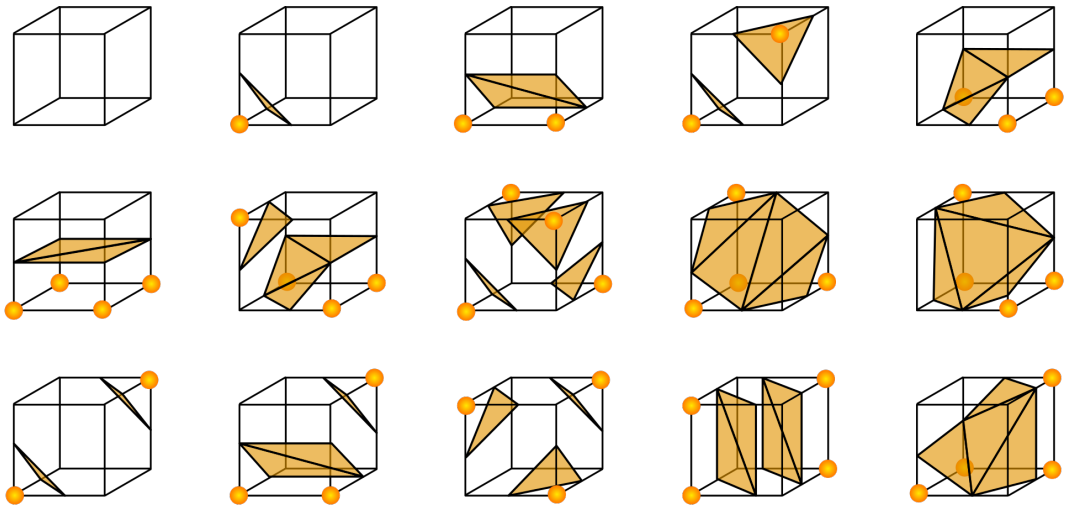
Hlavní vlastností algoritmu, ale i jeho nevýhodou je, že tvar a směr vytvořené struktury jsou zcela ovlivněny náhodnými hodnotami Perlin noise, což znamená, že není možné přesně kontrolovat jejich podobu. Jedinou možností jak ovlivnit výsledný vzhled je změna počáteční pozice „červa“ nebo *seedu* použitého generátoru šumu.

Ukončení generování můžeme řídit například počtem požadovaných segmentů nebo i libovolnou podmínkou. V případě generování struktur jeskyně je ukončovací podmínka minimální vzdálenost posledního segmentu ke stanovenému cílovému bodu.

4.3.2 Algoritmus Marching cubes

Algoritmus Marching Cubes slouží k vykreslování izoploch v objemových datech (trojrozměrné pole hodnot). Základní myšlenkou je definice voxelu (krychle) pomocí hodnot v jeho osmi vrcholech. Pokud má jeden nebo více vrcholů krychle hodnoty menší než uživatelem určená hodnota izoplochy a zároveň jeden nebo více vrcholů má hodnoty větší než tato hodnota, víme, že voxel musí nějak přispět k celkové izoploše.

Určením hran krychle, které jsou protínány izoplochou, můžeme vytvořit trojúhelníky, které dělí krychli mezi oblasti pod a nad izoplochou. Podle možných hodnot ve vrcholech krychle existuje $2^8 = 256$ konfigurací pro vytvářené trojúhelníky (viz Obrázek 4.10). Dvě z nich jsou triviální, když jsou všechny body uvnitř nebo vně kostky, a nepřispívají k celkové izoploše. Pro všechny ostatní konfigurace musíme určit, kde izoplocha protíná hrany krychle, a použít tyto body k vytvoření jednoho nebo více trojúhelníků. Pozice těchto bodů mohou být vypočítány interpolací podél hran neboli



Obrázek 4.10. 15 konfigurací voxelu [29]

interpolací hodnot v přilehlých vrcholech, nebo mohou být ve výchozí pozici uprostřed hrany. Interpolované polohy obvykle poskytují lepší stínování a hladší plochy.

Jednotlivé konfigurace protínajících hran jsou uloženy v tabulce, dostupné z [28] (zdroj také obsahuje podrobný pseudokód algoritmu). Do tabulky můžeme jednoduše přistupovat pomocí jednoznačného indexu dané konfigurace.

Tento proces aplikujeme na celý objem. Propojením příspěvků všech voxelů získáme reprezentaci celkového povrchu.

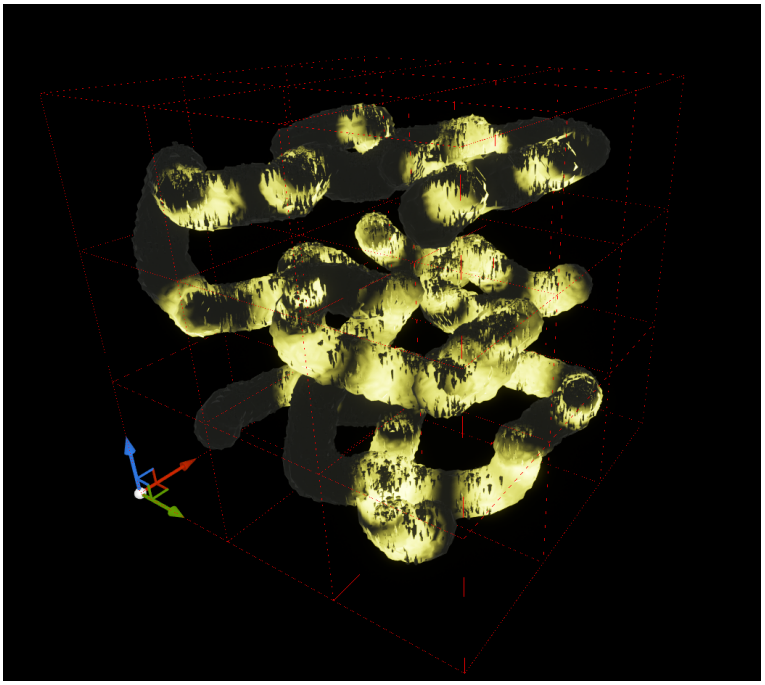
4.3.3 Rozdělení světa a implementace

Obdobně jako v případě lesního prostředí řídí celý proces generování manažer ve třídě *CaveManager*. Proces generování je rozdělen do čtyř částí: *Inicializace prázdného světa, Generace souřadnic tunelů jeskyně, Vykreslení povrchu tunelů a Dekorace vnitřního prostoru jeskyně.*

Při *inicializaci světa* jsou podle uživatelem nastaveného rozsahu světa v osách x,y,z vytvořené instance třídy *MarchingRender*, jak je vidět na Obrázku 4.11. Třída obsahuje strukturu pro ukládání objemových dat a implementuje Marching Cubes algoritmus. Pro vytvoření polygonální meshe je použita stejná třída ProceduralMeshComponent jako u lesního prostředí (viz Kapitola 4.2.3). *CaveManager* si udržuje seznam částí světa pro pozdější úpravu objemových dat a vytvoření polygonální meshe. K částí světa lze přistoupit indexem, který je jednoznačně určený polohou vrcholu krychle reprezentující část světa.

Generace souřadnic tunelů je implementováno pomocí prohledávání do hloubky. Prohledávání začíná ve vrchních vrstvách světa a postupuje směrem k nižším vrstvám, neboli nemůže se vracet k vyšším vrstvám, čímž je zamezeno vytvoření jedno dlouhého tunelu bez odboček. Pořadí prohledávání je dáno seřazením sousedních kusů světa pomocí šumu vygenerovaného ve středu jednotlivých kusů. Použitím deterministického prohledávání do hloubky a seřazení pomocí Perlinova šumu je zajištěno získání vždy stejné podoby jeskynních tunelových struktur pro daný *seed*.

Při prohledávání jsou mezi dvěma sousedními kusy světa resp. jejich středů vytvořeny tunely zmíněným algoritmem Perlin Worms. Cesta tunelu je generována na základě dvou vektorů. První vektor udává směr k cílovému sousednímu středu a tím nutí tunel přibližovat se k cíli. Druhý vektor vytvořený generátorem šumu přispívá k náhodnému zakřivení tunelu. Na obrázku 4.12 je první vektor znázorněn modře a druhý zeleně.



Obrázek 4.11. Jeskynní prostředí velikosti 3x3x3

Výpočet náhodného vektoru zakřivení tunelu je následující.

```
float NoiseValueXY = (Noise->GetNoise(X, Y, Z) + 1) / 2;
float NoiseValueXZ = (Noise->GetNoise(Y, X, Z) + 1) / 2;

FVector PerlinDirection = FVector(FMath::Cos(NoiseValueXY * 2.0f * PI),
                                  FMath::Sin(NoiseValueXY * 2.0f * PI),
                                  FMath::Sin(NoiseValueXZ * 2.0f * PI));
```

Získaný šum v aktuálním bodě na cestě tunelu může interpretovat jako úhel vektoru na jednotkové kružnici, čím získáme azimut a elevaci. Tyto dva úhly pomocí goniometrických funkcí přepočteme na náhodný vektor generovaný Perlinovým šumem.

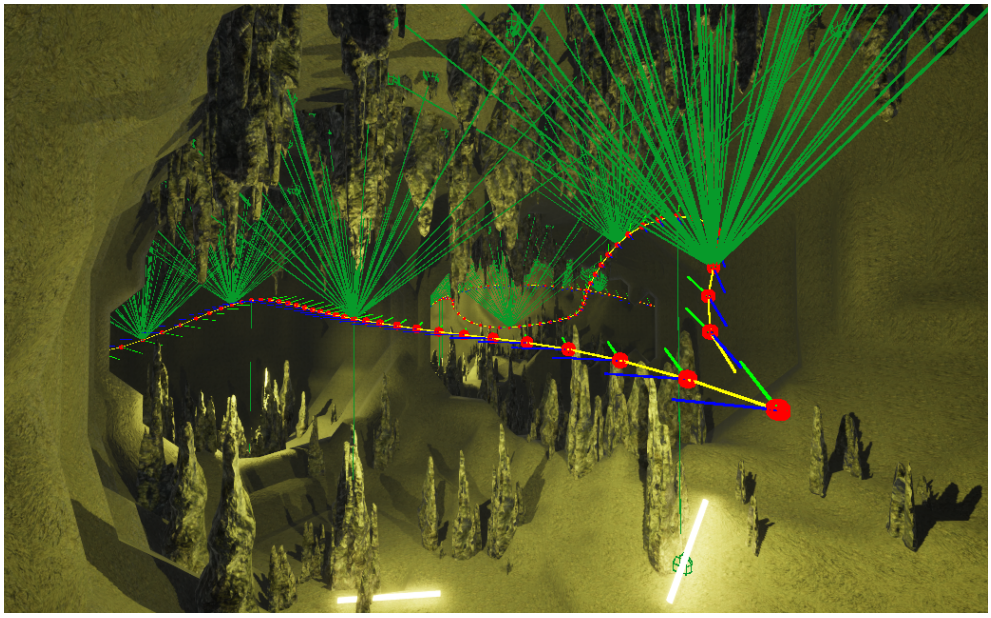
Nyní lineární kombinací získáme směr jednoho segmentu tunelu, posuneme se o konstantní délku a celý proces opakujeme, dokud se tunel dostatečně nepřiblíží k cílovému středu sousedního kusu světa.

```
FVector RandomDirection = DesireDir * 0.6 + PerlinDirection * 0.4;
RandomDirection = Lenght * RandomDirection.Normalize();
CurrentLocation = CurrentLocation + RandomDirection;
```

V každém vygenerovaném bodě se vytváří kulová plocha, jejíž poloměr je určen generátorem šumu v uživatelem nastaveném rozmezí. S volbou poloměru souvisí i volba kroku mezi body na cestě tunelu. Tyto parametry musí být správně proporčně zvoleny, aby nedocházelo ke vzniku oddělených nepřístupných tunelů. Ze znalosti nerovnice kulové plochy (1) jsou všechny bodu uvnitř plochy nastaveny jako prázdné a tato informace je uložena v objemových datech části světa, do které vygenerovaný bod přísluší.

$$(x - X)^2 + (y - Y)^2 + (z - Z)^2 \leq R^2 \tag{1}$$

K zajištění souvislosti generované polygonální plochy je nutné tuto informaci uložit i do objemových dat sousedních částí světa, protože vygenerovaná kulová plocha může zasahovat najednou do více oblastí.



Obrázek 4.12. Vytvoření detailů jeskynního prostředí

Nyní po vyplnění objemových dat můžeme aplikovat algoritmus Marching Cubes v každé části světa a vytvořit tak polygonální povrch jeskyně.

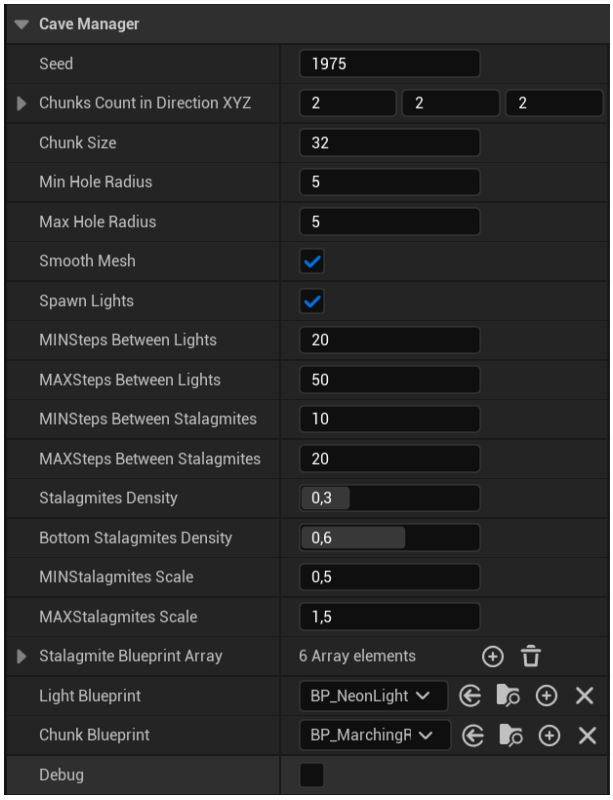
4.3.4 Vytvoření detailů jeskynního prostředí

Pro dokreslení detailů jeskynního prostředí a zvýšení realismu jsou ve vnitřním prostoru jeskyně dogenerovány krápníky různých tvarů a velikostí, a průzkumná světla. Generování probíhá podél vygenerovaných bodů na cestě tunelu (viz Obrázek 4.12). Rozmístění detailů je náhodné v rámci stanoveného maximálního a minimálního kroku mezi detaily. Náhodný výběr kroku je určen Perlinovým šumem aktuálně zpracovávaného bodu na cestě.

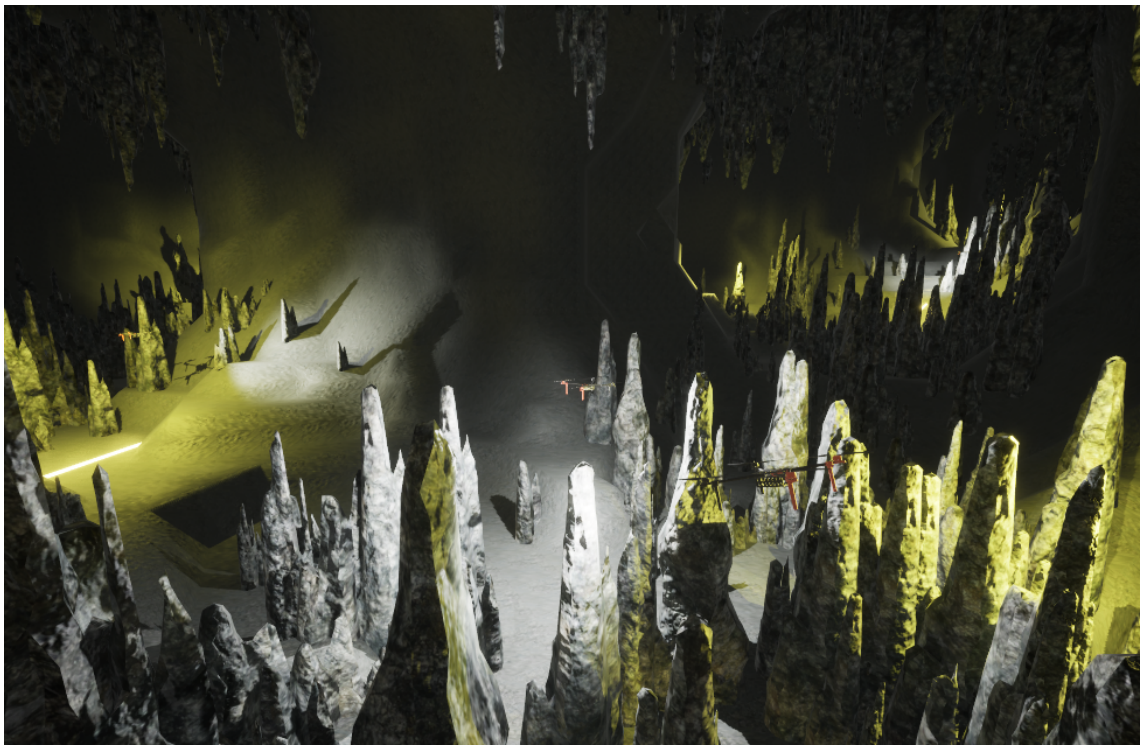
Způsob generování horních krápníků – stalaktitů je následující: ve směru kulové výseče je vyslán určený počet paprsků (použita je metoda sledování paprsku neboli ray casting). Po dopadu paprsků na povrch jeskyně je získána hodnota Perlinova šumu v daném bodě, a pomocí stanoveného prahu je rozhodnuto, zda na tomto místě bude nebo nebude vytvořen stalaktit, čímž můžeme ovlivnit množství krápníků ve scéně. Dolní krápníky – stalagmity jsou vytvořeny obdobným způsobem: paprsky jsou vyslány kolmo dolů z horních stalaktitů a pomocí Perlinova šumu je opět rozhodnuto, zda zde budou resp. nebudou vytvořeny stalagmity.

Náhodná velikost z rozsahu a náhodný model stalaktitů nebo stalagmitů jsou také vybrány na základě získaného šumu, což zajišťuje zachování konzistence prostředí daného zrna (seedu).

Veškeré nastavitelné parametry generátoru jeskyně jsou na obrázku 4.13 a na obrázku 4.14 je příklad vygenerovaného prostředí.



Obrázek 4.13. Vygenerovaného prostředí jeskyně



Obrázek 4.14. Vygenerovaného prostředí jeskyně se simulovanými drony

Kapitola 5
Vyhodnocení dosažených výsledků

V rámci poslední kapitoly vyhodnotíme dosažené výsledky. Rozebereme míru fotorealičnosti a složitost procedurálního generovaní ve vytvořených prostředích. Dále otestujeme výkonnost navrženého spojení fyzikální simulace s Unreal Enginem v rámci vytvořených prostředí i porovnáme naše řešení s konkurenčním simulátorem.

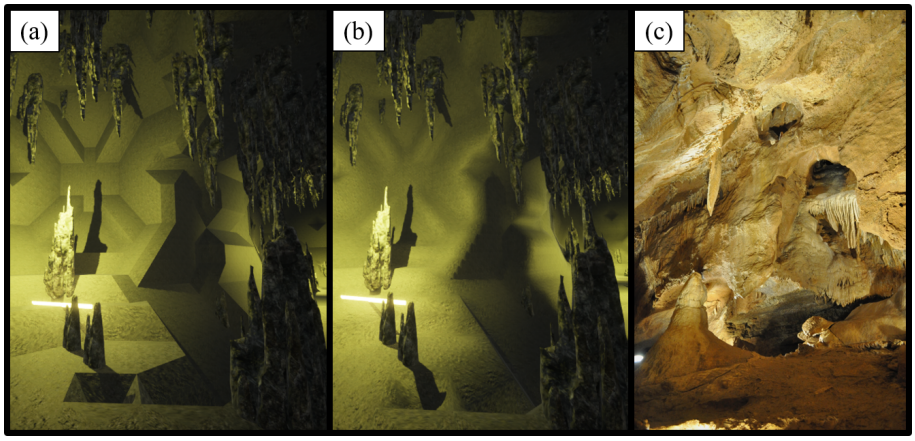
5.1 Zhodnocení procedurálně generovaných prostředí

Při vyhodnocení vlastností implementovaných metod procedurálního generování se zaměříme hlavně na vizuální kvalitu, která je zejména potřebná pro trénovaní metod počítačového vidění z obrazu simulovaných kamer, a výpočetní složitost generovaných modelů.

Lesní prostředí poskytuje věrně fotorealisticky vypadající scenérie lesa zejména díky fotogrammetrií získaných modelů vegetace a pokročilým metodám osvětlovacích modelů v Unreal Enginu. Využitím koherentního generátoru šumu vzniká rozmanitý kopcovitý terén dodávající na uvěřitelnosti prostředí. Díky tomu, že meshe kusů světa obsahují relativně málo vrcholů (10x10) oproti jeskynnímu prostředí, je tvorba terénu rychlá a může byt provedena v reálném čase, což umožňuje tvorbu nekonečného prostředí.

Obrázek 5.1. Porovnání virtuálního a reálného prostředí lesa: (a) UE5 (b) Šumavský les [30]

Výpočetní složitost generování *jeskynního prostředí* roste s množstvím objemových dat (voxelů). Vyšší rozlišení voxelů znamená více trojúhelníků v meshi, čímž je povrch jeskyně detailnější, ale zároveň se zvyšuje časová náročnost na její vytvoření. Také výpočet normál, které jsou důležité pro osvětlení a správné renderování (viz Obrázek 5.2), pro každý vrchol je časově náročný a výrazně prodlužuje generování. Z tohoto důvodu použitá metoda pro generování jeskyně nedosahuje takových kvalit jako prostředí lesa a jedná se spíše abstrakci přírodní jeskyně. Nicméně svojí komplexní strukturou lze použít jako náročné testovací prostředí pro metody lokalizace a mapování vyžívající především sensor LiDAR.



Obrázek 5.2. Porovnání virtuální a reálné jeskyně: UE5 (a) bez interpolací normál (b) s interpolací normál, (c) Koněpruské jeskyně [31]

V Tabulce 5.1 je uvedeno porovnání průměrných časů doby potřebné pro vygenerování prostředí velikosti 3x3 a rozměru jednoho kusu světa 5000x5000 cm.

Les	Jeskyně bez interpolace normál	Jeskyně s interpolací normál
0.347 s	0.965 s	6.519 s

Tabulka 5.1. Průměrná doba generace prostředí z 5 generací

5.2 Testování rychlosti získávání senzorických dat

Zpracování a získávání senzorických dat vyžaduje výpočetní čas, který závisí na několika faktorech, jako je komplexnost prostředí nebo počet simulovaných dronů ve scéně. Proto jsme provedli testy pro zjištění výkonu a použitelnosti navrženého simulátoru ve vytvořených prostředích skladu, lesa a jeskyně.

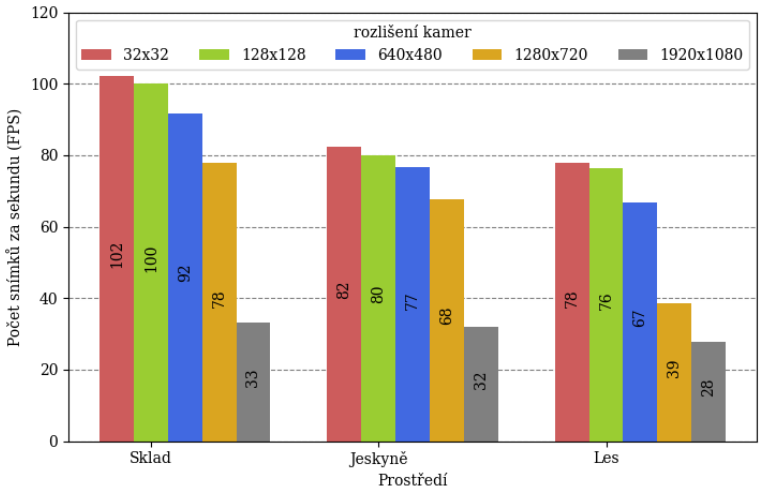
Zaměřili jsme se na otestování virtuálních senzorů: LiDARu a kamery. Veškeré testy byly provedeny na notebooku s procesorem Intel Core i7-13700HX a grafickou kartou Nvidia RTX 4060.

5.2.1 Kamera

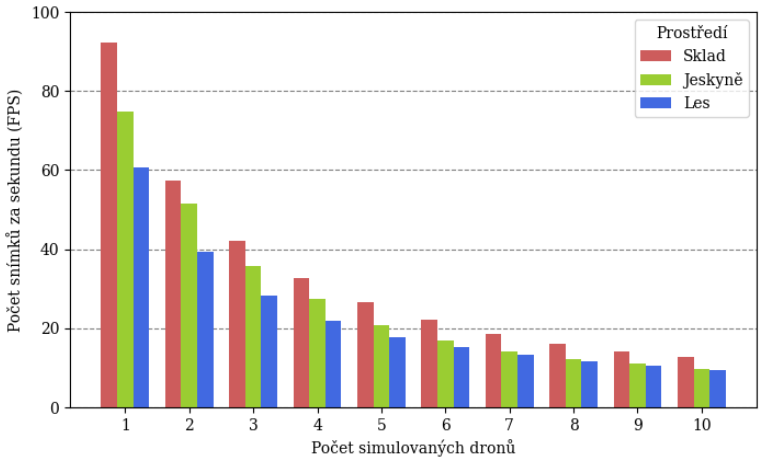
Simulovaná kamera je schopná díky pokročilému renderovacímu řetězci produkovat velmi věrná data obrazu prostředí s vysokých rozlišením, která jsou srovnatelná s výstupy z reálného senzoru kamery.

Otestovali jsme odezvu kamery ve vytvořených prostředích s různým nastavením rozlišení. Cílem testu bylo získání 50 snímků prostředí opakovaným dotazem a z délky trvání této operace byl vypočten počet snímků za sekundu. Výsledek testu při simulaci jednoho dronu je zobrazen na obrázku 5.3. Tento průběh měření má očekávaný trend. Pokles snímků je způsoben náročností operace vykreslení obrazu z pohledu dronu, která je závislá na komplexnosti daného prostředí (počtu vykreslovaných objektů) a nastaveném rozlišení kamery.

Déle byl proveden test při zvyšování počtu simulovaných dronů. Rozlišení kamer bylo fixně nastaveno na 640x480 pixelů. Dotazy na snímky byly pro každý dron vykonávány paralelně a následně byl vypočten průměr rychlosti odezvy ze všech dronů. Výsledek testu a pokles snímků s rostoucím počtem dronů je zobrazen na obrázku 5.4.



Obrázek 5.3. Srovnání rychlosti vykreslení obrazu z RGB kamery v počtu snímků za sekundu pro různá rozlišení kamery ve všech prostředí

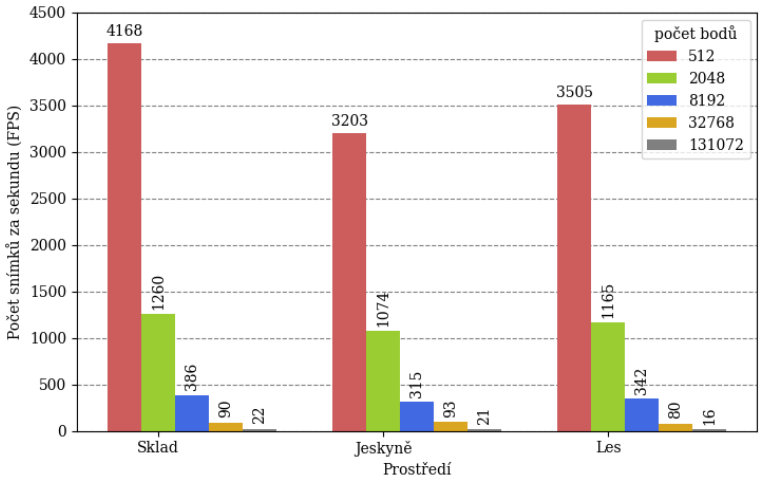


Obrázek 5.4. Srovnání rychlosti vykreslení obrazu z RGB kamer v počtu snímků za sekundu s rozlišením 640x480 při rostoucím počtu dronů

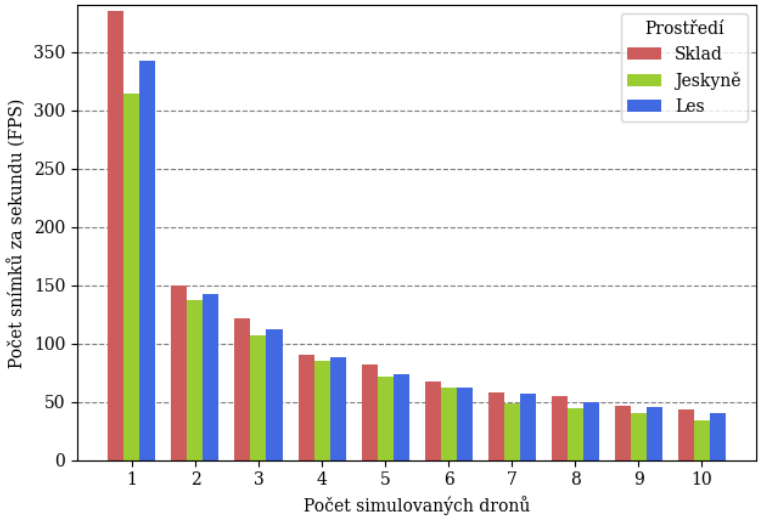
5.2.2 LiDAR

Odezvu simulovaného senzoru LiDARu jsme testovali v závislosti na počtu bodů ve výsledném mračnu bodů (point cloud). Provedený test proběhal stejně jako v případě kamery, jednalo se o získání požadovaného počtu snímků a následný výpočet odezvy senzoru. Na obrázku 5.5 je zobrazen výsledek testu. Můžeme si všimnout výrazného poklesu odezvy při zvyšovaném počtu bodů. Nicméně simulovaný LiDAR je schopný produkovat mračno bodů srovnatelně jako reálné senzory, kde se rychlosti získávání dat pohybují v jednotkách až desítkách hertz a počet získaných bodů se pohybuje ve stovkách tisíc v závislosti na modelu senzoru. Výkon senzoru v jednotlivých prostředích se výrazně neliší.

Obdobně jako v případě testování kamery byla zjištěna odezva simulovaných senzorů LiDARu při rostoucím počtu dronů s fixně nastaveným rozlišením mračna bodů 8192. Výsledek testu je zobrazen na obrázku 5.6.



Obrázek 5.5. Srovnání rychlosti odezvy LiDARu v počtu snímků za sekundu pro různá rozlišení mračna bodů ve všech prostředí



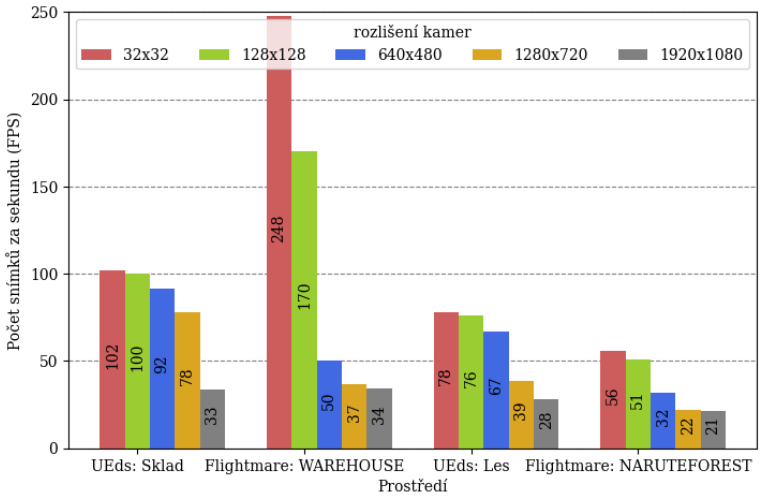
Obrázek 5.6. Srovnání rychlosti odezvy LiDARu v počtu snímků za sekundu pro různý počet bodů v mračnu bodů (point cloud) ve všech prostředí při rostoucím počtu dronů

5.2.3 Porovnání se simulatorem Flightmare

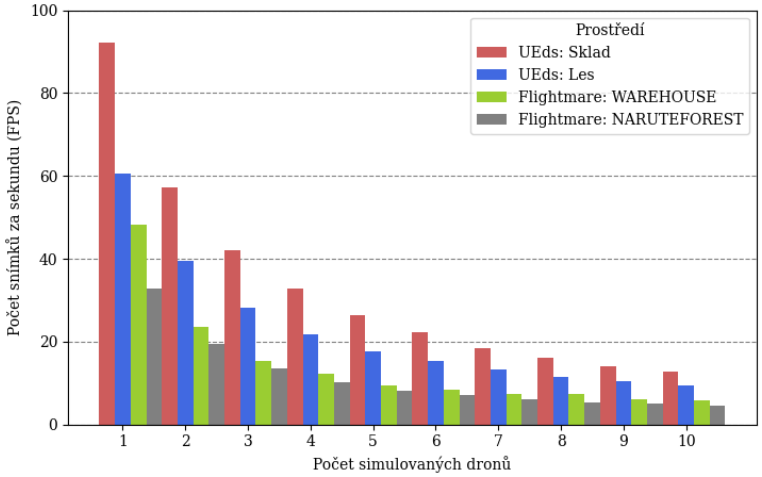
Na závěr porovnáme naše řešení s konkurenčním simulátorem *Flightmare*, který má podobnou architekturu a nabízí také prostředí skladu a lesa. Navíc se jednoduše instaluje díky podrobné dokumentaci[1]. Testovali jsme pouze výkon simulátoru při získávání senzorických dat ze simulované kamery, protože plnohodnotný senzor LiDAR *Flightmare* nenabízí. Umožňuje pouze extrakci prostředí reprezentované jako mračno bodů. Scénáře testů byly stejné jako v Kapitole 5.2.1.

Nejdříve jsme otestovali výkon jednoho dronu při variabilním rozlišení kamer. Z výsledku testu na Obrázku 5.7 vidíme, že naše řešení je téměř dvakrát výkonnější při vyšším rozlišení kamery, které je důležité pro metody počítačového vidění. Při testu s více drony (viz Obrázek 5.8) s fixně nastaveným rozlišení kamer na 640x480 pixelů náš systém také dosahuje dvojnásobného výkonu. Závěrem můžeme říci, že naše řešení je plně konkurenceschopné v porovnání se simulátorem *Flightmare*.

[1] `https://flightmare.readthedocs.io/en/latest/first_steps/envs_and_navigation.html`



Obrázek 5.7. Srovnání rychlosti odezvy RGB kamer simulátorů UEds a Flightmare při různém rozlišení kamer



Obrázek 5.8. Srovnání rychlosti odezvy RGB kamer s rozlišením 640x480 simulátorů UEds a Flightmare při simulaci více dronů

Kapitola 6
Závěr

V úvodu práce jsme se seznámili s problematikou simulátorů používaných při vývoji bezpilotních dronů neboli Unmanned Aerial Vehicle (UAV). Většina dostupných fotorealistických simulátorů neumožňuje simulace autonomních dronů, ale zaměřují se na učení metod umělé inteligence jako Reinforcement Learning nebo Deep Learning, případně poskytují fyzikální model dronu a řízení pouze na bázi aktuátorů.

Proto v rámci bakalářské práce bylo vytvořeno spojení simulace fyziky dronů skupiny MRS se simulátorem dronů v herním enginu Unreal Engine 5 (UEds), což umožňuje simulaci autonomního letu, prozkoumávání a navigaci v neznámých realistických prostředích. Byl vyřešen proces transformace souřadnic mezi rozdílnými souřadnicovými systémy simulátorů. Spojení je realizováno jako rozšíření fyzikální simulace v ROSu a zajišťuje cyklické získávání dat ze senzorů simulovaných v UEds na základě aktuální polohy dronů z fyzikální simulace. Získaná data jsou předány do informačních kanálů (ROS Topic) pro další zpracování nadřazených algoritmů autonomního letu a vizualizaci. Spojení podporuje paralelní simulaci jednoho i více dronů vybavenými senzory jako LiDAR a kamera. Pomocí experimentů byla potvrzena schopnost systému poskytovat realistické rychlosti přenosu dat ze senzorů.

Vytvořili jsme tři různá prostředí: sklad, les a jeskyně za účelem vývoje a testování metod řízení autonomních dronů. První prostředí bylo vytvořeno ručně a díky pečlivému výběru modelů poskytuje věrný a detailní model skladu. Pro lesní a jeskynní prostředí jsme se rozhodli použít procedurální generování, které je efektivnější a umožňuje vytváření rozsáhlých a komplexních prostředí. Vytvoření takovýchto prostředí pouze manuálně by stálo značné úsilí a čas. Tyto prostředí jsou implementovány v UE modulech a jsou jednoduše přenositelné mezi projekty.

Lesní prostředí bylo generováno pomocí Perlinova šumu a komponenty Procedural Content Generation v Unreal Engine 5, což umožnilo vytváření rozsáhlých a fotorealisticky vypadajících lesů. Byl vytvořen nástroj pro tvorbu nekonečného lesního prostředí, které se dynamicky vytváří v závislosti na pohybu dronů.

Jeskynní prostředí bylo vytvořeno kombinací algoritmů Perlin Worms a Marching Cubes. Tyto algoritmy umožnily generování komplexních struktur jeskyní s detaily, jako jsou krápníky a průzkumná světla. Přestože výsledkem není zcela fotorealistické prostředí může být použito pro simulace záchranných operací a autonomní navigace v náročných neznámých podmínkách.

Do budoucna se nabízí možnost dále rozšířit a zdokonalit procedurální generování lesního prostředí, například přidáním náhodných mýtin, lesních školek, polomů stromů a podobně. Díky rozdělení světa na části a použití PCG komponenty jsou rozšíření snadno implementovatelné.

Literatura

[1] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Muller, Vladlen Koltun a Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*. 2023, vol. 620 (no. 7976), pp. 982–987.

[2] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun a Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*. 2022, vol. 7 (no. 62), p. eabk2822.

[3] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio a Davide Scaramuzza. *Flightmare: A flexible quadrotor simulator*. In: *Conference on Robot Learning*. 2021. pp. 1147–1157.

[4] Shital Shah, Debadeepta Dey, Chris Lovett a Ashish Kapoor. *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. In: *Field and Service Robotics. eprint: arXiv:1705.05065*. 2017.
`https://arxiv.org/abs/1705.05065`.

[5] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou a Sertac Karaman. *FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality*. In: *arXiv preprint:1905.11377*. 2019.
`http://arxiv.org/abs/1905.11377`.

[6] Open Robotics. *Gazebo*.
`https://gazebosim.org`. [Online; citováno 12.12.2023] .

[7] Unity Technologies. *Unity Real-Time Development Platform*.
`https://unity.com/`. [Online; citováno 23.5.2024] .

[8] Epic Games. *The most powerful real-time 3D creation tool - Unreal Engine*.
`https://www.unrealengine.com/en-US`. [Online; citováno 23.5.2024] .

[9] Multi-robot Systems Group. *Multi-robot Systems*.
`https://mrs.felk.cvut.cz`. [Online; citováno 4.12.2023] .

[10] Tomas Baca, Matej Petrlik, Matous Vrba, Vojtech Spurny, Robert Penicka, Daniel Hert a Martin Saska. The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles. *Journal of Intelligent & Robotic Systems*. 2021, vol. 102 (no. 1), pp. 26.

[11] Daniel Hert, Tomas Baca, Pavel Petracek, Vit Kratky, Robert Penicka, Vojtech Spurny, Matej Petrlik, Matous Vrba, David Zaitlik, Pavel Stoudek a others. MRS drone: A modular platform for real-world deployment of aerial multi-robot systems. *Journal of Intelligent & Robotic Systems*. 2023, vol. 108 (no. 4), pp. 64.

[12] Jack Collins, Shelvin Chand, Anthony Vanderkop a David Howard. *A Review of Physics Simulators for Robotic Applications*. In: *IEEE Access*. 2021. pp. 51416-51431.
`https://doi.org/10.1109/ACCESS.2021.3068769`.

[13] The MathWorks Inc. *Unreal Engine Simulation for Unmanned Aerial Vehicles*. https://www.mathworks.com/help/uav/ug/3d-simulation-for-unmanned-aerial-vehicles.html. [Online; citováno 14.12.2023] .

[14] Jakub Jirkal. *Drone Simulation Using Unreal Engine*. Bakalářská práce. FEL, ČVUT v Praze. 2023. https://dspace.cvut.cz/handle/10467/109185.

[15] Multi-robot Systems Group. *MRS UAV open-source research platform*. https://mrs.felk.cvut.cz/system. [Online; citováno 2.12.2023] .

[16] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng a others. *ROS: an open-source Robot Operating System*. In: *IEEE ICRA workshop on open source software*. vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[17] Robot Operating System. *3D visualization tool for ROS - Rviz*. http://wiki.ros.org/rviz. [Online; citováno 23.5.2024] .

[18] Epic Games. *Unreal Engine 5 - Marketplace*. https://www.unrealengine.com/marketplace/en-US/. [Online; citováno 13.11.2023] .

[19] Univerzity of Zurich. *High-speed AI Drone Overtakes World-Champion Drone Racers*. https://www.news.uzh.ch/en/articles/media/2023/Drone-race.html. [Online; citováno 16.12.2023] .

[20] Destrolas. *Perlin and Simplex Noise*. 24. 10. 2018. https://leatherbee.org/index.php/2018/10/24/perlin-and-simplex-noise/. [Online; citováno 14.4.2024] .

[21] Flafla2. *Understanding Perlin Noise*. 9. 8. 2014. https://adrianb.io/2014/08/09/perlinnoise.html. [Online; citováno 14.4.2024] .

[22] Epic Games. *Procedural Content Generation - Unreal Engine Documentation*. https://docs.unrealengine.com/5.2/en-US/procedural-content-generation-overview/. [Online; citováno 18.12.2023] .

[23] Epic Games. *ProceduralMeshComponent - Unreal Engine Documentation*. https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Plugins/ProceduralMeshComponent?application_version=5.3. [Online; citováno 24.3.2024] .

[24] Epic Games. *UKismetProceduralMeshLibrary::CalculateTangent() - Unreal Engine Documentation*. https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Plugins/ProceduralMeshComponent/UKismetProceduralMeshLibrary/CalculateTangent-?application_version=5.3. [Online; citováno 2.4.2024] .

[25] DARPA. *Subterranean (SubT) Challenge*. https://www.darpa.mil/program/darpa-subterranean-challenge. [Online; citováno 7.1.2024] .

[26] Jason Bevins. *Perlin Worms*. 2005. https://libnoise.sourceforge.net/examples/worms/. [Online; citováno 23.4.2024] .

[27] Squirrel Eiserloh. *Math for Game Programmers: Digging with Perlin Worms - GDC Vault*. 2018.

https://www.gdcvault.com/play/1025191/Math-for-Game-Programmers-Digging. [Online; citováno 23.4.2024] .

[28] Paul Bourke. *Polygonising a scalar field*. 1994. https://paulbourke.net/geometry/polygonise/. [Online; citováno 23.3.2024] .

[29] Wikipedia. *Marching cubes*. https://en.wikipedia.org/wiki/Marching_cubes. [Online; citováno 23.3.2024] .

[30] *Tisícovky Čech, Moravy a Slezska*. http://www.tisicovky.cz/cs/hory/sumava/homole-hlv388/. [Online; citováno 13.5.2024] .

[31] Wikipedia. *Koněpruské jeskyně*. https://cs.wikipedia.org/wiki/Kon%C4%9Bprusk%C3%A9_jeskyn%C4%9B. [Online; citováno 13.5.2024] .

Příloha A
Seznam přiložených souborů

Modely získané z UE Marketplace lze používat bezplatně, ale nepřikládáme je, protože zabírají příliš velké místo na disku. Proto zveřejňujeme pouze zdrojové soubory a použité rendery v této práci, které jsou přiloženy v archivu `sources.zip`. Obsah jednotlivých složek je uveden níže. Plná verze simulátoru je uložena ve verzovacím systému skupiny MRS.

`mrsMultirotorSimulator`	ROS balíček MRS Simulace doplněný o komunikaci s UE5, `DroneControllerRos.h`
`ProceduralGenProject`	Zdrojové kódy UE modulů implementující procedurální generaci prostředí
`Photos`	Rendery použité v práci
`BenchmarksGraphs`	Naměřená data, grafy a jejich zpracování