FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Návrh a implementace simulátoru ptačího letu ve virtuálním 3D prostředí
Student:	Jakub Schinko
Vedoucí:	doc. Ing. Mgr. Petr Klán, CSc.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

1. Seznamte se podrobně s odbornou literaturou a modely dynamiky letu ptáků.
2. Prostudujte a naučte se pracovat v systému virtuální reality Unreal.
3. Navrhněte základní organizaci a strukturu 3D scény pro implementaci simulátoru jako horské prostředí s potoky, lesem a hnízdy ptáků na stromech.
4. Vytvořte 3D objekty těchto scén a implementujte scény.
5. Připravte a realizujte sadu rozšiřujících schopností virtuálních ptáků zahrnující chytání nebo pouštění předmětů a aplikace částicových systémů.
6. Prostudujte a implementujte do scény simulátoru vybraný hejnový algoritmus včetně řešení případných kolizí.
7. Testujte simulátor z pohledu uživatele.
8. Optimalizujte simulátor podle výsledku testů.
9. Pokuste se stanovit míru "imerzivity" virtuálního simulátoru a porovnejte ji s nevirtuální variantou.
10. Replikujte mechaniky na více hráčů a vytvořte způsob, jak hrát přes webovou síť.
11. Nakonfigurujte umělou inteligenci v simulátoru tak, aby fungovala po webové síti.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 28. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Návrh a implementace simulátoru ptačího letu ve virtuálním 3D prostředí

Jakub Schinko

Katedra ...(Katedra softwarového inženýrství)
Vedoucí práce: doc. Ing. Mgr. Petr Klán, CSc.

13. května 2021

Poděkování

Velice děkuji svému vedoucímu doc. Ing. Mgr. Petru Klánovi, CSc. za rady, korektury a pomoc při tvorbě této práce

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel licenční smlouvu o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisu.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Schinko, Jakub. *0.0.0*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato práce se zabývá návrhem a následnou implementací mechanik, které slouží jako základ hry na virtuální realitu, napsané v enginu Unreal Engine. Nejprve práce pojednává o aktuálním vývoji technologie virtuální reality a způsobu vývoje tohoto odvětví. V práci se dále rozebírá implementace jednotlivých herních mechanik. Mezi jednotlivé implementace patří pohyb hráče ve VR simulující ptačí let, systém herní AI zabývající se simulací ptáků v hejnech a tvorba 3D scény, ve které se hráč nachází. Dále práce obsahuje testování jednotlivých mechanik a shrnutí budoucího vývoje této práce.

Klíčová slova Unreal Engine, virtuální realita, flocking algoritmus, ptačí hejna, pohyb ve VR, level design

Abstract

This thesis pursues design and implementation of mechanics, which serve as a building block of a virtual reality game written in Unreal Engine. At firt, thesis describes current development of VR technology and prognosis referring its future impact. The thesis then describes individual VR mechanics. Described mechanics are bird flight as VR locomotion mechanic, bird flocking system as main game AI and algorithms referring to level design in VR. At last, thesis contains findings that were observed during testing.

Keywords Unreal Engine, virtual reality, flocking algorithm, bird flocks, VR locomotion, level design

Obsah

Seznam obrázků

Úvod

Virtuální realita je technologie umožňující uživateli se ocitnout v simulovaném prostředí, s kterým je možné interagovat. Přestože začátky této technologie vznikly už v 70. letech 20. století, teprve za posledních pár let se tato technologie rozšířila mezi širší povědomí. Podle statistických údajů se počet prodaných kusů za rok 2020 pohybuje kolem 68 milionů. Výrobců zařízení pro virtuální realitu existuje mnoho. Mezi nejznámější patří např. Oculus, HTC, anebo Playstation. Zařízení se rozlišují výkonem, cenou, anebo nutností využití technologie bez cizího hardwaru. Nejčastějším způsobem, jak vyvíjet software pro tuto technologii je využití platforem určených pro tento účel. Software pro virtuální realitu lze vyvíjet např. v Unity, Unreal Enginu, anebo v platformě Neos. Pro tuto práci byla zvolena platforma Unreal Engine. Práce pojednává o vývoji software pro VR a jejím výstupem je implementace mechanik, které slouží jako základ hry na virtuální realitu.

Kapitola 1

Cíl práce

Hlavním cílem této práce je vytvořit herní prvky, které budou sloužit jako základ hry na virtuální realitu. Práce je rozdělená na 6 částí. V první části budou definovány hlavní úskalí, výzvy a technologie spojené s vývojem ve virtuální realitě. V další části bude rozebrána implementace pohybu letu ve virtuální realitě. Následně bude popsána implementace simulace ptáků v hejnech jako hlavní herní umělá inteligence. Dalším bodem bude rozepsání postupů při tvorbě 3D scény a popis jednotlivých nástrojů. Následovat bude popis herních mechanik, které se budou zabývat kontrolou hejn z hráčské perspektivy. V neposlední řade budou popsány dodatečně implementované mechaniky. Před samotným shrnutím bude sepsán proces testování a ladění software. V závěru bude shrnut vývoj ve virtuální realitě v obecné rovině a jaký způsobem se bude uchylovat směr této technologie.

Kapitola 2

Technologie a úskalí spojené s virtuální realitou

Vývoj na virtuální realitu je v mnoha věcech odlišný od klasického vývoje 3D software. V této kapitole budou popsány následující body:

- VR sickness - Kapitola se zabývá problematikou VR sickness.
- Důležitost optimalizace - Kapitola popisuje jakou roli hraje optimalizace ve VR.
- Unreal Engine - Kapitola popisuje použitý nástroj pro tvorbu software.

2.1 VR Sickness

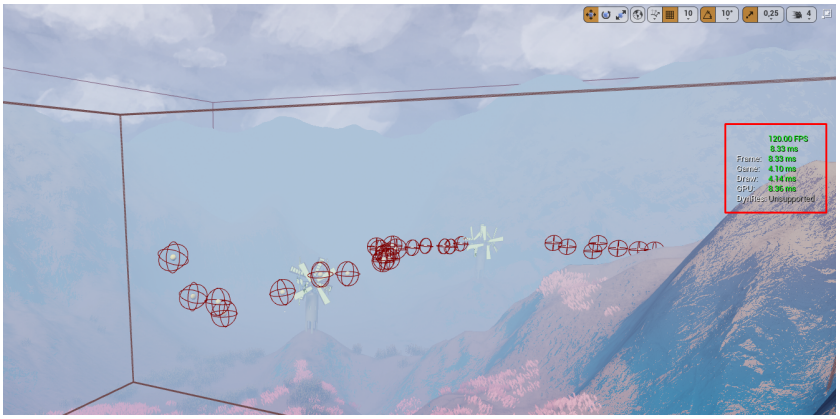
Při snaze simulovat realitu nasazením upravených monitorů na hlavu hráče, uživatelé budou po nějakém čase v určitém nepohodlí. Tento jev nazýváme VR sickness. Identifikace hlavních úskalí a problémů spojených s virtuální realitou je esenciální k tvorbě software v této disciplíně. V první řadě jsou zde problémy spojené se stavem dnešního hardwaru v této oblasti. Virtuální realita simuluje prostředí pomocí dvěma stereoskopickým displejům umístěnými před očima. VR zařízení jednotlivých výrobců se rozlišují rozlišením, počtem snímků za sekundu, anebo zorným polem. Čím lépe jsou tyto technické požadavky vyřešeny, tím menší má VR sickness vliv. Krom hardwarových existují softwarová omezení. V rámci vývoje je možné simulovat situace, které jsou lidem nepříjemné, jelikož v normální realitě by nemohli nastat. Jelikož součást práce je implementace pohybu ve virtuální realitě, pro co nejlepší uživatelský zážitek je potřeba zredukovat efekt VR sickness.

Pokud hráč stojí na místě, ale ve virtuální realitě se pohybuje určitou rychlostí, hráč vnímá pohyb, ale tělo necítí reálnou akceleraci. Lidský mozek je tak zmatený, což částečně způsobuje nepohodlí ve VR. Jeden ze způsobů, jak

vyřešit tuto problematiku, je využití pohybových hardware simulátorů, které dokáží simulovat vnímání reálné fyziky. Pokud ovšem chceme vytvořit herní zážitek pro většinu uživatelů, musíme realizovat pohyb, který je dostupný na VR brýle a dva ovladače, které patří mezi standardní vybavení většiny uživatelů. Pokud se při vývoji připustí absence pocitu akcelerace, je důležité dodržovat další pravidla. Jedno z těchto pravidel je neměnit rotaci kamery hráče. Tento jev je pro hráče nepříjemný a motá se mu z toho hlava. Další způsob, jakým zvýšíme imerzivitu a snížíme VR sickness je donutit hráče být v kontrole pohybu. Pokud hráč dokáže předpovídat daný pohyb ve VR, zvýší se imerzivita a VR sickness se méně projevuje. Stejně jako lidem sedící v autě na straně spolujezdce se dělá nevolno při čtení, pohyb ve virtuální realitě, který hráč nedokáže předpovídat, je pro lidský organismus nepříjemný.

2.2 Důležitost optimalizace

V rámci vývoje na VR je velice důležité myslet na optimalizaci. Jelikož většina dnešních headsetů běží maximálně při 90 snímcích za sekundu, dovolí nám to při vývoji ztratit maximálně 20-30 FPS každou delta sekundu, podotkneme-li, že pokud se aplikace začne zobrazovat při méně jak 60 FPS, stane se nehratelnou. Zároveň se dá tvrdit, že výsledná aplikace je zdraví nebezpečná. Na rozdíl od tradičních 3D aplikací, u kterých si vývojář může dovolit ztratit až 75 FPS, u virtuální reality je to okolo 20 FPS, beru-li v potaz horní nastavenou hranici Unreal Enginu 120 FPS a faktu, že tradiční hry se dají hrát i při 45 FPS. Pokud chci například vytvořit algoritmus pro pohyb ptačích hejn, počet vytvořených ptáků v běhu programu se bude odvíjet od maximálního snížení dovoleného FPS.



Obrázek 2.1: Statistické nástroje využité v rámci optimalizace.

Během vývoje byli použity optimalizační nástroje stat fps a stat unit (viz obr. 2.1). Pomocí těchto nástrojů jsem detekoval pokles FPS, náročné vykres-

lování geometrie a nebo jiné bottlenecky.

2.2.1 Optimalizace v 3D geometrii

V rámci této bakalářské práce byl využit nástroj Unreal Enginu, kterému se říká instancované meshe. Pokud v enginu do 3D scény přidáváme více totožných objektů, hodí se nám daný 3D objekt tzv. instancovat. To znamená, že pro 3D objekt existuje pouze jeden vizuál, není možné přiřadit negativní škálu dané instanci, s jejich kolizemi se moc dobře nepracuje, je obtížné je animovat, ale pokud je využíváme, zvýšíme počet modelů v 3D scéně při stejném snížení FPS.

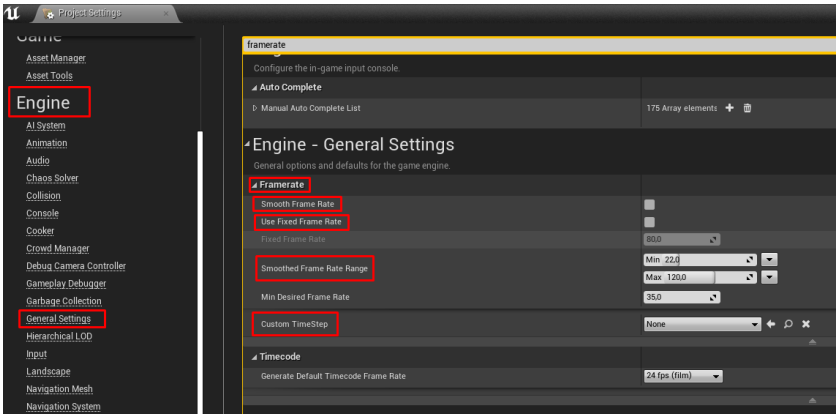
Unreal Engine obsahuje spoustu vysokoúrovňových nástrojů pro tvorbu prostředí. Je potřeba alespoň rámcově vědět, jakým způsobem fungují, abychom mohli lépe optimalizovat výsledný software. Jako jedno z hlavních pravidel bych zmínil potřebu nevykreslovat věci, které uživatel nevnímá. Unreal Engine spoustu práce vývojáři ulehčí, přesto je potřeba dodržovat tyto pravidla v rámci tvorby vlastních systémů. Například ve společnosti Epic Games, tvůrců Unreal Enginu, existuje zásada, že každý člen vývojářského týmu, i když se jedná o netechnickou pozici, potřebuje znát základní nástroje použité k detekci úskalí v optimalizaci. Předchází se tím tzv. Janitor Effectu.

2.2.2 Optimalizace v síti

Samotná optimalizace je důležitá i z hlediska hraní po síti. Díky nástrojům Unreal Enginu sice není potřeba řešit práci se sockety, posílání packetů, serializaci dat, jejich kódování nebo jakékoliv routování. Přesto je potřeba počítat s výpadky sítě, malým připojením hráče, verifikací dat kvůli podvádění, nebo predikcí pohybu jiných hráčů. Vždy je potřeba zvážit požadavky software a podle nich implementovat algoritmy. Pokud máme například kompetitivní online souboj, klademe velký důraz na co nejpřesnější lokaci v běhu programu ostatních hráčů. Pokud hrajeme šachy, nemusíme aktualizovat polohu v běhu programu vůbec.

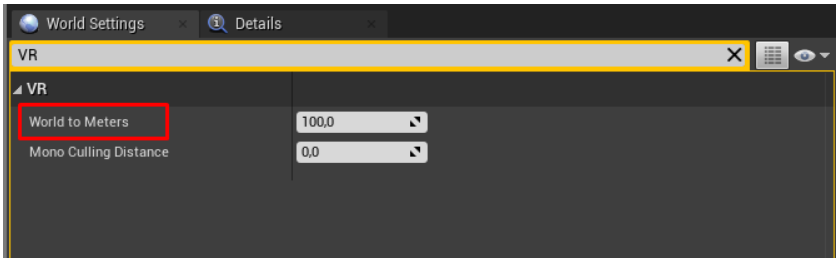
2.2.3 Nastavení enginu pro VR

Aby software pro virtuální realitu vykazoval nejlepší výsledky, je potřeba změnit určitá nastavení v enginu.



Obrázek 2.2: Nastavení fixní snímkové frekvence

U standardních 3D aplikací engine automaticky upravuje snímkovou frekvenci, aby vylepšil herní zážitek při poklesu FPS. Jelikož při vývoji na VR máme větší nároky na optimalizaci, k poklesu FPS by nikdy nemělo dojít. Proto používáme v rámci nastavení Unreal enginu fixní snímkovou frekvenci (viz obr. 2.2).



Obrázek 2.3: Nastavení škály ve virtuální realitě

Při tvorbě software na VR je potřeba reprezentovat jednu unreal jednotku jako jeden centimetr (viz obr. 2.3).

2.3 Unreal Engine

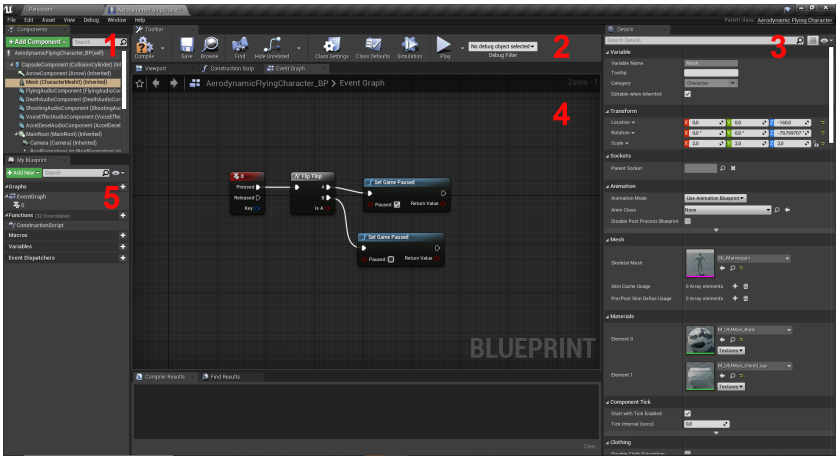
Pro vytvoření této práce byl zvolen nástroj Unreal Engine. Mezi některé výhody tohoto enginu patří například jednoduchá manipulace s optimalizovaným 3D prostředím, kolizní systém, množství podporovaných platforem, škálovatelný vývoj her po síti, integrace PhysX pro simulaci fyziky a mnoho dalších. Využití PhysX fyzikálního enginu se použilo v rámci implementace pohybu letu hráče ve VR. Tato kapitola se zabývá základními částmi enginu, především jeho nástroji, které byli použiti v této práci.

Obrázek 2.4: Základní rozhraní Unreal Enginu

1. Place Actors Window - Nabídka defaultních objektů Unreal Enginu.
2. Toolbar - Panel nástrojů.
3. World Outliner - Seznam objektů v 3D scéně.
4. 3D scéna - Prostředí reprezentující simulaci.
5. World Settings - Nastavení 3D scény.
6. Content Browser - Nabídka vlastních objektů a blueprintů.

2.3.1 Programování

Pokud chceme implementovat vlastní funkcionalitu v rámci toho enginu, je potřeba znát jazyk c++. Existuje sice možnost napsat celý software pomocí nástroje vizuálního programování nazývaný Blueprints (viz obr. 2.5), přesto časté použití vizuálního programování vede k problémům s optimalizací a nebo se může stát, že neexistuje možnost, jak daného výsledku docílit jinak než pomocí c++. Unreal Engine je možné buď spustit přes Launcher, anebo si stáhnout zdrojový kód. Kompilace ze zdrojového kódu je nutná, pokud využíváme experimentálních nástrojů Unreal Enginu, anebo chceme například vytvořit dedikovaný server pro multiplayer. Krom znalosti jazyka c++ je také potřeba znát makra a knihovny, abychom byli schopni komunikovat s editorem scén.



Obrázek 2.5: Prostředí blueprintu

1. Components - Seznam komponentů objektu reprezentovaného blueprintem.
2. Toolbar - Panel nástrojů.
3. Details - Detaily zvoleného komponentu.
4. Event Graph - Prostředí vizuálního programování.
5. Proměnné, makra, funkce použité v blueprintu.

2.3.2 Editor scén

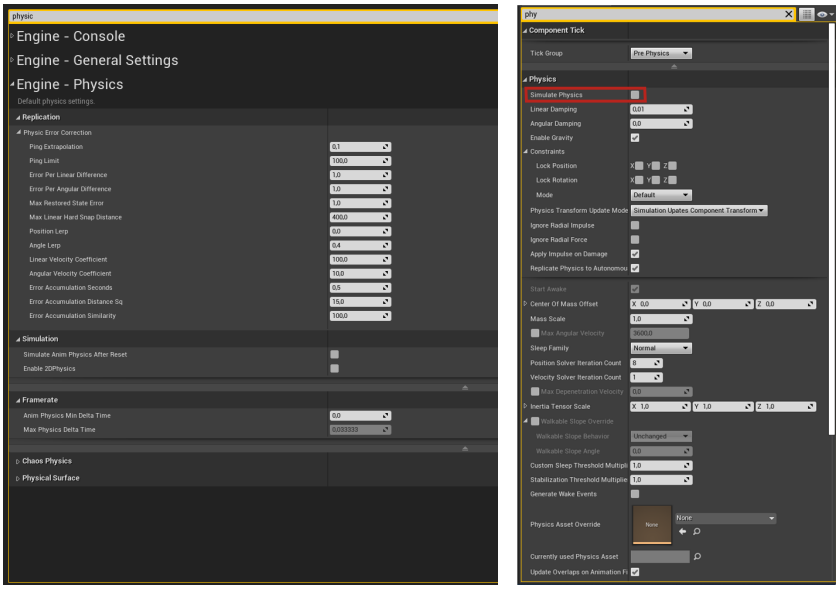
Hlavní výhodou Unreal enginu je editor scén, který nám umožňuje manipulovat s objektama ve světě (viz obr. 2.6). Při vývoji je důležité se seznámit se základními objekty tohoto enginu, s kterými vývojář může interagovat v rámci editoru scén. Pokud chce vývojář přidat další funkcionality, stačí využít dědičnosti těchto základních objektů a implementovat navržené algoritmy.

Obrázek 2.6: Editor scén

1. Nastavení gizma pro změnu lokace.
2. Nastavení gizma pro změnu rotace.
3. Nastavení gizma pro změnu škály.

2.3.3 Fyzika

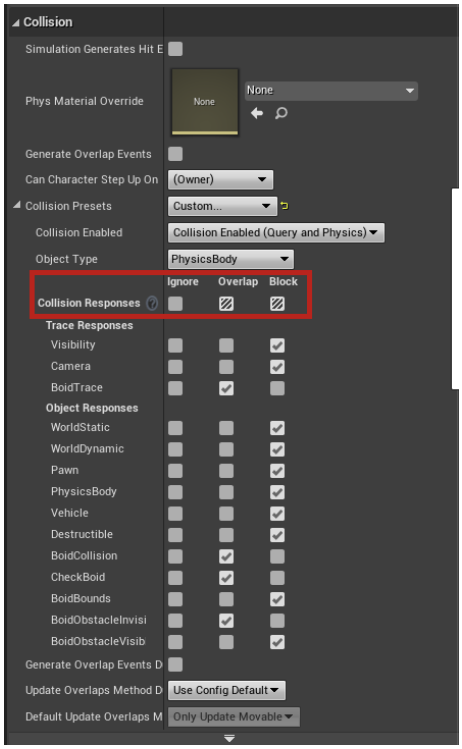
Unreal Engine má sadu nástrojů umožňující ovládání fyziky v simulaci. Stačí objektu nastavit základní fyzikální parametry jako např. hmotnost a poté můžeme v běhu programu přidávat vektory síly, nebo upravovat fyzikální proměnné, jako například vliv gravitace. V rámci této práce jsem využil knihoven pro manipulaci s fyzikou a samotná konfigurace probíhala v rámci uživatelského rozhraní Unreal Enginu. V rámci této práce byli nejprve zpracovány data jednotlivých VR zařízení, mezi které patří dva ovladače a VR brýle. Následně podle získaných dat z těchto zařízení, mezi které patří lokace, rotace, rychlost, a nebo relativní vzdálenosti, byly dopočítány silové vektory, díky kterým fyzikální engine PhysX hýbal s postavou hráče v prostoru.



Obrázek 2.7: Ukázka prostředí sloužící ke konfiguraci fyziky

2.3.4 Kolize

Abychom mohli v simulaci stát na místě, anebo narazit do stěny, je potřeba využít kolizní systém Unreal Enginu. Možnost neimplementovat vlastní kolizní systém je pohodlné, jelikož algoritmy Unreal Enginu jsou v základu velice optimalizované. Pokud je definována kolize u nějakého herního objektu, ať už to je hráč, pták, anebo stěna, musíme nejdříve přiřadit objektu kolizní profil a poté definovat 1 ze 3 možností interakce s dalším objektem jiného kolizního profilu (viz obr. 2.8). Mezi interakce patří ignorovat kolizi, překrytí kolize, blokování kolize. Při nastavování kolizí, vývojář má možnost spouštět jednotlivé události při těchto interakcích. V rámci této práce bylo potřeba např. definovat kolizní profily pro objekty reprezentující ptáky, které mají zůstat v ohraničeném prostoru.



Obrázek 2.8: Ukázka prostředí sloužící ke konfiguraci kolizí

- Ignore - Objekt ignoruje kolize.
- Overlap - Objekt detekuje kolize překryvu.
- Block - Objekt detekuje náraz.

Kapitola 3

Implementace letu ve VR

V rámci této práce byl navržen způsob pohybu ve virtuální realitě simulující let ptáka. Program byl napsaný v c++ pomocí fyzikálního enginu PhysX. Před samotným návrhem algoritmu bylo potřeba spočítat rychlost jednotlivých VR zařízení jako rozdíl poloh po sobě jdoucích snímků, vzájemnou vzdálenost ovladačů a směrové vektory reprezentující směr ovladačů. Pomocí těchto dat byli navrženy algoritmy, jejichž výstupem jsou síly působící na hráče v simulaci. V této kapitole bude detailněji rozepsaná implementace následujících algoritmů:

- Aerodynamický odpor
- Aerodynamický vztlak
- Otáčení v prostoru
- Uhýbání do stran

3.1 Aerodynamický odpor křídel

Algoritmus simuluje hráči pohyb ptačího letu, konkrétně máchání křídly. Křídla v běhu programu jsou reprezentovány hráčovými ovladači. Když hráč ve virtuální realitě máchá rukama, jsou tím získány údaje o rychlosti obou ovladačů v prostoru. Pomocí daných rychlostí a směrů ovladačů bude spočítána výsledná síla působící na hráče. Algoritmus je navržený podle fyzikálního vzorce aerodynamického odporu (viz vzorec 3.1), kdy násobíme kvadrát rychlosti, hustotu vzduchu, účinnou plochu křídla a součinitel odporu. Implementace je na rozdíl od původního vzorce zjednodušena na součin kvadrátu rychlosti, zjednodušeném výpočtu účinné plochy a konstanty (viz pseudokód 7). Aerodynamický odpor je dále rozdělen na globální a lokální složku, kdy lokální složka ignoruje pohyb hráče v prostoru. Výsledná síla je součet těchto dvou složek.

$$F = \frac{1}{2} C_x \rho S v^2 \tag{3.1}$$

- C_x - součinitel odporu
- ρ - hustota vzduchu
- S - účinná plocha
- v^2 - kvadrát rychlosti

Fyzikální vzorec pro výpočet aerodynamického odporu

```
float LokalniMagnitudaOdporu = VelikostRelativniRychlosti * VelikostRelativniRychlosti * HodnotaUcinnePlochy * LokalniKonstanta ;
float GlobalniMagnitudaOdporu = VelikostGlobalniRychlosti * VelikostGlobalniRychlosti * HodnotaUcinnePlochy * GlobalniKonstanta ;
FVector LokalniSmerOdporu = VelikostRelativniRychlosti.Normalized();
FVector LokalniSilaOdporu = LokalniSmerOdporu * LokalniMagnitudaOdporu;
FVector GlobalniSmerOdporu = VelikostGLobalniRychlosti.Normalized();
FVector GlobalniSilaOdporu = GlobalniSmerOdporu * GlobalniMagnitudaOdporu;
FVector FinalniSilaOdporu = GlobalniSilaOdporu + LokalniSilaOdporu;
```

Pseudokód výpočtu aerodynamického odporu použitý při implementaci

3.1.1 Kvadrát rychlosti

Aktuální rychlost křídla spočítáme jako rozdíl lokací křídla v dvou následujících snímcích. Pokud od této hodnoty odečteme rychlost hráče, získáme aktuální rychlost v lokálních souřadnicích.

3.1.2 Výpočet účinné plochy

V rámci simulace výpočet účinné plochy pouze připomíná hodnoty získané v reálném fyzikálním světě. Účinná plocha křídla ve vzorci pro aerodynamický odpor se projevuje nakláněním křídla ve směru pohybu. Účinná plocha je tedy reprezentována pomocí tohoto naklonění. Výstupní hodnota je transformovaný úhel vektoru směru pohybu křídla a směrového vektoru samotného křídla.

3.1.3 Součinitel odporu

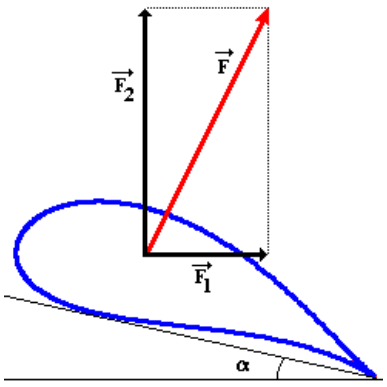
Součinitel odporu, který reprezentuje tvar křídla se projevuje v reálné fyzice tím, že např. při máchnutí směrem nahoru bude získána menší výsledná síla než při máchnutí směrem dolů (nebo naopak). Tento fakt algoritmus zohlednil opět zjednodušením, kdy síla se projevuje pouze při máchání rukama dolů.

3.2 Aerodynamický vztlak

Dalším algoritmem přispívající k simulaci ptačího letu je výpočet aerodynamického vztlaku. Jako podklad byl použit fyzikální vzorec pro aerodynamický vztlak (viz vzorec 3.2). Díky tomuto algoritmu má možnost hráč tzv. plachtit ve vzduchu pomocí svých virtuálních křídel. Vstupní data pro výpočet získáme opět z pozice a rotace hráčových ovladačů. Pokud hráč má rozpažené ruce a jeho palce směřují dopředu, získáme úhel mezi směrovým vektorem reprezentující průměr směrů palců u obou ovladačů a směrem pohybu. Magnituda síly působící na hráče je součin tohoto úhlu, kvadrátu rychlosti hráče a libovolně zvoleného parametru. Směr této síly je vektorový součin vektoru reprezentující směr pohybu hráče a vektoru, který získáme odečtením souřadnic obou ovladačů. Strana směru je poté určena díky předem spočítanému úhlu mezi směrem pohybu hráče a směrem ovladačů.

$$F = \frac{1}{2} C_y \rho S v^2 \tag{3.2}$$

- C_y - součinitel vztlaku
- ρ - hustota vzduchu
- S - účinná plocha
- v^2 - kvadrát rychlosti



Obrázek 3.1: Směry sil odporu a vztlaku

- F_1 - Směr aerodynamického odporu
- F_2 - Směr aerodynamického vztlaku

Pseudokód výpočtu aerodynamického vztlaku použitý při implementaci

```
FVector SmerVztlaku = FVector::CrossProduct(SmerPohybu, VektorRozdilOvladacu);
SmerVztlaku.Normalize()
float MagnitudaVztlaku = RozpetiKridel * KonstantaVztlaku * VelikostRychlostiHrace * VelikostRychlostiHrace * HodnotaUcinnePlochy;
FVector VztlakovaSila = SmerVztlaku * MagnitudaVztlaku;
```

3.2.1 Rozpětí křídel

V rámci ladění tohoto pohybu bylo potřeba zjistit aktuální rozpětí křídel hráče, podle kterého se odvíjí magnituda finální síly vztlaku. Pokud má hráč ruce rozpažené, síla se projevuje standardně podle vzorce. Naopak při menším rozpažení hráče se vztlak projevuje méně. Tento vztah je vyjádřen jako podíl aktuálního rozpětí a maximálního rozpětí. Vyjádření této hodnoty je v podobě funkce reprezentovanou křivkou, aby bylo možné při testování odladit pohyb.

3.2.2 Ladění plachtění hráče

V rámci testování byl nejprve odladěn pohyb máchání křídly. Následně k tomuto pohybu byly přidány další typy algoritmů, například vzorec pro výpočet letu simulující plachtění. Během testování kombinace těchto dvou pohybů se ukázalo, že hráč přestane mít kontrolu nad máchání křídly. Program byl tedy implementován tak, aby se plachtění projevovalo pouze, kdy hráč nemáchá křídly.

3.3 Otáčení v prostoru

Pro nejoptimálnější uživatelský zážitek z hlediska VR sickness, rotace kamery ve virtuálním světě vždy kopíruje pohled hráče v reálném světě. Algoritmus otáčení (viz pseudokód 7) je tedy nastaven tak, aby směr hráče reprezentoval směr pohybu postavy ovládané ve VR. Abychom toho docílili, nejprve zjistíme vstupní vektor. Pokud máme rozpažené ruce a naše palce směřují před hráče, vstupní vektor je definovaný jako průměr směrů palců. Tento vektor poté promítneme do vodorovné plochy a porovnáváme úhel mezi tímto vektorem a aktuálním směrem pohybu postavy ve VR promítnuté do stejné plochy. Pokud úhel přesáhne určitou hranici, začne na hráče působit síla reprezentovaná jako výsledek vektorového součinu směru hráče a světové osy Z v požadovaném směru. Poté co hráčův aktuální směr se opět dostane daný úhel, síla přestane působit.

```
if AktuálniUhel >= HraniceUhlu then
  SmerSily = FVector::CrossProduct(RychlostHrace, SvetovaSouradniceZ);
  if FVector::DotProduct(RychlostHrace, SmerOvladacu) > 0 then
    SmerSily *= -1;
  end if
  FVector VyslednaSila = SmerSily * RotacniKonstanta;
end if
```

3.4 Uhýbání do stran

Další pohyb, kterým hráč disponuje, je úhyb do stran. Algoritmus (viz pseudokód 6) pracuje s rozdílem výšek obou ovladačů ve světových souřadnicích. Na hráče poté působí stejná síla jako v algoritmu pro otáčení v prostoru v požadovaném směru. Hráč tak může uhýbat do stran při letu, aby se mohl vyhýbat překážkám.

```
FVector SmerSily = FVector::CrossProduct(SmerOvladacu, SvetovaSouradniceZ);
float VyskovyRozdilOvladacu = LokacePravehoOvladace.Z - LokaceLevehoOvladace.Z;
VyskovyRozdilOvladacu = FMath::Clamp(VyskovyRozdilOvladacu, -120.0f, 120.0f);
VyskovyRozdilOvladacu = VyskovyRozdilOvladacu / 2;
float MagnitudaSily = VyskovyRozdilOvladacu * Konstanta;
FVector VyslednaSila = MagnitudaSily * SmerSily;
```

Kapitola 4

Hejnový algoritmus jako hlavní prvek herní AI

V roce 1986 Craig Reynolds publikoval v rámci ACM SIGGRAPH konference práci popisující program simulující pohyb ptáků v hejnech. Podle jeho práce byly implementové základní algoritmy koheze, separace a seřazení. V rámci implementace byl definován typ pohybu, kde v každém snímku simulace se přičítá výsledný vektor těchto vlastností jako akcelerace k momentální rychlosti. Jelikož některé situace nebo pohyby jsou spjaté s detekováním neviditelných stěn, bylo potřeba vytvořit sadu kolizních profilů s kterýma se v době běhu programu mohlo manipulovat. V rámci této kapitoly bude popsána implementace jednotlivých částí tohoto systému.

4.1 Reprezentace v Unreal Enginu

Každý člen hejna je reprezentovaný jako jeden objekt nazývaný BoidActor. Předtím samotnou implementací pohybu bylo potřeba vyřešit grafickou reprezentaci daného ptáka. V rámci enginu je možné každému objektu přiřadit 3D geometrii nazývanou Mesh. K dané geometrii poté můžeme pomocí nástrojů Unreal Enginu přiřadit jednotlivé animace, které po dané konfiguraci mohou přecházet mezi sebou. V rámci implementace bychom využili komponentu, která se jmenuje skeletal mesh. Jedná se o 3D model připravený v 3D modelovacím programu (např. v Blenderu), který obsahuje jak geometrii, tak kostru. Hýbáním jednotlivými kostmi vytvoříme animace, které následně importujeme do Unreal Enginu. Pokud by každý člen hejna používal tuto komponentu, konečné vykreslení bude z hlediska optimalizace náročné.

Z tohoto důvodu jsem vytvořil systém, v kterém jednotlivé objekty vykreslují pouze instance stejného 3D objektu, čímž jsem zvýšil horní hranici počtu vytvořených ptáků v běhu programu. Tento způsob měl určité nevýhody mezi které patří například absence nástrojů interpolování animací v rámci Unreal

Enginu. Samotné animace museli být exportované jako textury, které v rámci Unreal Enginu byly použity v shaderu. Aby každá instance měla rozdílnou animaci v běhu programu, bylo potřeba předat unikátní data přímo do shaderu, který poté řídil animace jednotlivých instancí. Tato metoda byla náročnější na implementaci, na druhou stranu se zlepšila optimalizace, která je ve virtuální realitě klíčová.

4.2 Výpočet hejnového pohybu

Každý objekt ptáka vlastní kolizi ve tvaru koule, která zaznamenává ostatní ptáky v dosahu. Každý pták má pole těchto sousedů, podle kterých upravuje svůj pohyb.

4.2.1 Seřazení

První algoritmus, který upravuje pohyb ptáka je seřazení. Algoritmus počítá průměrný směr ostatních ptáků v dosahu a následně tento směr přičítá k aktuální rychlosti jako akceleraci (viz pseudokód 6).

```
FVector Steering;
for i:=1 to PocetSousedu do
  Steering += Sousedi[i].SmerovyVektorRychlosti;
end for
Steering = Steering / PocetSousedu;
Steering.Normalize();
```

4.2.2 Koheze

Další algoritmus se zabývá kohezí ptáků. Nejprve si spočteme střed všech sousedních ptáků a spočítáme směrový vektor mezi lokací ptáka a tímto středem. Tento vektor přičítáme k aktuální rychlosti, čímž simulujeme shlukování ptáků v hejnu (viz pseudokód 7).

```
FVector Steering;
for i:=1 to PocetSousedu do
  Steering += Sousedi[i].Lokace;
end for
Steering = Steering / PocetSousedu;
Steering = Steering - Lokace;
Steering.Normalize();
```

4.2.3 Separace

Aktuálně ptáci v hejnu upravují směr podle ostatních a také se k sobě přibližují. Potřebujeme ještě zaručit, že se budou vzájemně vyhýbat a k tomu nám slouží separace. Na pseudokódu níže znázorním jakým způsobem algoritmus funguje.

```
FVector Steering;
for i:=1 to PocetSousedu do
  float VzdalenostOdSouseda = FVector::Distance(Lokace, Sousedi[i].Lokace)
  Steering += Sousedi[i].Lokace;
  Rozdil = Lokace - Sousedi[i].Lokace
  if VzdalenostOdSouseda * VzdalenostOdSouseda > 0 then
    Rozdil = Rozdil / (VzdalenostOdSouseda * VzdalenostOdSouseda)
    Steering = Steering + Rozdil
  end if
end for
Steering = Steering / PocetSousedu;
Steering.Normalize();
```

4.2.4 Dodatečné informace

V rámci ladění hejnového algoritmu jsem musel omezit maximální rychlost ptáků. Dále jsem nastavil jednotlivým vlastnostem váhy, které jsem během testování upravoval, aby pohyb vypadal nejlépe.

4.3 Vyhýbání se překážkám

Aby ptáci nenaráželi do zdi, bylo potřeba implementovat algoritmus vyhýbání se překážkám. Pták neustále kontroluje volnost průletu ve směru letu. V každém snímku spouští pták paprsek, který při nárazu do jakéhokoliv objektu spustí algoritmus úhybu. Pokud má pták před sebou nějakou překážku, začne spouštět další paprsky v jiných směrech do té doby než najde volný průlet. Pokud uspěje, přidá k aktuální rychlosti směrový vektor reprezentující nový směr. Směrový vektor je vynásoben magnitudou, která se odvíjí od vzdálenosti mezi ptákem a překážkou. Menší vzdálenost znamená větší magnitudu. Do vzorce je dosazená funkce reprezentovaná křivkou, do které dosazujeme vzdálenost.

4.4 Sledovací pohyb

Další funkcionalitu, kterou disponují členové hejna je následování bodu v prostoru. Ve hře využívám této funkcionality např. k tomu, aby ptáci následovali hráče. Pták sleduje bod v prostoru, který jsem nazval FollowingPoint. Tento objekt má dvě konfigurace.

4.4.1 KeepDistance

První se nazývá KeepDistance a jak název napovídá, ptáci se snaží následovat bod v prostoru tak, aby dodržoval co nejlépe stejnou vzdálenost mezi bodem a sebou. V rámci implementace jsem si definoval vzdálenost jak daleko od bodu se má pták zhruba držet. Pokud aktuální vzdálenost toto přesahuje, přidávám rychlost ve směru následujícího bodu. Rychlost závisí na vzdálenosti a aktuální rychlosti bodu. Jestliže aktuální vzdálenost člena hejna je menší jak stanovená míra, pták zpomaluje. Aktuální zpomalení je opět závislé na vzdálenosti a rychlosti bodu.

4.4.2 Impact

Další konfigurace sledování pohybu se nazývá Impact. Pokud vytvoříme FollowingPoint s touto konfigurací, ptáci letí střemhlav za tímto objektem. Hráč uvidí ve hře tuto konfiguraci, když nepřátelští ptáci začnou útočit po hráči.

4.4.3 Statické formace

Předchozí pohyby ptáka je charakteristický svojí rychlostí, ke které v každém snímku přidávám akcelerace. Pohyb tak vypadá více přirozeně. V rámci gameplay byla implementovaná mechanika, kdy hráč má možnost lítat se svým ptactvem ve statické formaci (viz obr. 4.1). Lokaci formace jsem zvolil jako vektorový součet lokace hráče a součinu směrového vektoru kamery a parametru reprezentující vzdálenost.

Obrázek 4.1: Statická formace hejna

Formace ptactva se objeví před očima hráče v nastavené vzdálenosti. V první části implementace pozice ptáka kopírovala definovaný bod statické formace. Takto implementovaný pohyb vypadal příliš sekaný a nehezký. Aby pohyb vypadal více hladký, pohyb byl spočítán jako interpolace mezi aktuální vzdáleností ptáka a cílovou vzdáleností reprezentující lokaci formace. Průběh interpolace je definovaná křivkou, která byla upravena na cílové hodnoty na základě testování.

4.5 Projektilový pohyb

Hlavní účel statické formace je příprava získaných ptáků na jejich vystřelení. Hráč má tedy možnost ovládat svoje ptáky jako zbraň. Tento pohyb simuluje projektilové vystřelení (viz obr. 4.2). Poté, co je projektil vystřelen, se člen hejna po určité vzdálenosti vrací. Tento přechod je definovaný jako přechod stavu.

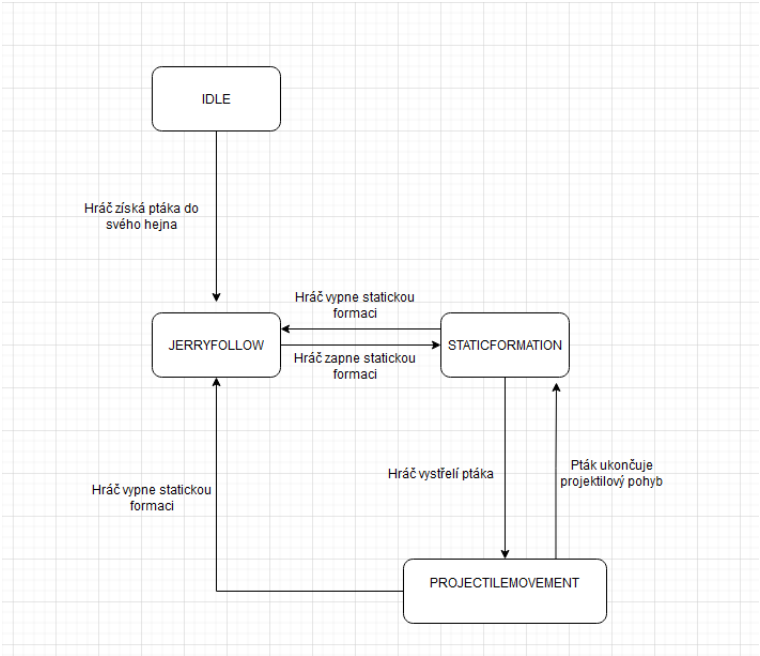
Obrázek 4.2: Projektilový pohyb

4.6 Stavy reprezentující pohyb

Pro efektivní využití jednotlivých algoritmů chování ptáků, byl navržen zjednodušený stavový automat reprezentující následující stavy:

- Idle - Stav reprezentující počáteční stav ptáka v hejnu.
- JerryFollow - Stav reprezentující následování bodu v prostoru
- StaticFormation - Stav reprezentující pohyb ve statické formaci
- Projectile - Stav reprezentující projektilový pohyb

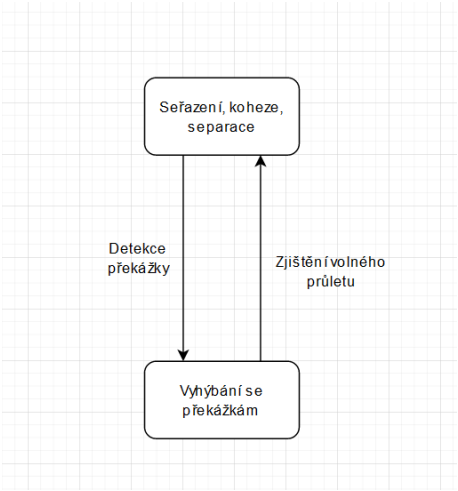
Jednotlivé stavy mohou, ale také nemusí využívat kombinace algoritmů zmíněných výše.



Obrázek 4.3: Diagram znázorňující využití stavů z hráčské perspektivy

4.6.1 Idle

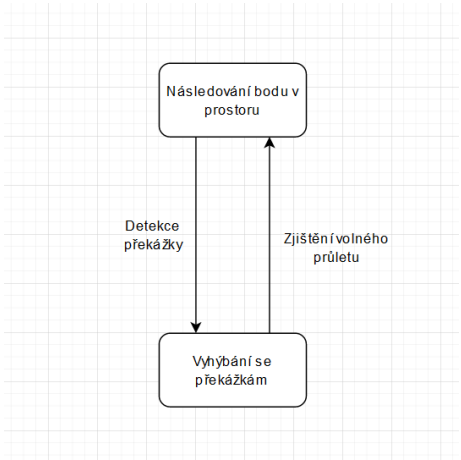
Ptáci v tomto stavu využívají vlastností seřazení, koheze a separace a vyhýbání se překážkám. Během vyhýbání ptáci ignorují ostatní vlastnosti.



Obrázek 4.4: Diagram popisující Idle stav

4.6.2 JerryFollow

Pokud se pták nachází v tomto stavu, musí mít definovaný objekt FollowingPoint s konfigurací KeepDistance. Stejně jako v předchozím stavu, při vyhýbání se překážkám, ptáci ignorují sledovací pohyb.



Obrázek 4.5: Diagram popisující JerryFollow stav

4.6.3 StaticFormation

Tento stav reprezentuje pohyb ve statické formaci. Zároveň je u tohoto stavu definován přechod na projektilový pohyb. Statická formace se v rámci gameplay využívá jako zbraň, kterou může hráč ovládat.

4.6.4 Projectile

Na základě vstupu hráče, pták definovaný tímto stavem se chová jako projektil. Využívá tedy předem definovaný projektilový pohyb. Po zasažení objektu, anebo překonání určité vzdálenosti, pták přechází opět do statické formace.

Kapitola 5

Tvorba prostředí

K tvorbě prostředí byly využity knihovny a nástroje Unreal Enginu. Samotný svět disponuje lesy, trávou, horami, jeskyněmi, architekturou atd. Tvorba prostředí je rozdělená na několik úrovní, které v této kapitole budou rozepsány. V rámci tvorby prostředí byly použity jak zabudované nástroje v Unreal Enginu, tak vlastní nástroje vyvinuté pro herní účely.

5.1 Landscape

Nástroj tvorby landscape se využívá pro tvorbu terénu. Pomocí několika štětců a jiných nastavení si vývojář velice rychle dokáže vytvořit základní kostru pohoří, průsmyků, jezer, údolí atd. K této geometrii je poté důležité přiřadit materiál. Jakým způsobem materiál nastavíme odpovídá vzhledu, který landscape bude nabývat.

Obrázek 5.1: Grafické rozhraní nástroje tvorby landscape

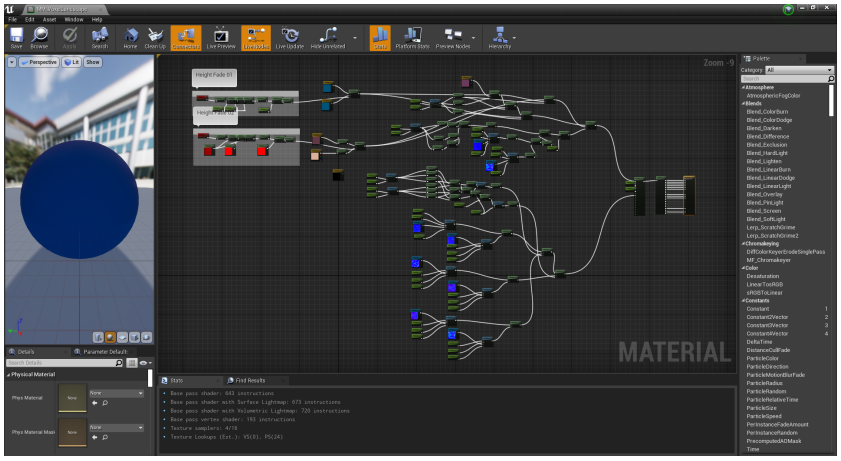
5.2 Procedural Foliage Spawner

Pro tvorbu přírody byl použit Procedural Foliage Spawner. V rámci tohoto nástroje je možné si definovat jednotlivé 3D modely, jako hlavní komponenty prostředí. Těmto komponentám poté stačí nastavit množinu vlastností a definovat v prostoru pomyslný kvádr, kde se budou tyto komponenty zobrazovat. Výhoda tohoto nástroje spočívá v instancování jednotlivých 3d objektů. Pokud nastavíme jednotlivé parametry správně, na libovolné ploše nám vyroste přirozeně vypadající příroda.

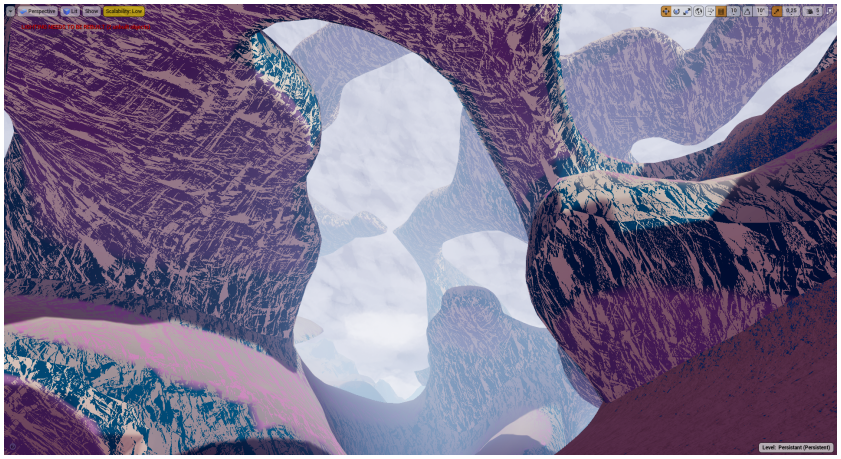
Obrázek 5.2: Použití nástroje Procedural Foliage Spawner

5.3 Voxel plugin

Voxel plugin nám umožnil tvorbu jeskyní v 3D scéně (viz obr. 5.4). Tento plugin nám přidává možnost definovat prostředí v mřížce. Stejně jako tomu je u landscape, potřebujeme přiřadit k prostředí generované voxelem nějaký materiál.



Obrázek 5.3: Materiál reprezentující vizuál jeskyní



Obrázek 5.4: Vytvořené jeskyně pomocí pluginu Voxel

5.4 Mlha a vzdálenost

Kvůli optimalizaci bylo potřeba každému modelu, nebo instanci přiřadit atribut nazývaný cull distance. Tento atribut definuje, při jaké vzdálenosti od hráče se geometrie vykreslí. Pokud se hráč hýbe, při správném nastavení cull distance se jednotlivé geometrie často objevují a mizí. Tento jev byl eliminován použitím mlhy.

Obrázek 5.5: Nastavení různých hodnot cull distance. V levém obrázku je hodnota vyšší, stromů se tak vykreslí více.

Obrázek 5.6: Využití mlhy v 3D scéně

5.5 Gameplay Bricks System

V rámci této práci byl vyvinut systém, který generuje jednotlivé objekty po křivce v 3D scéně. Tento systém byl nazván Gameplay Bricks System. Hlavní myšlenka systému spočívá ve využití tzv. spliny, což je 3D křivka, s kterou level designer může manipulovat v rámci scény. Po této křivce se poté generují jednotlivě nadefinované objekty, reprezentující základní prvky této hry.

5.5.1 Platformer Spline

Hlavním objektem toho systému je Platformer Spline. Tento objekt reprezentuje 3d křivku, kterou si vývojář libovolně protáhne po vytvořené scéně. U tohoto objektu si poté můžeme nastavit množinu parametrů, které ovlivní její vlastnosti. Mezi parametry patří:

- float MinBlocksGap - Minimální vzdálenost mezi dvěma následujícími objekty.
- float MaxBlocksGap - Maximální vzdálenost mezi dvěma následujícími objekty.
- float MinOffset - Minimální vzdálenost kolmo od směru PlatformerSpline
- float MaxOffset - Maximální vzdálenost kolmo od směru Platformer-Spline

Obrázek 5.7: Využití Platformer Spline

Dále je potřeba vyplnit konfigurační pole objektů, které se mají objevit podél 3d křivky. Při generaci objektů podél 3d křivky je vždy brán náhodně jeden z těchto objektů. Objekty, které jsou definovány v tomto poli jsou součástí Gameplay Bricks System.

5.5.2 Penízek

Objekt s názvem Coin je jeden z objektů v Gameplay Bricks System. Objekt má kolizi, která detekuje dotyk s hráčem, nebo hráčovým hejnem. Při dotyku se spustí zvuk signalizující akvizici penízku a zároveň je objekt penízku zničen. Jedná se o základní objekt ve hře reprezentující systém skóre.

Obrázek 5.8: Objekty generované podél Platformer Spline. Zelená koule - SpeedBoost. Žlutá koule - Penízek

5.5.3 SpeedBoost

Mezi další stavební kámen tohoto systému patří objekt SpeedBoost. Jak již název připomíná, pokud hráč sebere tento bod, dojde k dočasnému zrychlení hráčova pohybu.

Kapitola 6

Dodatečné herní mechaniky a implementace

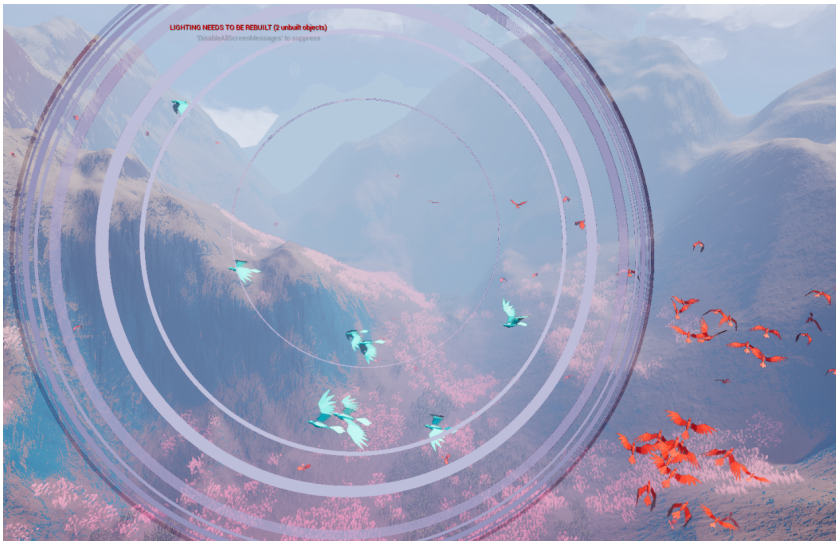
V této kapitole budou rozepsány další mechaniky a algoritmy, které se objevují v praktické části této práce.

6.1 Kontrola ptačích hejn z hlediska hráče

Hráč má ve hře možnosti interagovat s ptačími hejny. Myšlenka hry spočívá s manipulací ptačích hejn. Rozvinutí těchto mechanik je hlavní způsob, jak vytvořit příjemný zážitek při hraní této hry. V rámci této práce bylo navrženo a implementováno několik základních mechanik. Další mechaniky a způsoby jak využít kontroly ptačích hejn hráčem bude rozvíjeno mimo tuto práci. Implementované mechaniky budou rozepsány níže.

6.1.1 Akvizice člena hejna

Hráč na stisknutí tlačítka spustí zvuk ptačího křiku a zároveň vytvoří částicový systém, který křik graficky reprezentuje (viz obr. 6.1). Od kamery hráče se spustí množina paprsků, která připomíná tvar kužele. Pokud některý z těchto paprsků narazí na ptáka v prostoru, pták přejde ze stavu Idle do stavu JerryFollow. Lokace objektu FollowingPoint, který je potřeba vytvořit a přiradit ptactvu v rámci přechodu do stavu JerryFollow, kopíruje souřadnice hráče. Tato lokace je poté upravena tak, aby hráč svoje ptactvo viděl. Výsledná lokace tohoto bodu je vektorový součet lokace hráče a normalizovaného vektoru směru pohybu hráče vynásobeným danou konstantou.



Obrázek 6.1: Mechanika ptačího křiku

6.1.2 Využití statické formace

V případě akvizice členů hejna, získává hráč možnost manipulace. Pokud drží tlačítko ovladače, určitý počet ptáku vlastního hejna se seřadí do statické formace. Výpočet pozice formace vychází ze směru ovladačů. Poté co se určitý počet ptáků seřadí ve formaci, hráč má možnost pomocí stisknutí tlačítka vystřelit jednotlivé členy svého hejna jako projektil.

6.2 Implementování mechanik po síti

Unreal Engine disponuje vlastním frameworkem, který dokáže zprovoznit daný software po síti. Při implementaci není nutné samotné posílání dat po síti a technické problémy s těmito technologiemi. Pro naše účely nám stačí znát jak funguje replikační systém v rámci Unreal Enginu. Pokud má objekt v Unrealu nastavenou replikaci, můžeme jeho vlastnosti zprovoznit pro více hráčů. Aby více hráčů mohlo interagovat s jedním objektem používáme v rámci implementace tzv. Remote Procedure Calls (RPC). U každého objektu máme možnost aby se funkce vykonala lokálně u klienta, na serveru, a nebo u všech zařízení.

6.2.1 Zprovoznění letu po síti

Nejprve bylo potřeba zprovoznit mechaniku lítání pro více hráčů po webové síti. Objekt reprezentující hráče musí mít nastavený atribut replicates, aby se lokace hráče propagovala na server. Pokud se hra takto spustí, změna lokace neovlivní pohyb, jelikož funkce je volaná u klienta. Proto je ještě potřeba

vytvořit funkci, kde se volá přidání sil a u té nastavit, že se spustí na serveru. Tuto funkci poté volá klient v momentě přidání sil k pohybu hráče. Pokud takto nastavíme objekt reprezentující hráče, pohyb funguje po webové síti.

6.2.2 Konfigurace algoritmu simulující ptačí hejna po síti

Stejně jako v předchozím bodě bylo potřeba nastavit jednotlivé funkce tak, aby se vykonávali na serveru. Zároveň objekty reprezentující ptáky v hejnu museli mít nastavenou replikaci po síti. Po zakliknutí replikace v nastavení a úpravě algoritmů tak, aby byli volané na serveru pomocí RPC, algoritmus simulující ptačí hejna byl zprovozněn po webové síti.

Kapitola 7

Testování

V rámci této kapitoly bude popsána analýza testování jednotlivých algoritmů. V rámci analýzy budou sepsány poznatky v jednotlivých iteracích.

7.1 Testování pohybu simulující let hráče

Pohyb hráče se skládá z jednotlivých pohybových algoritmů. Tyto algoritmy byli nejprve testovány zvlášť a následně jejich kombinace. V následujících sekcích popíšu chyby a připomínky poznatků.

7.1.1 Verze 1.0

V první verzi byli implementovány pouze dva algoritmy pohybu a to aerodynamický vztlak a aerodynamický odpor. U aerodynamického odporu chyběla globální složka algoritmu. V této verzi byly testovány mechaniky hráče bez využití ptačích hejn.

Chyby a připomínky

- Nastavené hodnoty parametrů nebyly odladěné, odpor a vztlak nebyl uživatelsky přívětivý.
- Aerodynamický odpor se příliš neprojevoval v kombinaci se vztlakem.
- Hráč neměl kontrolu nad změnou směru.
- Výpočty obsahovali chybu, kvůli kterým se výsledná síla měnila v závislosti poklesu FPS.

7.1.2 Verze 2.0

V další verzi byly opraveny výpočetní chyby. Ukázalo se, že v prvotní verzi program počítal rychlost bez aplikace deltasekundy. V této verzi zároveň byly

implementovány algoritmy otáčení a úhybu. Zároveň se program upravil tak, aby vztlak se projevoval pouze v případě nízké odporové síly. V této verzi byly implementovány systémy 3D prostředí, konkrétně Gameplay Bricks System. Testovalo se nabírání Penízků a SpeedBoostů.

Chyby a připomínky

- Odpor opět nebyl rozdělen na lokální a globální složku. Výsledný let byl občas matoucí.
- Oproti minulé verzi pohyb byl více uživatelsky přívětivý.
- Konstanty otočení a úhybu nebyly odladěné.
- Hráč při zrychlení SpeedBoostu mohl nabrat příliš velkou rychlost.

7.1.3 Verze 3.0

Verze 3.0 představuje odladěnou verzi s nejpřívětivějším uživatelským zážitkem. Byla přidána lokální a globální složka aerodynamického odporu. Hra se chovala stejně při poklesu FPS. V této byla testována akvizice ptáků do vlastního hejna a mechaniky spjaté s jeho kontrolou. Hráč tak mohl vytvořit statické formace ptáků a využít je ke střelbě.

	Verze 1.0	Verze 2.0	Verze 3.0
Pohyb máchání křídly	✓	✓	✓
Pohyb plachtění	✓	✓	✓
Kontrola otáčení		✓	✓
Úhyb do stran		✓	✓
Tvorba vlastního hejna			✓
Nastavení kolizí se světem		✓	✓
Sbírání bodů		✓	✓
SpeedBoost zrychlení		✓	✓
Využití statické formace			✓
Střílení ptáků			✓
Absence výskytu chyb		✓	✓

Obrázek 7.1: Srovnávací tabulka mechaniky letu

7.2 Testování algoritmů simulující ptačí hejna

V této kapitole bude popsány jednotlivé iterace testování ptačích hejn.

7.2.1 Verze 1.0

V první verzi byl implementován pouze pohyb a základní algoritmy koheze, seřazení a separace. Jednotlivý ptáci byly v 3D scéně vykresleny jako jehlany s šipkami reprezentující směr.

Chyby a připomínky

- Testování pohybu bylo obtížné, jelikož členové hejna po krátkém časovém intervalu odletěli mimo viditelný dosah.
- Simulace vypadala nedodělaně, jelikož v této verzi neexistovali 3D modely reprezentující ptáky.
- Parametry jako maximální rychlost, váhy jednotlivých vlastností nebyly vyladěné. V této verzi hejna spíše připomínala hmyz

7.2.2 Verze 2.0

V této verzi bylo implementováno vyhýbání se překážkám. Zároveň byla vytvořena grafická reprezentace ptáka v 3D scéně pomocí instancovaných meshů. Bylo potřeba v rámci shaderu přidat vlastní data reprezentující každou instanci a také vytvořit animace 3D modelu na úrovni shaderu pomocí 3D modelovacího programu Blender. Pro snadné testování byl implementován objekt s názvem BoidSpawner, který slouží jako rozhraní pro vytvoření ptáku a ohraničení jejich letu.

Chyby a připomínky

- Kombinace hejnových algoritmů a vyhýbání se překážkám způsobovalo občasný průlet ohraničení, nebo 3D geometrie.
- Systém již měl správně nastavené parametry, tedy samotný pohyb v této verzi vypadal přijatelně.

7.2.3 Verze 3.0

Tato verze byla upravena tak, aby vyhýbání překážkám mělo vždy prioritu. Zároveň algoritmus úhybu byl upraven tak, aby intenzita vyhýbání byla definovaná pomocí vzdálenosti mezi ptákem a nárazem. Díky těmto změnám, pták nenarazí do zdi. V této verzi byly následně přidány algoritmy následování bodu v prostoru, statické formace a projektilový pohyb.

Chyby a připomínky

- Následování bodu bylo nevyladěné. Zrychlování a zpomalování ptáků při následování nebylo plynulé.

- Implementace projektilového pohybu a statických formací fungovali správně. Jediným problémem byla absence jejich rotace podle aktuálního směru.

7.2.4 Verze 4.0

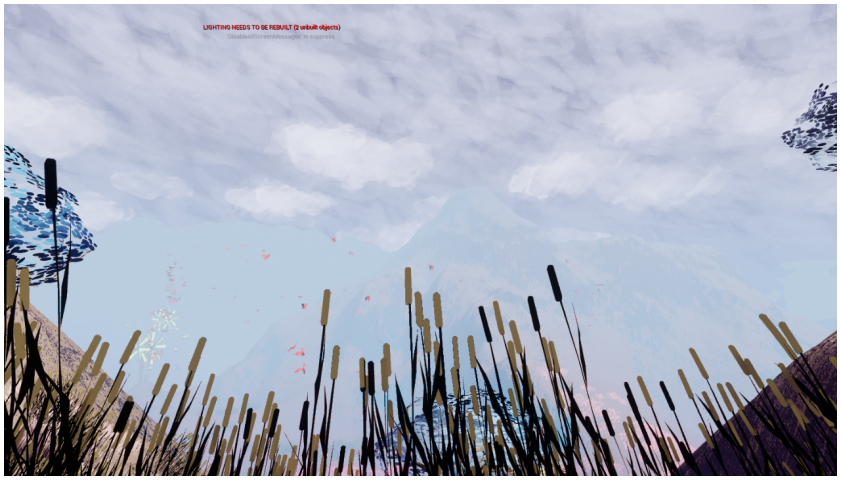
V této verzi byly odladěny předchozí chyby a připomínky. Dále byla implementována abstrakce stavových automatů reprezentující ptačí pohyb. Tato abstrakce bude v budoucnu sloužit k implementaci složitějších gameplay situací.

	Verze 1.0	Verze 2.0	Verze 3.0	Verze 4.0
Koheze	✓	✓	✓	✓
Seřazení	✓	✓	✓	✓
Separace	✓	✓	✓	✓
Úhyb překážkám		✓	✓	✓
Následování bodu v prostoru			✓	✓
Statické formace			✓	✓
Vystřelovací pohyb			✓	✓
Abstrakce stavovým automatem				✓
Grafická reprezentace ptáka		✓	✓	✓

Obrázek 7.2: Srovnávací tabulka implementace ptačích algoritmů

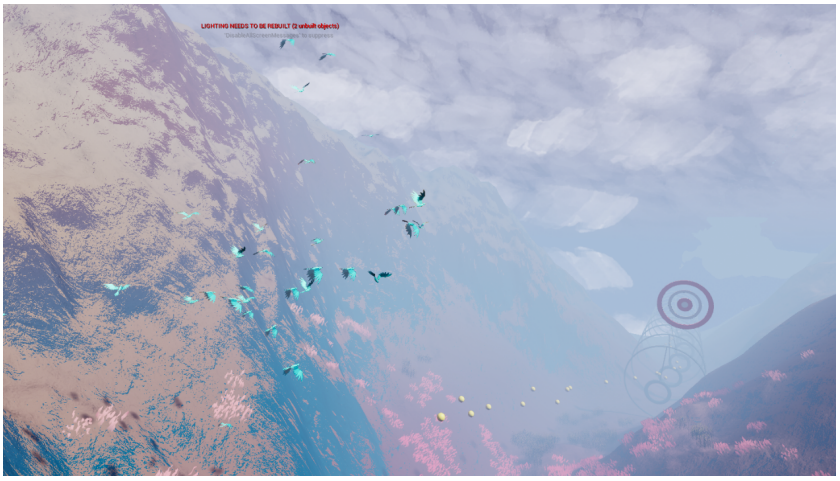
7.3 Popis testovací úlohy

Algoritmy byly otestovány v rámci testovací úlohy prezentující implementované mechaniky. Hráč se nejprve objeví v mapě v hornatém údolí reprezentující hnízdo. Před hnízdem se nachází údolí, ve kterém se nachází hejno červených ptáků. Hráč nejprve otestuje mechaniku křiku pomocí stisknutí tlačítka. Po stisknutí tlačítka ovladače v simulaci se zobrazí grafická reprezentace křiku doprovázená zvukovým podkladem. Cílem je nejprve pomocí mechanik letu doletět k hejnu a nasbírat křikem jednotlivé členy hejna. Pták po akvizici změní barvu z červené na modrou. Dále se hráč snaží letět podél vytvořených penízků se svým ptactvem. Hráč se snaží letět tak, aby nenarazil a sebral co nejvíce penízků. Během letu hráč prolétává kolem terčů, které se snaží trefit pomocí mechaniky statických formací a projektilového pohybu. Testovací úloha končí v momentě průletu posledním penízkem.



Obrázek 7.3: Start hráče v údolí reprezentující hnízdo

Obrázek 7.4: Hejna červených ptáků



Obrázek 7.5: Let s posbíranými hejny ptáků

Obrázek 7.6: Ptáci poletující podél 3D modelů ve světě

Závěr

Cílem této práce bylo navrhnout a zrealizovat základy pro hru na virtuální realitu. Mezi implementované mechaniky patří pohyb simulující let ptáka z hráčské perspektivy, herní AI reprezentovaná systémem ptačích hejn, systémy pro tvorbu prostředí a následná kombinace těchto systémů. Hlavní důvod pro zrealizování letu ve VR byla invence nové pohybové mechaniky. Ve většině VR her se hráč pohybuje joystickem, teleportem, anebo pohyb kopíruje reálný pohyb hráče. Mechanika letu ve VR pro herní účely mi proto dávala smysl zrealizovat.

Využití algoritmů pro simulaci ptačích hejn má potenciál v tvorbě unikátních herních zážitků. V budoucnu mám v plánu rozvinout tento systém o další gameplay mechaniky. Hejna by mohli např. reprezentovat nepřátelské jednotky, nebo identifikovat životy hráče. Tento systém by se také mohl velice pěkně využít k závodní hře. Tvorba prostředí a jeho využití z hlediska gameplay je velice rozsáhlá kapitola vývoje pro VR, kterou jsem pouze částečně zpracoval, neboť jsem zjistil, že počet způsobů a metod jak k této problematice přistupovat je vysoký. Snažil jsem se tedy přistupovat k tvorbě 3D scén převážně z hlediska optimalizace. Často jsem nemohl využít tradičních nástrojů, jelikož v rámci vývoje na VR bych příliš zatěžoval cílové zařízení.

Myslím si, že tato práce splnila stanovené cíle, jelikož implementované mechaniky přinášejí uživateli zajímavé herní zážitky.

Literatura

[1] Wong, Timm. Boids. *Stanford* [online]. 2008 Available from: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2008-09/modeling-natural-systems/boids.html

[2] Sane, Sanjay P., and Michael H. Dickinson. "The control of flight force by a flapping wing: lift and drag production."*Journal of experimental biology* 204.15 (2001): 2607-2626.

[3] EPIC GAMES, *Unreal Engine 4 Documentation.* Unreal Engine Documentation. (n.d.). https://docs.unrealengine.com/en-US/index.html.

[4] Reynolds, Craig W. "Flocks, herds and schools: A distributed behavioral model."*Proceedings of the 14th annual conference on Computer graphics and interactive techniques.* 1987.

[5] McCaffrey, Mitch. *Unreal Engine VR Cookbook: Developing Virtual Reality with UE4.* Addison-Wesley Professional, 2017.

[6] Somrak, Andrej, et al. "Estimating VR Sickness and user experience using different HMD technologies: An evaluation study."*Future Generation Computer Systems* 94 (2019): 302-316.

[7] Geršak, Gregor, Huimin Lu, and Jože Guna. "Effect of VR technology matureness on VR sickness."*Multimedia Tools and Applications* 79.21 (2020): 14491-14507.

Příloha A

Seznam použitých zkratek

3D Graphical user interface

Janitor Effect Extensible markup language

blueprint Objekt s konfiguračním prostředím v rámci enginu. Disponuje možností vizuálního programování.

gizmo 3D kurzor sloužící ke kontrole objektu v editoru scén

VR virtuální realita

headset virtuální brýle, které simulují hráči virtuální realitu

FPS frames per second

deltasekunda časový rozdíl dvou po sobě jdoucích snímků

instancovaný mesh 3D objekt v Unreal Enginu, který je méně náročný na vykreslován;í

bottleneck část v simulaci, kde dochází k problémům s optimalizací

mesh 3D objekt

skeletal mesh 3D objekt, který obsahuje kostru

shader program, který řídí vykreslování grafiky

cull distance vzdálenost od kamery v simulaci, po které se objekt přestane vykreslovat

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF