**Technická univerzita v Košiciach**
**Fakulta elektrotechniky a informatiky**

# Softvérové riešenia pre vizualizáciu experimentálnych údajov vo virtuálnej realite

**Bakalárska práca**

**2025** **Oleh Leshchenko**

**Technická univerzita v Košiciach**
**Fakulta elektrotechniky a informatiky**

# Softvérové riešenia pre vizualizáciu experimentálnych údajov vo virtuálnej realite

**Bakalárska práca**

| | |
|---|---|
| Študijný program: | Informatika |
| Študijný odbor: | Informatika |
| Školiace pracovisko: | Katedra počítačov a informatiky (KPI) |
| Školiteľ: | Ing. Štefan Korečko, PhD. |

**Košice 2025** **Oleh Leshchenko**

## Abstrakt v SJ

Táto bakalárska práca sa zaoberá návrhom, implementáciou a porovnaním softvérových riešení na vizualizáciu experimentálnych údajov vo virtuálnej realite. Hlavným cieľom bolo otestovať alternatívne platformy – Unreal Engine a Unity – ako alternatívu k existujúcemu webovému riešeniu NDMVR. Vytvorené boli funkčné prototypy v každom prostredí, ktoré boli následne testované na prenosnom počítači a v náhlavnej súprave Meta Quest 2. Výsledky experimentov ukázali výrazné rozdiely vo výkone v závislosti od platformy a cieľového zariadenia. Práca poskytuje odporúčania pre ďalší vývoj riešení.

## Kľúčové slová v SJ

virtuálna realita, vizualizácia údajov, Unreal Engine, Unity, NDMVR, výkon, porovnanie platforiem, histogramy, VR vizualizácia

## Abstrakt v AJ

This bachelor's thesis focuses on the design, implementation, and comparison of software solutions for visualizing experimental data in virtual reality. The main objective was to evaluate alternative platforms — Unreal Engine and Unity — as potential replacements for the existing web-based solution NDMVR. Functional prototypes were developed in each environment and tested on a laptop and the Meta Quest 2 VR headset. The experimental results revealed significant performance differences depending on the platform and target device. The thesis provides recommendations for further development.

## Kľúčové slová v AJ

virtual reality, data visualization, Unreal Engine, Unity, NDMVR, performance, platform comparison, histograms, VR visualization

## Bibliografická citácia

LESHCHENKO, Oleh. *Softvérové riešenia pre vizualizáciu experimentálnych údajov vo virtuálnej realite*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2025. 65s. Vedúci práce: Ing. Štefan Korečko, PhD.

77195

**TECHNICKÁ UNIVERZITA V KOŠICIACH**

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

# ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

**Softvérové riešenia pre vizualizáciu experimentálnych údajov vo virtuálnej realite**

Software Solutions for Experimental Data Visualization in Virtual Reality

Študent: **Oleh Leshchenko**
Školiteľ: **Ing. Štefan Korečko, PhD.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce:
Pracovisko konzultanta:

Pokyny na vypracovanie bakalárskej práce:

1. Oboznámiť sa s webovým komponentom NDMVR pre vizualizáciu údajov z fyzikálnych experimentov vo virtuálnej realite.
2. Analyzovať ďalšie súčasné možnosti a softvérové riešenia pre takúto vizualizáciu.
3. Pre vybrané softvérové riešenia navrhnúť alternatívne vizualizácie so základnou funkcionalitou porovnateľnou s komponentom NDMVR.
4. Alternatívne vizualizácie implementovať s dôrazom na spracovanie zdrojových údajov v príslušnom formáte a sieťovú komunikáciu.
5. Implementované vyzualizácie experimentálne vyhodnotiť, v porovnaní s komponentom NDMVR.
6. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 23.05.2025
Dátum zadania bakalárskej práce: 31.10.2024

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -10-

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

## Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

## Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce za jeho čas a odborné vedenie počas riešenia mojej záverečnej práce. Rovnako by som sa rád poďakoval svojim rodičom a priateľom za ich podporu a povzbudzovanie počas celého môjho štúdia.

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Zoznam výpisov

# 1 Úvod

Vizualizácia údajov je veľmi dôležitým aspektom pre analýzu a lepšie pochopenie zložitých javov, najmä pokiaľ ide o experimentálne údaje pre vedcov. Vo fyzike (primárne v časticovej fyzike), kde experimenty generujú veľké množstvo údajov, môže byť bez vizualizácie oveľa ťažšie analyzovať namerané údaje a pochopiť priebeh udalostí, ku ktorým došlo počas experimentu [1]. Reprezentácia v n-rozmernom priestore pomocou virtuálnej reality otvára nové možnosti intuitívnej analýzy a komplexného prehľadu údajov.

Táto bakalárska práca pokračuje v práci na softvérovom komponente NDMVR na vizualizáciu experimentálnych údajov v rozšírenej realite. NDMVR poskytuje možnosť vizualizovať údaje priamo cez webový prehliadač pomocou softvérového rámca AFrame a knižnice JSROOT bez potreby inštalácie ďalšieho softvéru. Hoci tento prístup má výhodu vysokej dostupnosti, predchádzajúce testy ukázali, že pri vizualizácii veľkého množstva údajov môže mať určité výkonnostné obmedzenia. Preto bolo rozhodnuté preskúmať alternatívne riešenia, aby sa zistilo, či iné platformy poskytujú lepšiu stabilitu alebo výkon. Webová verzia bola ďalej rozvíjaná v rámci tímového projektu paralelne s touto prácou. Táto bakalárska práca sa preto zameriava na alternatívne riešenia.

Motiváciou tejto práce je nájsť softvér na vizualizáciu experimentálnych údajov v prostredí virtuálnej reality, ktorý používateľom umožní jednoduchý prístup k údajom. Použitie virtuálnej reality môže zvýšiť zapojenie používateľov a efektívnosť analýzy údajov pomocou trojrozmerného prostredia, ktoré uľahčuje orientáciu a interakciu. Systém je navrhnutý pre trojrozmerné virtuálne prostredie a aktuálne podporuje vizualizáciu histogramov v jednom, dvoch a troch rozmeroch. Do budúcnosti je možné jeho funkcionalitu rozšíriť tak, aby podporoval analýzu histogramov s vyššou dimenzionalitou (ako naznačuje názov NDMVR — N-DiMensional Virtual Reality).

Cieľom tejto bakalárskej práce je preskúmať a porovnať možnosti, ako sú Unreal Engine, Unity a webové technológie, s cieľom nájsť platformu na efektívnu vizualizáciu veľkého množstva údajov. Hoci použitie webových technológií má

výhodu dostupnosti (nie je potrebný žiadny ďalší softvér), je dôležité posúdiť, či sú výkonnostné a optimalizačné možnosti Unreal Engine a Unity lepšie ako možnosti webovej platformy.

## 1.1 Formulácia úlohy

Úlohou tejto práce je analyzovať možnosti a obmedzenia existujúcej webovej platformy a preskúmať, či alternatívne riešenia, ako napríklad Unreal Engine a Unity, môžu poskytnúť stabilnejšie a efektívnejšie prostredie na vizualizáciu experimentálnych údajov vo VR. Cieľom je nielen zlepšiť plynulosť a výkon, ale aj posúdiť prístupnosť a jednoduchosť používania týchto platforiem z pohľadu používateľa (napr. vedca alebo výskumníka), ktorému platforma poskytuje nástroje na analýzu údajov.

Práca sa zameria na niekoľko kľúčových oblastí:

- Základná analýza existujúcej platformy NDMVR a oboznámenie sa s alternatívnymi riešeniami (Unreal Engine a Unity) so zameraním na ich výkonnostný potenciál pri vizualizácii veľkého množstva údajov. Táto časť bola spracovaná v *Kapitole 3*.
- Prieskum alternatívnych platforiem (Unreal Engine, Unity) s cieľom implementovať vzorové riešenia na týchto platformách a analyzovať ich výhody a nevýhody, najmä z hľadiska výkonu pri práci s veľkými súbormi údajov a možnosti optimalizácie pre prostredie VR. Analýza prác, v ktorých sa porovnáva výkonnosť Unreal Engine a Unity, bola spracovaná v *Kapitole 4*. Implementácia riešení bola podrobne opísaná v *Kapitole 5*.
- Posúdenie výkonnostného správania platforiem s ohľadom na ich potenciál pre budúce použitie v kontexte používateľských potrieb. Výsledky budú založené na praktických testoch v kontrolovaných podmienkach, aby bolo porovnanie čo najobjektívnejšie. Metodika experimentálneho porovnania troch prostredí bola opísaná v *Kapitole 6*. a analýza výsledkov tohto porovnania sa nachádza v *Kapitole 7*.

Záverečným cieľom tejto práce je definovať platformu, ktorá nielenže ponúka vysoký výkon a používateľskú prívetivosť pri vizualizácii experimentálnych údajov vo virtuálnej realite, ale poskytuje aj dostatočný potenciál na ďalšie rozšírenie a prispôsobenie budúcim požiadavkám a výskumným scenárom.

# 2 Virtuálna realita

Táto kapitola predstavuje úvod do koncepcie virtuálnej reality (VR) so zameraním na jej využitie vo vedeckom výskume. Opisuje etapy vývoja VR od počiatočných imerzívnych myšlienok až po moderné technologické riešenia, ktoré umožňujú úzku interakciu s údajmi a prostredím.

Kapitola skúma kľúčové zložky virtuálnej reality, ktoré sú dôležité pre vedecké aplikácie: ponorenie, interakciu a zapojenie používateľa. Skúma tiež, či tieto komponenty môžu mať vplyv na vzdelávací proces a na pochopenie a analýzu údajov vo všeobecnosti.

Tieto informácie poskytujú zásadný kontext pre pochopenie ďalších častí práce, najmä skúmania softvérových riešení na vizualizáciu experimentálnych údajov vo virtuálnej realite.

## 2.1 Rozvoj virtuálnej reality

Koncept virtuálnej reality vznikol z túžby ponoriť používateľov do sveta podobného tomu nášmu, kde interakcia poskytuje okamžitú spätnú väzbu. To sa dosiahlo prostredníctvom trojrozmerného prostredia a použitia prilieb a rukavíc na prenos údajov a manipuláciu s objektmi v priestore. Medzi najstaršie zdokumentované prípady využitia virtuálnej reality patrí jej použitie v divadle na vytváranie imerzívnych predstavení [2]. Neskôr sa VR vďaka technologickému procesu a tejto imerzívnej povahe rozšírila na výskumné aplikácie, keďže poskytovala komplexné prostriedky na skúmanie zložitých systémov a parametrov. Virtuálna realita si postupne nachádza svoje miesto aj vo vedeckom výskume, najmä pri vizualizácii údajov a interaktívnej analýze.

Moderné systémy VR ponúkajú vysoký stupeň pohltenia a interaktivity. To umožňuje nielen využívanie vzdelávacích materiálov, ale aj vytváranie autentických situácií pri učení [3]. Vo vede a vzdelávaní je tento prístup nevyhnutný na štúdium komplexných javov alebo procesov.

## 2.2 Vizualizácia údajov a virtuálna realita

Vizualizácia údajov je nástroj, ktorý umožňuje premeniť nespracované údaje na vizuálne zrozumiteľné informácie, ako sú grafy a diagramy. Pomáha vidieť niektoré skryté aspekty, ktoré môžu byť pri pohľade na samotné čísla prehliadnuté.

Anscombovo kvarteto je skvelou ilustráciou tohto princípu [4]. Tento slávny príklad ukazuje, že štyri rôzne súbory údajov môžu mať rovnaké štatistické vlastnosti - napríklad rovnaký priemer, štandardnú odchýlku a korelačný koeficient - ale pri vizuálnom zobrazení vyzerajú úplne inak. To poukazuje na dôležitú vec: samotné číselné súčty môžu zakryť významné rozdiely v rozložení a vzťahoch údajov.

Vizualizácia údajov preklenuje priepasť medzi surovými číslami a ľudským chápaním, pričom transformuje abstraktné štatistické informácie na zmysluplné, interpretovateľné reprezentácie, ktoré môžu prispieť k rozhodovaniu a vedeckému pochopeniu.

Virtuálna realita prináša revolúciu vo vizualizácii údajov. Na rozdiel od tradičných dvojrozmerných prezentácií ponúka VR priestorový, interaktívny zážitok, ktorý používateľom umožňuje [5]:

- dynamicky sa presúvať medzi jednotlivými zobrazeniami údajov
- skúmať údaje z viacerých perspektív
- priamo manipulovať s dátovými štruktúrami
- vytvárať intuitívnejšie pochopenie zložitých vzťahov

Tento prístup môže byť obzvlášť prínosný v oblastiach, kde sa pracuje s veľkým množstvom údajov alebo zložitými dátovými štruktúrami — napríklad v oblasti časticovej fyziky.

## 2.3 Virtuálna realita vo vzdelávaní

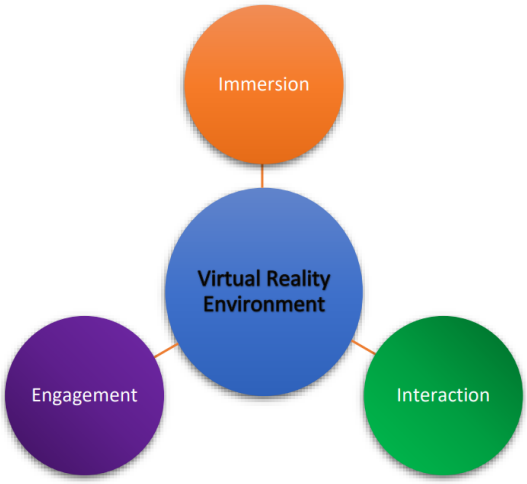
Virtuálna realita je každým rokom dostupnejšia pre ľudí. V dôsledku toho už mnohí uvažujú o jej využití v rôznych oblastiach: medicíne, molekulárnej biológii atď. V nasledujúcom texte sa analyzuje využitie virtuálnej reality vo vzdelávaní. V článku [3] sú podrobne opísané hlavné aspekty VR, ktoré zohrávajú dôležitú úlohu pri zvyšovaní kvality výučby.

Efektívne využitie VR vo vede závisí od troch hlavných zložiek [3]:

- Ponorenie (Immersion): VR dokáže vytvoriť pocit prítomnosti vo virtuálnom prostredí. Dosahuje sa to pomocou technológií, ktoré poskytujú vizuálnu, zvukovú a inú zmyslovú stimuláciu. Vďaka tomu je prostredie vnímané ako skutočné.

- Interakcia (Interaction): Možnosť používateľa ovplyvňovať prvky virtuálneho sveta. Ide o kľúčovú vlastnosť VR , ktorá umožňuje experimentovať a simulovať ako v reálnom živote. Tento prístup podporuje učenie prostredníctvom praxe a skúseností.

- Zapojenie (Engagement): Motivácia používateľa k aktívnej účasti na prostredí VR . Dosahuje sa to realistickým stvárnením prostredia, ktoré podnecuje záujem a túžbu pokračovať v skúmaní alebo učení.




Obrázok 2.1: Hlavné aspekty prostredia VR [3]

Aspekty, ktoré ovplyvňujú efektivitu vzdelávania vo virtuálnej realite, sú znázornené na obr. 2.1.

Virtuálna realita už dávno prekročila rámec jednoduchej vizualizácie a ponúka nové metódy skúmania údajov. Ako uvádza [6], prostredia VR umožňujú simulovať nielen statické objekty, ale aj dynamické procesy, s ktorými možno interagovať a študovať ich v reálnom čase.

Modelovanie viacrozmerných prostredí, kde možno VR využiť na štúdium údajov s veľkým počtom parametrov, je jednou z najperspektívnejších oblastí. Je to vďaka schopnosti systémov virtuálnej reality vytvárať viacrozmerné priestory, v ktorých môžu výskumníci študovať vplyv rôznych parametrov v interaktívnom prostredí. V tejto práci sa tento aspekt osobitne nerieši vzhľadom na iný hlavný cieľ, ale otvára nové možnosti ďalšieho zlepšovania NDMVR.

# 3 Použité prostredia a technológie

Táto kapitola sa zaoberá technologickými nástrojmi a prostrediami používanými na vizualizáciu experimentálnych údajov vo virtuálnej realite. Opisujú sa platformy ako NDMVR, Unreal Engine a Unity. Uvažuje sa o ich výkonnostných charakteristikách a funkciách. Pre túto prácu boli zvolené platformy Unreal Engine a Unity, ktoré patria medzi najpopulárnejšie herné rámce [7]. Vďaka ich rozšírenosti, rozsiahlej dokumentácii a komunite sú často využívané aj pri vývoji aplikácií pre virtuálnu realitu.

## 3.1 Platforma NDMVR

NDMVR (N-DiMensional Virtual Reality) predstavuje softvérové riešenie určené na vizualizáciu experimentálnych údajov priamo vo webovom prostredí pomocou virtuálnej reality.

V kapitole je stručne zhrnutá architektúra a základná funkcionalita systému, ktorý bol vyvinutý predovšetkým na vedecký výskum, najmä v oblasti časticovej fyziky.

Obsahuje aj úvodné zhodnotenie výkonnostných charakteristík NDMVR na základe predchádzajúcich pozorovaní — poukazuje na jeho výhody pri práci s menšími množstvami údajov a zároveň naznačuje výzvy, ktoré môžu vzniknúť pri spracovaní rozsiahlejších datasetov. Tieto poznatky slúžia ako východisko pre ďalší výskum a optimalizáciu systému.

### 3.1.1 Architektúra a funkčnosť NDMVR

NDMVR je softvérový komponent určený na vizualizáciu jedno-, dvoj-a trojrozmerných histogramov (TH1, TH2 a TH3) v prostredí webovej virtuálnej reality. Systém bol vyvinutý na použitie vo vedeckom výskume (najmä v časticovej fyzike), kde sa vyžaduje spracovanie a vizualizácia údajov [8]. Hlavnou výhodou NDMVR je, že sa dá používať bez inštalácie ďalšieho softvéru, lebo beží priamo vo webovom prehliadači.

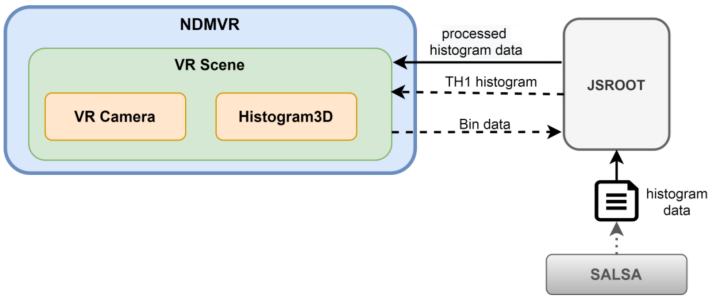
### Architektúra NDMVR

Architektúra NDMVR pozostáva z niekoľkých komponentov, ktoré zabezpečujú vizualizáciu údajov. Takisto na to boli použité aj softvérové knižnice a rámce. Medzi takéto knižnice a rámce patria:

- AFrame[9]: Ramec ktorý slúži na vytváranie a vykresľovanie trojrozmerných scén. Podporuje aj rôzne zariadenia, napríklad náhlavné súpravy VR, vďaka čomu je vizualizácia flexibilnejšia.
- JSROOT[10]: Táto knižnica sa používa na zobrazenie histogramov a je integrovaná do scény VR na vizualizáciu údajov. JSROOT je tiež zodpovedná za spracovanie údajov zhromaždených vo formáte ROOT.
- React.js[11]: Poskytuje optimalizáciu výkonu a flexibilitu pri vývoji komponentov.

Medzi hlavné komponenty architektúry patrí Histogram3D na vykresľovanie histogramov a VR Camera na poskytovanie pohľadu prvej osoby.[8]

Taktiež NDMVR je súčasťou väčšieho systému, ktorý zahŕňa ďalšie prvky, ako napríklad SALSA (Scalable Adaptive Large Structures Analysis). SALSA je zodpovedná za dynamické spracovanie údajov a podporuje distribúciu údajov, čo umožňuje systému pracovať s údajmi z rôznych zdrojov súčasne [8]. Tým sa výrazne rozširuje funkčnosť NDMVR a umožňuje analyzovať veľké množstvo údajov v reálnom čase. Vizualizácia architektúry systému je znázornená na obr. 3.1.




Obrázok 3.1: Komponenty prostredia NDMVR [8]

### Funkčnosť a používateľské rozhranie

Systém podporuje dva režimy prevádzky [8]:

- Normálny režim: Používa sa na zobrazenie 3D scén vo webovom prehliadači. V tomto režime môžu používatelia pracovať s histogramami, posúvať sa po osiach, vyberať jednotlivé biny na podrobné zobrazenie a zobraziť histogram TH1 pre vybraný kôš.
- Režim virtuálnej reality: Umožňuje používateľom ponoriť sa do trojrozmernej scény pomocou náhlavnej súpravy VR, ktorá poskytuje plnú interaktivitu a možnosť zobrazenia údajov z viacerých uhlov.

Používatelia sa môžu pohybovať po scéne pomocou štandardných zariadení (myš a klávesnica), ako aj ovládačov náhlavnej súpravy VR, čím sa systém stáva pre používateľa flexibilnejším.

### 3.1.2 Výkon a limity NDMVR

NDMVR vykazuje stabilný výkon pri práci s malým množstvom údajov, čo umožňuje jeho efektívne využitie na základnú vedeckú vizualizáciu a vzdelávacie účely. Systém dosahuje dobré výsledky pri vizualizácii údajov so stredným počtom binov, pričom poskytuje pohodlný a neprerušovaný používateľský zážitok.

Keď sa však množstvo údajov zvýši na veľkú škálu, napríklad na pole 1000 x 1000 binov, systém stráca stabilitu a môže sa zrútiť [12]. Toto obmedzenie znižuje jeho efektívnosť pri práci s veľkými súbormi údajov, čo je dôležitý aspekt vedeckého výskumu, ktorý si vyžaduje analýzu veľkého množstva informácií.

Treba však poznamenať, že tieto zistenia vychádzajú zo staršej verzie systému. Na jeho ďalšom vývoji sa medzičasom pracovalo v rámci paralelného tímového projektu. Aj preto sa v tejto práci porovnávajú alternatívne riešenia, ako Unreal Engine a Unity, s cieľom identifikovať najvhodnejšiu platformu pre budúce rozšírenie systému.

## 3.2 Platforma Unreal Engine

Táto kapitola stručne predstavuje Unreal Engine (UE) a jeho hlavné technologické charakteristiky, ktoré sú relevantné z hľadiska vizualizácie údajov vo virtuálnej realite. Pozornosť sa venuje najmä vybraným funkciám enginu, ktoré ho predurčujú na efektívne spracovanie a vykresľovanie komplexných 3D scenérií.

Zároveň sa diskutuje o potenciáli UE ako alternatívneho riešenia k pôvodnej webovej platforme NDMVR, s cieľom prekonať jej výkonnostné obmedzenia.

### 3.2.1 Úvod do nástroja Unreal Engine

Unreal Engine (UE) je výkonný a multifunkčný herný engine vyvinutý spoločnosťou Epic Games [13]. Unreal Engine bol pôvodne navrhnutý na vývoj videohier, ale stal sa univerzálnym nástrojom na vytváranie širokej škály interaktívnych aplikácií vrátane architektonickej vizualizácie, filmovej produkcie, virtuálnej reality a ďalších.

Unreal Engine má vďaka svojej architektúre a nástrojom na vykresľovanie schopnosť vytvárať vysokokvalitnú grafiku a komplexné scény. Podporuje pokročilé technológie, ako je sledovanie lúčov, fyzikálne správne osvetlenie a časticové efekty. To umožňuje vytvárať realistické prostredia a postavy. Vďaka výkonnému editoru a možnostiam programovania v Blueprints (vizuálne programovanie) a C++.

### 3.2.2 Používanie nástroja Unreal Engine vo virtuálnej realite

Unreal Engine je všeobecne uznávaný pre svoju schopnosť poskytovať prvotriedne VR zážitky a kompatibilitu s poprednými VR platformami, ako sú Oculus Rift a HTC Vive, ako aj Valve Index a PlayStation VR [13]. Vďaka tomuto množstvu funkcií sa stal najlepšou voľbou na vytváranie aplikácií VR v rôznych odvetviach, od hier cez simulácie až po školiace programy.

Použitie Unreal Engine pre VR ponúka výhodu jeho schopnosti podporovať technológie, ktoré umožňujú pohlcujúce a realistické prostredia VR [14]. Engine poskytuje snímkovú frekvenciu na pohodlnú prácu s náhlavnými súpravami VR, aby sa zabránilo akémukoľvek nepohodliu používateľa pri interakcii vo virtuálnom svete.

Unreal Engine ponúka funkcie navrhnuté špeciálne pre virtuálnu realitu[13, 14]:

- Technológia ray tracing, ktorá zlepšuje realistickosť tieňov a osvetlenia.
- Nástroje na vývoj VR aplikácií. Existujú pripravené šablóny, ktoré umožňujú vývojárom vytvárať VR projekty, čím šetria cenný čas a zdroje.
- Podpora ovládačov rôznych platforiem VR na implementáciu komplexnej mechaniky interakcie.

Unreal Engine ponúka funkcie, ktoré sa v kontexte tejto práce javia ako vhodné na vizualizáciu rozsiahlych experimentálnych údajov v oblasti časticovej fyziky. Vzhľadom na výkonnostné obmedzenia webovej verzie NDMVR pri práci

s veľkými množinami údajov môže byť Unreal Engine účinným riešením tohto problému. Jeho podpora vysokokvalitného vykresľovania a interaktívnych prostredí VR môže poskytnúť stabilnú a realistickú vizualizáciu, ktorá uľahčí podrobnú analýzu údajov a zlepší používateľský zážitok.

### 3.2.3 Výkon Unreal Engine pri práci s veľkým množstvom grafických údajov

Vykresľovanie veľkého množstva prvkov má výrazný vplyv na výkon. Preto sa v tejto časti bude rozoberať niekoľko technológií, ktoré Unreal Engine ponúka na optimalizáciu tohto procesu.

#### Organizácia a správa veľkého množstva údajov

Práca [15] preukázala účinnosť Unreal Engine pri vykresľovaní rozsiahlych fotogrametrických modelov. Autori používajú kombinovanú **octree štruktúru** a **asynchrónne načítanie uzlov**, pričom nadbytočné geometrie sú eliminované na základe vzťahov rodič-dieťa. Tým sa znižuje pamäťová náročnosť a urýchľuje načítanie údajov.

Tieto prístupy sa dajú použiť aj na vizualizáciu jednoduchých, ale masívnych geometrií, kde je potrebné efektívne spravovať len relevantné časti údajov na načítanie a vykresľovanie.

#### Technológia Instanced Static Meshes (ISM)

Unreal Engine poskytuje možnosť používať **Instanced Static Meshes (ISM)** — techniku, ktorá umožňuje vykresliť viacero kópií jednej mriežky (mesh) pomocou jediného volania metódy draw. Takéto inštancovanie výrazne znižuje množstvo výpočtov vykonávaných procesorom, čím sa zvyšuje výkon, najmä pri jednoduchých objektoch, ktoré sa opakujú vo veľkom počte [16].

Na dosiahnutie ešte vyššej efektivity možno použiť **Hierarchical Instanced Static Meshes (HISM)**, ktoré okrem inštanciácie ponúkajú aj podporu LOD (Level of Detail) a efektívne mechanizmy rozmiestňovania objektov (angl. *foliage placement*).

#### Technológia Compute Shaders

Ďalším dôležitým nástrojom sú **Compute Shader-y**, ktoré umožňujú vykonávať výpočty priamo na GPU. V prípade masívnej vizualizácie možno výpočtové sha-

dery použiť napríklad na výpočet polohy, mierky alebo farby inštancií bez potreby aktualizácie údajov prostredníctvom CPU [17].

Tento prístup je výhodný najmä vtedy, ak sa vizualizované hodnoty menia v reálnom čase alebo ak sú výsledkom online spracovania.

## 3.3 Platforma Unity

Táto kapitola poskytuje stručný prehľad o Unity, opisuje jeho základné technologické vlastnosti a grafické možnosti. Osobitná pozornosť je venovaná mechanizmom interakcie vo VR prostredí a technikám efektívneho vykresľovania veľkého množstva objektov, ako sú GPU instancing a compute shader-y. Unity je známe svojou flexibilitou a jednoduchým rozhraním, ktoré zjednodušuje vývoj interaktívnych 3D aplikácií.

Takisto sa stručne analyzuje potenciál Unity ako riešenia, ktoré dokáže zabezpečiť stabilnú vizualizáciu experimentálnych údajov.

### 3.3.1 Úvod do nástroja Unity

Unity je výkonný herný engine a vývojová platforma na vytváranie 2D, 3D aplikácií, aplikácií pre rozšírenú realitu (AR) a virtuálnu realitu (VR). Vďaka používateľsky prívetivému rozhraniu a prispôsobivosti môžu vývojári vytvárať dynamické prostredia pre rôzne platformy vrátane systémov Windows, macOS, Android a iOS. Unity poskytuje kompletnú sadu nástrojov vrátane fyzikálneho enginu, animačného systému a rozsiahlej knižnice zdrojov a zásuvných modulov, ktoré zjednodušujú proces vývoja [18]. Vďaka podpore vykresľovania v reálnom čase a dynamickej tvorby scén je vhodným nástrojom pre projekty vyžadujúce flexibilitu a škálovateľnosť.

### 3.3.2 Používanie nástroja Unity vo virtuálnej realite

Unity je veľmi populárnym nástrojom na vytváranie interaktívnych 3D aplikácií vrátane projektov virtuálnej reality vďaka svojej flexibilite a podpore viacerých platforiem [7]. Jeho možnosti umožňujú vytvárať trojrozmerné prostredia intuitívnym a zároveň funkčným spôsobom.

#### Technická implementácia VR v Unity

Moderný vývoj VR aplikácií v Unity je založený na XR Interaction Toolkit, modulárnom frameworku, ktorý umožňuje implementovať základnú interakciu vo vir-

tuálnom prostredí [19]. Medzi hlavné komponenty patria:

1. **XR Rig** - virtuálna reprezentácia používateľa vrátane kamery (pohľad prvej osoby) a ovládačov (ruky)
2. **Interactor / Interactable** - systém na spracovanie interakcií s objektmi (uchopenie, kliknutie, pohyb)
3. **Poskytovateľ teleportácie** - na realizáciu pohybu používateľa v priestore bez potreby fyzického pohybu
4. **XR Ray Interactor** - nástroj na interakciu s používateľským rozhraním (UI) pomocou laserového kurzora

Na implementáciu ovládania pohybu a manipulácie s objektmi sa používajú štandardné komponenty Unity: Rigidbody, Collider, XR Grab Interactable [19]. Dôležité je tiež nastaviť vrstvy, aby sa obmedzili dostupné oblasti pre pohyb a interakciu.

Pri vývoji VR aplikácií sa aktívne používa plátno Canvas s panelmi informácií, úloh alebo testov, ktoré môže používateľ aktivovať alebo prechádzať [19].

### 3.3.3 Výkon Unity pri práci s veľkým množstvom grafických dát

Unity poskytuje viacero mechanizmov na optimalizáciu práce s veľkým množstvom grafických objektov rovnakého typu. Hlavným problémom pri vykresľovaní veľkého počtu primitív (napríklad kociek) je zaťaženie CPU v dôsledku množstva volaní metódy draw a zároveň limity pamäte GPU.

#### Technológia GPU Instancing

**GPU instancing** umožňuje vykresliť veľké množstvo kópií jednej siete (mesh) v rámci jedného kresliaceho volania (draw call), pričom každá inštancia môže mať vlastné parametre, ako sú pozícia, mierka či farba. Tým sa výrazne znižuje počet volaní grafického API a znižuje sa záťaž na CPU [20].

V Unity sa táto technika realizuje prostredníctvom metódy `Graphics.DrawMeshInstanced`, ktorá prijíma pole transformačných matíc (`Matrix4x4`). Ďalšie individuálne parametre, ako napríklad farby, možno odovzdávať pomocou `MaterialPropertyBlock`. Shader musí podporovať inštancovanie pomocou `UNITY_INSTANCING_BUFFER` [21].

#### Obmedzenia a rozšírenia

Metóda `DrawMeshInstanced` je obmedzená na maximálne **1023 inštancií** na jedno volanie. Pre väčšie množstvo objektov je vhodné použiť metódu `DrawMeshInstancedIndirect`, ktorá umožňuje vykreslenie ľubovoľného počtu objektov pomocou `ComputeBuffer`, a to v jednom volaní [21].

#### Technológia Compute Shaders

**Compute shader-y** umožňujú vykonávať paralelné výpočty na GPU, ako napríklad spracovanie údajov pred vykresľovaním, filtrovanie, triedenie či výpočet mierky objektov. Sú užitočné najmä pri práci s dátami, ktoré majú priestorovú štruktúru, ako napríklad 3D matice alebo mračná bodov [22].

Príkladom použitia je rozšírenie FastPoints pre Unity, ktoré vizualizuje rozsiahle mračná bodov s desiatkami až stovkami miliónov bodov. Využíva `ComputeBuffer` a budovanie mimo-jadrovej (out-of-core) octree štruktúry na dynamické vykresľovanie len viditeľných častí scény [23].

#### Výhody GPU instancing a Compute shaderov

Kombinácia GPU instancingu a compute shaderov umožňuje:

1. výrazne znížiť počet volaní metódy draw,
2. vyhnúť sa vytváraniu tisícov samostatných objektov (GameObject),
3. škálovať vykresľovanie na státisíce objektov bez výrazného poklesu výkonu,
4. priamo odovzdávať individuálne vlastnosti (napr. veľkosť, farbu) pre každú inštanciu priamo do GPU.

## 3.4 Faktory ovplyvňujúce výkon vizualizácie

Okrem technológií navrhnutých pre konkrétne herné enginy je potrebné upozorniť na faktory, ktoré ovplyvňujú výkon bez ohľadu na vývojové prostredie.

Pochopenie základných prvkov, ktoré ovplyvňujú vizuálnu zložitosť, je prvým krokom pri vytváraní efektívneho vykresľovacieho prostredia v herných motoroch. Kľúčové komponenty určujú vzhľad a výkonnosť prostredia, ktoré sa môže pohybovať od základných prototypov až po komplexné scenérie vhodné pre film.

Podobne ako digitálne tapety, textúry fungujú ako vizuálne obaly vecí a materiály určujú, ako tieto predmety interagujú so svetlom, aby určili vlastnosti, ako je nepriehľadnosť alebo priehľadnosť [24]. Napriek tomu, že majú rovnaké textúry, rôzne materiály môžu na rôzne osvetlenie reagovať odlišne, čo vytvára rôzne vizuálne efekty.

Pri vytváraní prostredia je osvetlenie nevyhnutné na dosiahnutie rovnováhy medzi realizmom a viditeľnosťou. Herné rámce ponúkajú širokú škálu možností osvetlenia, od dynamických bodových svetiel, ktoré môžu meniť farbu a polohu, až po stacionárne bodové svetlá, ktoré imitujú pevné zdroje osvetlenia. Napriek tomu si dynamickejšie techniky osvetlenia zvyčajne vyžadujú väčší výpočtový výkon, takže ich použitie treba starostlivo zvážiť.

## 3.5 Práca s údajmi: formáty a spôsoby načítavania

Táto časť sa zaoberá možnosťami načítavania údajov a opisuje formát `JSON`, v ktorom sa údaje zadávajú na vizualizáciu histogramov.

### 3.5.1 Spôsoby načítavania údajov

Pre aktuálnu úlohu je dôležité preskúmať a nájsť pohodlný spôsob sťahovania údajov, ktorý by zohľadňoval potreby používateľov, najmä výskumných pracovníkov, ktorí často pracujú s veľkými súbormi údajov. Preto sa zvažovalo niekoľko prístupov k načítavaniu údajov.

#### Lokálne načítavanie údajov

Najjednoduchšou možnosťou je načítanie údajov z lokálneho súboru, napríklad CSV alebo JSON. Jej výhoda spočíva v jednoduchosti implementácie a minimálnych technických požiadavkách - obe prostredia (Unreal Engine a Unity) dokážu so súbormi pracovať rovnako. Absencia potreby nadväzovať ďalšie pripojenie k serveru robí tento prístup atraktívnym najmä pri testovacích scenároch a lokálnom overovaní.

Tento spôsob má však vážnu nevýhodu: ak používateľ pracuje s veľkou databázou, bude musieť zakaždým ručne nahrávať súbory, čo výrazne komplikuje pracovný proces.

#### Načítavanie údajov zo servera

Efektívnejším a atraktívnejším riešením je načítanie údajov zo servera. V prípade veľkých objemov dát sa často pristupuje k využitiu servera, čo zjednodušuje ich správu a odstraňuje potrebu ručného nahrávania súborov. Preto sme zvažovali dva spôsoby prenosu údajov: HTTP a WebSocket.

#### HTTP požiadavky

HTTP požiadavky sú štandardnou a široko používanou metódou získavania údajov zo servera. Sú vhodné na statickú analýzu, keď je potrebné jednorazovo stiahnuť určitý súbor údajov.

Táto metóda má však obmedzenia, ak potrebujete postupne získavať nové údaje. Pre každý nový histogram bude musieť používateľ poslať novú HTTP požiadavku, čo spôsobuje, že proces analýzy údajov nie je taký pohodlný, ako by mohol byť.

#### WebSocket

WebSocket rieši problém viacnásobných požiadaviek tým, že umožňuje nadviazať trvalé spojenie so serverom. Používateľ sa raz prihlási na server WebSocket a údaje sa automaticky prenášajú v reálnom čase bez potreby posielať zakaždým novú požiadavku.

Tento prístup je užitočný najmä vtedy, keď potrebujete sledovať dynamické zmeny. Napríklad počas experimentu, kde sa parametre menia každú sekundu, WebSocket umožňuje okamžite zobraziť tieto zmeny bez oneskorenia.

Vo všeobecnosti platí, že na statickú analýzu je najvhodnejšia požiadavka HTTP, zatiaľ čo WebSocket je efektívnejší na priebežné aktualizácie údajov.

### 3.5.2 Formát vstupných údajov

Údaje sa prijímajú vo formáte objektu JSON (štruktúru si môžete pozrieť na tomto odkaze [25]). Na ich spracovanie je potrebné implementovať modul na spracovanie údajov, ktorý extrahuje kľúčové parametre potrebné na zostavenie histogramov.

Napriek tomu, že objekt JSON obsahuje množstvo metainformácií, pre aktuálnu úlohu potrebujeme len nasledujúce hodnoty:

- fXaxis->fNbins - počet binov histogramu pozdĺž osi X
- fYaxis->fNbins - počet binov histogramu pozdĺž osi Y

- fZaxis->fNbins - počet binov histogramu pozdĺž osi Z
- fArray - pole číselných hodnôt, kde každý prvok zodpovedá výške (alebo frekvencii) príslušného binu histogramu

Tento minimálny súbor údajov postačuje na vizualizáciu histogramov so statickou šírkou binov. V opačnom prípade by boli potrebné aj ďalšie údaje. Po spracovaní sa hodnoty odovzdajú komponentu zodpovednému za vizualizáciu.

# 4 Analýza porovnávacích štúdií výkonu grafických enginov

Táto kapitola sa venuje analýze existujúcich štúdií, ktoré porovnávajú výkon grafických rámcov Unreal Engine a Unity. Ich závery môžu pomôcť identifikovať silné a slabé stránky týchto nástrojov. Zároveň sa však výsledky môžu líšiť v závislosti od typu projektu a konkrétnej aplikácie - najmä pri vizualizácii veľkého množstva údajov, na ktorú sa zameriava táto práca.

Do analýzy boli zahrnuté dve štúdie, ktoré sa priamo zameriavajú na porovnanie týchto dvoch platforiem. Skúmali sa metriky výkonu, ako je rýchlosť snímok (FPS), využitie pamäte (RAM), zaťaženie procesora a kvalita výstupu.

## 4.1 Porovnanie výkonu Unreal Engine a Unity v počítačových hrách

Nedávna výskumná práca[26] Abramoviča a Borchuka z roku 2024 porovnávala výkonnosť Unreal Engine a Unity pri vývoji 3D hier. Výskumníci vytvorili takmer identické hry v oboch enginoch a testovali ich na dvoch rôznych počítačových konfiguráciách, pričom analyzovali rôzne výkonnostné ukazovatele vrátane počtu snímok za sekundu, zaťaženia procesora, využitia pamäte a požiadaviek na spracovanie grafiky.

Výskum ukázal, že Unreal Engine neustále spotrebúva viac výpočtových zdrojov ako Unity. Na prvom testovacom systéme využíval Unreal Engine výrazne viac pamäte, konkrétne 588 MB dodatočnej pamäte RAM a o 2 844 MB grafickej pamäte, ako Unity. Druhý testovací systém vykazoval podobný priebeh: Unreal Engine potreboval o 409 MB viac pamäte RAM a o 2 796 MB viac grafickej pamäte.

Hoci Unreal Engine vo väčšine testovacích scenárov generoval nižšie snímkové frekvencie, bol stabilnejší s menším kolísaním snímkovej frekvencie. Výsledky využitia procesora boli menej jednoznačné: Unreal Engine vykazoval nižšie požiadavky na CPU na jednom systéme, ale vyššie využitie na inom, čo naznačuje,

že jeho správa zdrojov sa môže líšiť v závislosti od konkrétnych hardvérových konfigurácií.

Záverom tohto článku je, že hoci Unreal Engine poskytuje výkonné možnosti na vytváranie vizuálne zložitých aplikácií, jeho výrazne vyššie požiadavky na zdroje môžu byť problémom pre vývojárov, ktorí pracujú s menej výkonným hardvérom alebo sa snažia maximalizovať efektivitu zdrojov.

Preto sa oplatí venovať pozornosť optimalizácii Unreal Engine a zvážiť vplyv údajov na stabilitu vizualizácie.

## 4.2 Porovnanie výkonu vykresľovania animačných sekvencií v Unity a Unreal Engine

Tuomas Vuorinen v svojej bakalárskej práci[27] skúmal výkonnosť Unreal Engine a Unity pri vykresľovaní animačných sekvencií v 3D prostredí. Testovanie sa uskutočnilo na troch rôznych počítačových konfiguráciách. Analyzovali sa tieto parametre: čas vykresľovania, FPS, využitie pamäte, zaťaženie procesora a grafickej karty.

### 4.2.1 Porovnania výkonu

Praca ukázala, že Unreal Engine si vyžaduje viac výpočtových prostriedkov ako Unity. Na najslabšej konfigurácii Unreal Engine generoval výrazne vyššiu kvalitu, ale jeho vykresľovanie trvalo oveľa dlhšie ako v prípade Unity. Napríklad pri rozlíšení 4K a 60 FPS proces vykresľovania v Unreal Engine trval viac ako 48 minút, zatiaľ čo v Unity menej ako 1 minútu. Podobné výsledky sa pozorovali aj na výkonnejších konfiguráciách, kde Unreal Engine podobne vykazoval dlhší čas renderovania a vyššiu spotrebu prostriedkov.

### 4.2.2 Využitie pamäte a CPU

Na všetkých testovacích konfiguráciách Unreal Engine spotreboval viac pamäte RAM a grafickej pamäte. Napríklad na najslabšej konfigurácii Unreal Engine pri vykresľovaní v rozlíšení 4K spotreboval v priemere 30 831 MB pamäte RAM, zatiaľ čo Unity iba 15 520 MB. Využitie CPU sa však v oboch enginoch menilo v závislosti od rozlíšenia. Unity vykazoval nižšie využitie CPU pri nižších rozlíšeniach, zatiaľ čo Unreal Engine bol stabilnejší, hoci náročnejší na zdroje, pri vyšších rozlíšeniach.

### 4.2.3 Stabilita a kvalita výstupu

Unreal Engine generoval animačné sekvencie s výrazne vyššou kvalitou vďaka svojmu pokročilému systému osvetlenia a schopnosti pracovať s dostatočne detailnými modelmi. Unity však vykazoval väčšiu stabilitu a efektívnosť, čo umožnilo rýchlejšie vykresľovanie na slabšom hardvéri. Okrem toho sa kvalita produktov Unity môže porovnávať s Unreal Engine, ak sa viac času venuje optimalizácii osvetlenia a efektov.

### 4.2.4 Analýza výsledkov výskumu

Po analýze tejto práce môžeme konštatovať, že Unreal Engine je vhodnejší pre projekty, ktoré vyžadujú vysokú vizuálnu kvalitu a môžu si dovoliť vyššie hardvérové požiadavky. A Unity ponúka efektívnejšie využitie zdrojov a rýchlejšie vykresľovanie.

# 5 Návrh a Implementácia

Táto kapitola sa zaoberá vývojom a implementáciou riešení v dvoch herných prostrediach - Unreal Engine a Unity. Hlavným cieľom je vytvoriť jednotný prístup k vizualizácii experimentálnych údajov, ktorý umožní objektívne porovnať výkonnosť oboch platforiem.

V tejto kapitole sa bude analyzovať:

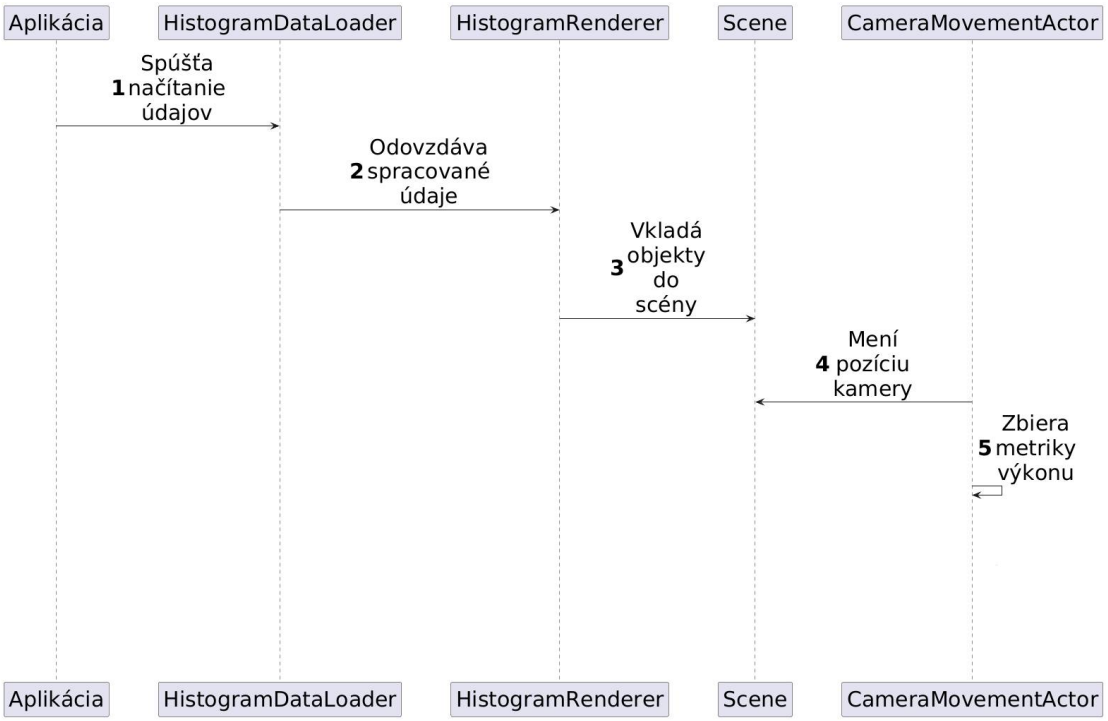
- Architektúra riešenia — štruktúra a princípy návrhu systému v oboch prostrediach.
- Výber technológií a nástrojov — zdôvodnenie použitých prístupov na výmenu údajov, organizáciu scény, spracovanie vykresľovania a ďalšie aspekty implementácie.
- Proces implementácie — technické aspekty realizácie, hlavné výzvy a špecifiká vývoja v jednotlivých prostrediach.

## 5.1 Architektúra riešení

Architektúra tejto práce bola inšpirovaná existujúcou webovou implementáciou, ktorá bola vytvorená ako prvá verzia systému v prostredí A-Frame. Webová verzia bola funkčná, ale niektoré časti systému neboli explicitne oddelené ako samostatné komponenty. V niektorých prípadoch boli časti logiky, ako napríklad načítanie údajov a ich vizualizácia, implementované spoločne v rámci jedného skriptu. Tento prístup fungoval, ale mohol sťažiť ďalšie rozširovanie a údržbu systému.

V prípade implementácií pre Unreal Engine a Unity boli tieto časti oddelené do samostatných modulov - nielen kvôli lepšej organizácii kódu a porovnateľnosti platforiem, ale aj preto, že to bolo praktickejšie a jednoduchšie ako kopírovať prístup z webovej verzie.

Sekvenčný diagram na obr. 5.1 znázorňuje spôsob interakcie medzi hlavnými komponentmi systému v rámci architektúry riešenia.




Obrázok 5.1: Sekvenčný diagram — Princíp interakcie komponentov

- **HistogramDataLoader** — Zodpovedá za čítanie údajov. Prijaté údaje spracúva a odosiela ich v pripravenej forme do HistogramRenderer na ďalšiu vizualizáciu.

- **HistogramRenderer** — Zodpovedá za generovanie objektov histogramu na scéne na základe údajov poskytnutých HistogramDataLoaderom. Mení veľkosť, farbu a v prípade potreby aj tvar týchto objektov podľa hodnôt údajov. V 1D a 2D histogramoch sú objekty sú vizualizované ako stĺpce, v 3D ako kocky v priestore XYZ. Ďalšie rozšírenie komponentu umožní používať rôzne geometrické tvary na vizualizáciu viacrozmerných údajov.

- **Scene** - prostredie (mapa) obsahujúce všetky komponenty. V budúcnosti ju možno nahradiť komponentom SceneManager, ktorý umožní dynamicky meniť tvar scény, ale na testovanie výkonu postačuje statická mapa. Okrem toho obsahuje aj prvky zodpovedné za vizuálne spracovanie scény, pohyb používateľa a podporu prostredia VR. V popise tohto komponentu budú popísané aj základné nastavenia scény, ako je osvetlenie, pozadie a umiestnenie objektov.

- **CameraMovementActor** — Riadi automatický pohyb kamery po definovanej trajektórii v testovacom režime. Umožňuje objektívne hodnotenie výkonu systému v rôznych podmienkach. Zhromažďuje tiež ukazovatele vý-

konnosti, ktoré charakterizujú výkonnosť systému.

## 5.2 Návrh komponentu HistogramDataLoader

Ako bolo uvedené v predchádzajúcej kapitole, HistogramDataLoader je zodpovedný za čítanie, spracovanie a prenos údajov do komponentu HistogramRenderer. V tejto časti sa bližšie pozrieme na to, ako táto zložka funguje, ako získať údaje a ako ich pripraviť pred prenosom do vizualizácie.

### 5.2.1 Načítavanie údajov

Po analýze metód načítavania údajov v odseku 3.5.1 boli vybrané metódy **HTTP** a **Websocket**.

Pri použití protokolu HTTP sa údaje načítajú priamo do nástroja HistogramDataLoader. V prípade WebSocket bol vytvorený samostatný modul **Broker**, ktorý je zodpovedný za vytvorenie spojenia, prijímanie údajov a ich následný prenos do HistogramDataLoader. Tento prístup umožňuje lepšie oddelenie povinností a umožňuje opätovné použitie modulu Broker v iných častiach systému.

### 5.2.2 Spracovanie údajov

Po prijatí údajov je ďalším krokom ich spracovanie. Teoreticky by sa tejto fáze dalo vyhnúť, keby sa použili objekty ROOT - štruktúry, ktoré obsahujú všetky potrebné informácie a pracujú s knižnicou ROOT. Táto knižnica je však kompatibilná len s jazykom C++, takže by sa dala použiť len v Unreal Engine, zatiaľ čo Unity a webová platforma by vyžadovali samostatné spracovanie. Keďže cieľom tejto práce je implementovať jednotné riešenie pre všetky prostredia, tento prístup bol zamietnutý. Ak výsledky práce ukážu, že Unreal Engine je najlepšie riešenie, bude vhodné sa k tejto otázke vrátiť.

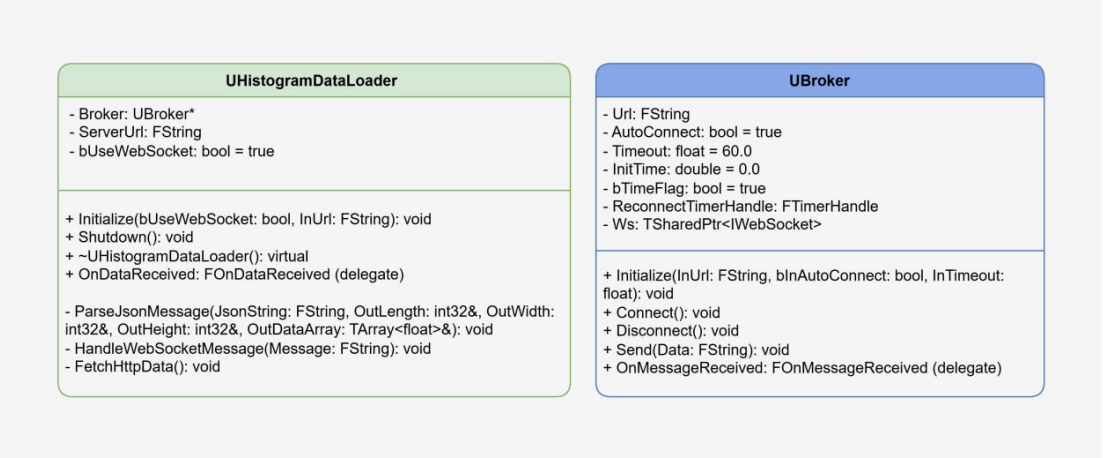
Preto bude súčasťou tohto komponentu analyzátor súborov JSON, ktorý získa potrebné údaje na generovanie histogramov, ktoré sú špecifikované v časti 3.5.2, a odošle ich komponentu HistogramRenderer na vykreslenie.

### 5.2.3 Implementácia HistogramDataLoader v Unreal Engine

Teraz prejdime k implementácii HistogramDataLoader v Unreal Engine. Reprezentujú ho triedy UHistogramDataLoader a UBroker. V tejto časti sa budeme zaoberať všetkými ich funkciami a ich implementáciou v logickom poradí.

### Deklarácia tried

Triedy UHistogramDataLoader a UBroker obsahujú nasledujúce metódy, ktoré sú znázornené na obrázku 5.2.




Obrázok 5.2: UML diagramy tried UHistogramDataLoader a UBroker

Tieto metódy sú zodpovedné za inicializáciu tried, prijímanie údajov cez HTTP alebo WebSocket, spracovanie odpovedí servera a parsovanie JSON. Postupne sa pozrime na ich implementáciu.

### Inicializácia

Prvým krokom operácie `UHistogramDataLoader` je zavolanie metódy `Initialize`, ktorá preberá dva parametre:

- `bUseWebSocket` (`bool`) – určuje, či sa použije pripojenie WebSocket.
- `ServerUrl` (`FString`) – adresa URL servera, z ktorého sa budú prijímať údaje.

Výsledné parametre sa uložia do premenných triedy na ďalšie použitie pri nastavovaní pripojenia. Potom sa vytvorí spojenie so serverom, ktoré bolo určené na základe týchto údajov.

Ak je aktivovaný režim `WebSocket`, komponent vytvorí novú inštanciu modulu `Broker`, ktorý zabezpečuje samotnú komunikáciu so serverom. Modul `Broker` sa inicializuje so zadanou adresou a štandardným časovým limitom a potom sa k nemu zaregistruje obsluha `HandleWebSocketMessage`, ktorá odpovedá na prichádzajúce správy. Prijaté údaje sa potom spracujú a odovzdajú na vizualizáciu.

Ak nie je povolený režim WebSocket, údaje sa získavajú pomocou protokolu HTTP, pričom sa volá metóda `FetchHttpData`, ktorá odošle požiadavku a spracuje odpoveď.

#### Načítavanie údajov cez HTTP

Ak parameter `bUseWebSocket` je nastavený na `false`, komponent `UHistogramDataLoader` vykoná načítanie údajov pomocou HTTP protokolu. Unreal Engine na tento účel poskytuje modul `HTTP`, ktorý umožňuje vytváranie asynchrónnych požiadaviek typu REST.

Na vytvorenie požiadavky sa používa objekt typu `TSharedRef<IHttpRequest>`, ktorý sa vytvára volaním metódy `FHttpModule::Get().CreateRequest()`. Tento objekt je následne potrebné nakonfigurovať pred jeho odoslaním.

V rámci konfigurácie sa nastavujú tieto základné parametre:

- `SetURL` — nastavuje cieľovú URL adresu, ktorú určuje parameter `ServerUrl`.
- `SetVerb` — definuje typ HTTP metódy, v tomto prípade `"GET"`.
- `SetHeader` — nastavuje hlavičku požiadavky, napríklad `Content-Type: application/json`, čo signalizuje, že klient očakáva odpoveď vo formáte JSON.

Na spracovanie odpovede sa používa spätné volanie `OnProcessRequestComplete`, ku ktorej je priradená lambda funkcia. Táto funkcia overí úspešnosť požiadavky, skontroluje platnosť odpovede a extrahuje prijaté údaje vo forme reťazca. Následne sa reťazec odovzdá metóde `ParseJsonMessage` na spracovanie údajov.

Po úspešnom spracovaní JSON správy sa extrahované hodnoty odosielajú prostredníctvom delegátu `OnDataReceived` na ďalšie spracovanie a vizualizáciu.

#### Načítavanie údajov cez WebSocket

Ak parameter `bUseWebSocket` je nastavený na `true`, na získavanie údajov sa použije protokol WebSocket.

Na rozdiel od HTTP, ktorý má v Unreal Engine vstavanú podporu cez modul `HTTP`, protokol WebSocket si vyžaduje manuálnu aktiváciu pluginu *Experimental WebSocket Networking Plugin*. Tento plugin nie je súčasťou základnej distribúcie a je potrebné ho zapnúť v nastaveniach projektu pred použitím WebSocket komunikácie.

Samotná komunikácia WebSocket bola v rámci tejto práce presunutá do samostatného modulu **Broker**, ktorý zabezpečuje nadviazanie spojenia, prijímanie správ, správu odpojenia a prípadné opätovné pripojenie. Tento prístup umožňuje zjednodušiť komponent HistogramDataLoader a zároveň zvýšiť znovupoužiteľnosť komunikačnej logiky.

Broker sa vytvára pri inicializácii komponentu HistogramDataLoader, ak je aktivovaný WebSocket režim. Metóda `Broker::Initialize` prijíma parametre adresy servera, informáciu o automatickom pripájaní a maximálny časový limit pre opakované pokusy o pripojenie.

Interná metóda `Connect` vytvára nového klienta pomocou rozhrania `FWebSocketsModule::Get().CreateWebSocket(Url)` a definuje niekoľko spätných volaní pomocou anonymných funkcií:

- `OnConnected` — aktivuje sa po úspešnom pripojení. Slúži na logovanie a zrušenie prípadného časovača pre opätovné pripojenie.
- `OnConnectionError` — aktivuje sa pri chybe spojenia. V prípade, že ešte nevypršal definovaný časový limit, pokus o pripojenie sa zopakuje po krátkej prestávke.
- `OnClosed` — spúšťa sa pri ukončení spojenia. Jeho správanie je analogické ako pri chybe – pokúsi sa o opätovné pripojenie, ak je to povolené.
- `OnMessage` — reaguje na každú prichádzajúcu správu zo servera. Táto správa sa následne odosiela HistogramDataLoaderu, ktorý ju spracuje prostredníctvom funkcie `HandleWebSocketMessage`.

Všetky prijaté správy zo servera sú prostredníctvom udalosti `OnMessageReceived` odovzdané späť do komponentu `HistogramDataLoader`, ktorý ich spracováva metódou `HandleWebSocketMessage`. Táto metóda najprv overí, či správa nie je prázdna, a následne ju odovzdá metóde `ParseJsonMessage`. Po úspešnom spracovaní údajov sú hodnoty odoslané ďalej cez delegát `OnDataReceived` na vizualizáciu.

Pri ukončení programu alebo uvoľnení objektu `UHistogramDataLoader` sa automaticky uzatvára aj WebSocket spojenie volaním metódy `Disconnect` z modulu Broker, ktorá zabezpečuje korektné ukončenie komunikácie so serverom, ak je spojenie aktívne.

#### Spracovanie JSON správ

Spracovanie údajov v Unreal Engine zabezpečuje spoločná metóda `ParseJsonMessage`, ktorá je volaná po prijatí údajov z HTTP alebo WebSocket spojenia. Táto metóda vykonáva dekódovanie reťazca vo formáte JSON a z neho extrahuje potrebné informácie.

Na prácu s objektmi JSON boli použité knižnice `JsonReader` a `JsonSerializer`, ktoré umožňujú konvertovať reťazec typu `FString` na objekt `FJsonObject`. Implementácia používa `TSharedRef<TJsonReader<>>` na prijatie správy JSON a potom získa požadované polia pomocou funkcií, ako sú `GetObjectField`, `GetIntegerField` alebo `TryGetArrayField`.

#### Prenos údajov

Po úspešnom spracovaní JSON správy metódou `ParseJsonMessage` sú extrahované údaje — rozmery histogramu a pole hodnôt — odoslané pomocou dynamického delegátu `OnDataReceived`. Tento delegát má štyri parametre: dĺžku, šírku, výšku a pole typu `TArray<float>`. Delegát je deklarovaný pomocou `DECLARE_DYNAMIC_MULTICAST_DELEGATE_FourParams`, čo umožňuje viacerým komponentom prihlásiť sa na túto udalosť a reagovať na prichádzajúce dáta.

Použitie delegátu umožňuje oddeľovať zodpovednosť za načítavanie údajov od vizualizačných komponentov. Každý komponent, ktorý sa prihlási na odber tejto udalosti, môže reagovať vlastným spôsobom, čím sa zvyšuje modularita a flexibilita systému.

### 5.2.4 Implementácia HistogramDataLoader v Unity

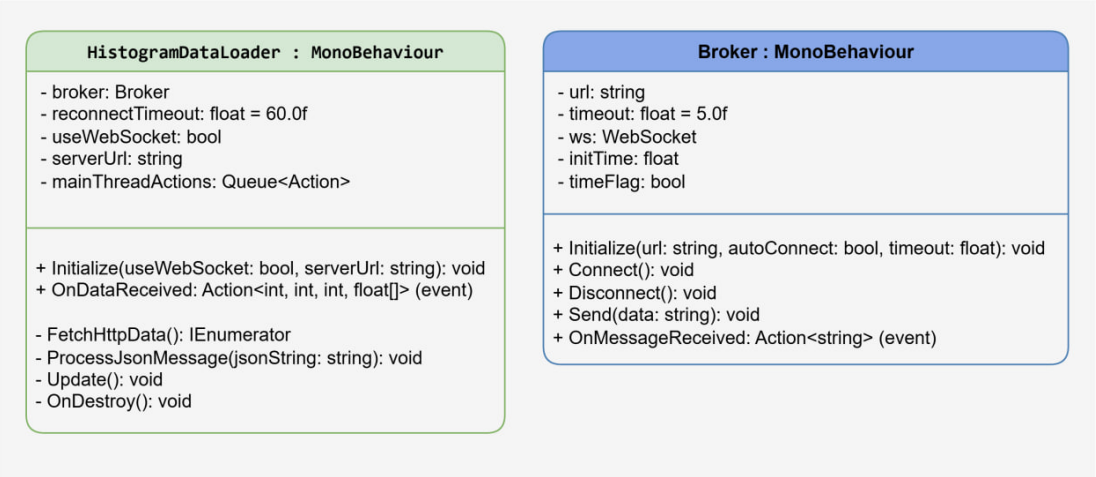
Komponent `HistogramDataLoader` v prostredí Unity má veľmi podobnú funkciu ako `UHistogramDataLoader` v Unreal Engine. V oboch prípadoch ide o modul, ktorý zabezpečuje načítanie údajov zo servera (cez HTTP alebo WebSocket), ich spracovanie vo formáte JSON a následné odoslanie pripravených údajov ďalším častiam systému.

Preto sa v tejto časti sústredíme najmä na rozdiely medzi implementáciami v týchto dvoch prostrediach, pričom sa vynechajú tie časti, ktoré fungujú analogicky.

#### Deklarácia tried

Komponent `HistogramDataLoader` v Unity je reprezentovaný triedami HistogramDataLoader a Broker, ktoré obsahujú nasledujúce hlavné metódy, znázor-

nené na obrázku 5.3.




Obrázok 5.3: UML diagramy tried HistogramDataLoader a Broker

### Inicializácia

Inicializácia sa vykonáva metódou `Initialize`, ktorá prijíma dva parametre: `useWebSocket` (typ boolean) a `serverUrl` (typ string). Tieto hodnoty sa uložia ako interné premenné. Následne sa podľa hodnoty `useWebSocket` rozhodne, či sa spustí coroutine pre HTTP požiadavku alebo sa vytvorí WebSocket pripojenie.

### Načítavanie údajov cez HTTP

Na komunikáciu cez HTTP sa používa trieda `UnityWebRequest` z modulu `UnityEngine.Networking`. Požiadavka sa vytvorí metódou `UnityWebRequest.Get(serverUrl)`, nastaví sa hlavička `"Content-Type"` na `"application/json"` a spustí sa coroutine `FetchHttpData` pomocou `StartCoroutine`.

Coroutine čaká na dokončenie požiadavky pomocou `yield return request.SendWebRequest()`. Po prijatí odpovede sa overí, či bola požiadavka úspešná pomocou `request.result == UnityWebRequest.Result.Success`. V prípade úspechu sa získaný reťazec `request.downloadHandler.text` odovzdá na spracovanie metóde `ProcessJsonMessage`, ktorá zabezpečí extrakciu potrebných údajov. Tie sú následne odoslané cez udalosť `OnDataReceived`.

#### Načítavanie údajov cez WebSocket

Pre WebSocket komunikáciu sa na rozdiel od Unreal Engine používa knižnica `WebSocketSharp`. Obsluha komunikácie je implementovaná v samostatnom komponente `Broker`, podobne ako v Unreal Engine.

Aj tu sa využívajú spätné volania ako `OnOpen`, `OnMessage`, `OnError` a `OnClose`, ktoré spracúvajú udalosti súvisiace s komunikáciou. Rozdielom je, že v Unity je nutné všetku interakciu s komponentmi vykonávať v hlavnom vlákne. Z tohto dôvodu sa volanie udalosti
`OnDataReceived` zapisuje do fronty `mainThreadActions` a spracúva sa v metóde `Update()`.

Rovnako ako v Unreal Engine, ani v Unity nie je WebSocket podpora súčasťou základnej distribúcie, preto bolo potrebné manuálne pridať externú knižnicu cez správcu balíkov.

Pri integrácii knižnice `WebSocketSharp` sa vyskytli technické problémy so správcom balíkov vo Visual Studiu. Knižnicu nebolo možné úspešne pridať priamo do existujúceho projektu. Jediným fungujúcim riešením bolo vytvoriť nový čistý projekt v Visual Studiu, pridať do neho knižnicu a potom ručne skopírovať vygenerovaný súbor `.dll` do priečinka `Plugins` v pôvodnom projekte.

#### Spracovanie JSON správ

Prijaté údaje vo formáte `string` sa spracúvajú metódou `ProcessJsonMessage`, ktorá používa knižnicu `Newtonsoft.Json`. Pomocou `JObject.Parse` sa vytvorí strom objektov, z ktorého sa extrahujú polia: `fXaxis`, `fYaxis`, `fZaxis` a pole `fArray`.

Keďže štruktúra JSON zodpovedá výstupu z ROOT formátu, pri špeciálnych prípadoch (napríklad typ `TObjArray`) sa prepne na druhý objekt v poli, rovnako ako v Unreal Engine. Po spracovaní sa údaje odosielajú cez `OnDataReceived` ďalšiemu komponentu na vizualizáciu.

#### Prenos údajov

Spracované údaje sa odosielajú prostredníctvom udalosti `OnDataReceived`, ktorá má štyri parametre: dĺžku, šírku, výšku a pole typu `float[]`. Keďže udalosti v Unity musia byť volané z hlavného vlákna, volanie tejto udalosti je zapísané do fronty `mainThreadActions` a vykonané v metóde `Update()`.

Tento prístup zabezpečuje, že komponenty vizualizácie môžu bezpečne reagovať na nové údaje bez rizika porušenia vlákien. Architektúra zostáva flexibilná

a oddelená — načítanie údajov je nezávislé od ich prezentácie.

## 5.3 Návrh komponentu HistogramRenderer

Ako bolo uvedené v predchádzajúcich častiach, HistogramRenderer je komponent zodpovedný za zobrazenie údajov vo forme histogramu. Vstupné údaje dostáva z komponentu HistogramDataLoader, ktorý ich získava zo servera a pripravuje ich na vizualizáciu.

Úlohou HistogramRenderer-a je premeniť tieto údaje na súbor 3D objektov, ktorých poloha, veľkosť a farba závisí od hodnoty každého jednotlivého binu. Keďže cieľom tejto práce je porovnať výkon pri zobrazovaní veľkého množstva údajov, komponent bol navrhnutý s dôrazom na optimalizáciu renderovania a škálovateľnosť.

### 5.3.1 Prijatie a interpretácia údajov

HistogramRenderer funguje ako pasívna vizualizačná zložka, ktorá očakáva príchod údajov vo forme udalosti z komponentu HistogramDataLoader. Tieto údaje obsahujú rozmery histogramu v troch osiach (šírka, dĺžka, výška) a jednorozmerné pole hodnôt, ktoré reprezentujú výšku alebo intenzitu jednotlivých binov.

Po prijatí údajov komponent spustí proces generovania vizualizačných prvkov. Každý bin sa reprezentuje ako trojrozmerný objekt (napr. kocka alebo obdĺžnikový hranol), ktorého:

- pozícia je odvodená z jeho indexu v 3D mriežke,
- veľkosť (výška kvádra alebo dĺžka steny kocky) je úmerná jeho hodnote,
- farba je určená podľa škály závislej od rozsahu hodnôt v celom histograme.

Komponent je navrhnutý tak, aby mohol spracovať aj veľmi veľké objemy údajov (desiatky až stovky tisíc binov), pričom používa optimalizačné techniky vizualizácie, ktoré minimalizujú počet výpočtov a výstupných operácií. Pre každý bin sa vypočítava pozícia a transformácia samostatne, pričom sa využívajú preddefinované funkcie na škálovanie hodnôt a výpočet farby.

Na rozdiel od komponentov, ktoré sa aktualizujú neustále v reálnom čase, HistogramRenderer reaguje len na nové dáta. To znamená, že vizualizácia je vždy výsledkom jednej dávky údajov, čo znižuje systémové nároky a eliminuje potrebu priebežného monitorovania zmien.

Nasledujúce časti popisujú konkrétne kroky procesu generovania geometrie a optimalizačné opatrenia, ktoré boli implementované pre zachovanie výkonu pri veľkom množstve prvkov.

Na účely zjednodušenia a sprehľadnenia implementácie boli všetky výpočtové funkcie, ako je určenie pozície, veľkosti a farby binu, centralizované do pomocnej triedy `HistogramRenderUtils`. Tento modul je zdieľaný v oboch implementáciách (v Unreal Engine aj v Unity) a slúži ako knižnica na predspracovanie vizualizačných parametrov, čo zvyšuje prehľadnosť a modularitu kódu.

### 5.3.2 Farebné a veľkostné mapovanie hodnôt

Vizualizácia histogramu je založená na premene číselných hodnôt na geometrické a vizuálne vlastnosti jednotlivých binov. Dve kľúčové vlastnosti, ktoré sa pre každý bin určujú, sú jeho veľkosť (mierka) a farba. Obe tieto vlastnosti sú odvodené od samotnej hodnoty binu a od celkového rozsahu hodnôt v histograme.

Aby bolo možné určiť mierku a farbu konzistentne, je potrebné poznať minimálnu a maximálnu hodnotu v poli údajov. Tieto hodnoty sa zistia pred samotným generovaním geometrie a slúžia na normalizáciu. Normalizovaná hodnota $v_{norm}$ sa vypočíta podľa vzorca:

$$v_{norm} = \frac{v - v_{min}}{v_{max} - v_{min}}$$

kde $v$ je hodnota binu, $v_{min}$ je minimálna hodnota v poli a $v_{max}$ maximálna. Výsledná hodnota $v_{norm} \in [0, 1]$ sa následne použije ako vstup do funkcie, ktorá určuje:

- veľkosť objektu — v prípade 1D a 2D histogramu sa mení najmä výška, zatiaľ čo pri 3D histogramoch sa mierka aplikuje vo všetkých troch osiach,
- farbu — napríklad prechodom medzi dvoma farbami (napr. modrá pre minimum, červená pre maximum).

Tieto výpočty sú zrealizované v pomocných funkciách modulu `HistogramRenderUtils`. Použitie tejto schémy umožňuje jednotné mapovanie hodnôt na vizuálne prvky, čím sa zaručuje konzistentný vzhľad vizualizácie a zároveň sa podporuje vizuálna interpretácia údajov používateľom.

### 5.3.3 Optimalizácia renderovania

Vzhľadom na to, že cieľom tejto práce je porovnať výkon vizualizácie veľkého množstva údajov, bol komponent HistogramRenderer navrhnutý s dôrazom na

efektivitu renderovania. Pri návrhu boli zohľadnené optimalizačné techniky analyzované v kapitolach 3.2.3 a 3.3.3, pričom najvýznamnejšie z nich boli priamo aplikované do architektúry.

Na zníženie počtu volaní metódy *draw* a odľahčenie CPU sa využíva technika **GPU instancovania**. Vizuálne prvky (biny histogramu) sú generované ako inštancie jedného základného objektu, pričom každá inštancia má vlastnú pozíciu, mierku a farbu. Vďaka tomu sa aj tisíce objektov dajú vykresliť pomocou minimálneho počtu volaní renderovacieho API.

Ďalším optimalizačným prvkom je **asynchrónne spracovanie údajov**. Výpočty transformácií, normalizácie hodnôt a farebného mapovania prebiehajú v samostatnej úlohe mimo hlavného vlákna, čím sa znižuje riziko spomalenia hernej slučky.

Celkový návrh podporuje aj **škálovateľnosť** — komponent je schopný prijať veľké množstvo údajov bez potreby zásahu do štruktúry systému. Takéto riešenie umožňuje vizualizáciu údajov s rôznou granularitou a rozsahom pri zachovaní interaktívneho výkonu.

Konkrétna realizácia týchto techník bude podrobne popísaná v nasledujúcich častiach venovaných implementácii.

Na objektívne porovnanie výkonnosti medzi platformami je spolu s hlavnou optimalizovanou verziou implementovaný testovací režim, v ktorom sú niektoré optimalizačné metódy zámerne vypnuté. V tomto režime sa nepoužíva príkaz `cull distance`, ktorý vypína filtrovanie neviditeľných objektov podľa vzdialenosti a tiež vypína asynchrónne spracovanie údajov. Okrem toho sa v testovacom režime nepoužíva farebné zobrazenie - všetky vizualizované objekty majú jednotnú bielu farbu. Tento upravený režim zabezpečuje rovnaké podmienky pre všetky platformy a umožňuje objektívne vyhodnotiť výkon bez skreslení spôsobených pokročilými optimalizáciami.

### 5.3.4 Implementácia HistogramRenderer v Unreal Engine

Komponent `HistogramRenderer` je reprezentovaný triedou `AHistogramRenderer`. Predpona **`A`** v názve triedy je štandardom v Unreal Engine pre triedy odvodené od `AActor` — teda objekty, ktoré možno umiestniť priamo do scény. V tomto prípade je to nevyhnutné, keďže komponent zabezpečuje priestorové vykresľovanie údajov vzhľadom na svoju pozíciu v scéne. Keďže trieda obsahuje veľké množstvo premenných a metód, namiesto ich úplného výpisu sú jednotlivé časti popísané postupne podľa konkrétnej funkcionality, v súvislosti s ktorou sa používajú. Medzi hlavné atribúty patria nastavenia zobrazovania (na-

pr. `BinMesh`, `BinPadding`, `MaxDrawDistance`), parametre pripojenia k serveru (`ServerUrl`, `bUseWebSocket`, `bUseServerData`) a konfiguračné premenné spracovania údajov. Trieda obsahuje aj transformácie a farebné informácie pre jednotlivé biny, ktoré sa generujú dynamicky po prijatí nových dát. Kľúčové metódy, ako napríklad `HandleDataReceived` a `RenderHistogram`, zabezpečujú získavanie údajov a následné vykreslenie inštancií objektov pomocou komponentu `InstancedMesh`. Ich implementácia bude podrobne opísaná v nasledujúcich častiach.

**Inicializácia a konfigurácia**

Pri vytvorení inštancie triedy `AHistogramRenderer` sa najskôr zavolá konštruktor, ktorý inicializuje základné hodnoty a vytvorí renderovací komponent `InstancedMesh` typu `UHierarchicalInstancedStaticMeshComponent`. Tento komponent je následne nastavený ako koreňová komponenta herného objektu a slúži na efektívne vykresľovanie veľkého množstva rovnakých objektov s rôznymi transformáciami.

Ďalšia konfigurácia prebieha vo funkcii `BeginPlay`, kde sa nastavujú parametre ako maximálna vzdialenosť vykresľovania (`MaxDrawDistance`), základný mesh binov (`BinMesh`) a štandardný materiál. Ak je povolený asynchrónny režim vykresľovania, nastaví sa aj príznak `bUseAsyncRendering`, ktorý ovplyvňuje ďalšie spracovanie údajov.

Súčasťou inicializácie je aj príprava vstupných údajov — buď zo servera (ak je `bUseServerData` aktivované), alebo z lokálne definovaných testovacích dát. Táto voľba ovplyvňuje, či bude komponent reagovať na externé vstupy alebo okamžite vykreslí testovaciu scénu.

**Spracovanie prijatých údajov**

Ak je komponent nastavený na prijímanie údajov z servera, metóda `HandleDataReceived` sa zavolá po prijatí údajov z `HistogramDataLoader`. Získané rozmery histogramu (`Length`, `Width`, `Height`) a jednorozmerné pole hodnôt binov vo vnútorných premenných sa uložia na ďalšie použitie .

Vypočítajú sa aj minimálne a maximálne hodnoty z DataArray (jednorozmerného poľa). Tieto hodnoty sú uložené v `MinHistogramValue` a `MaxHistogramValue` a používajú sa na škálovanie veľkosti a určenie farby každého bina. Vlastná vizualizácia sa vykoná až po spracovaní a uložení celého súboru údajov.

### Vykresľovanie histogramu

Na začatie vykresľovania histogramu je potrebné najprv poskytnúť základnú mriežku (mesh). Táto sieť reprezentuje tvar jedného binu a slúži ako šablóna pre všetky ostatné inštancie. Na túto sieť sa aplikuje vlastný materiál, ktorý podporuje tri `CustomDataFloats`, používané na dynamické ovládanie farby každej inštancie v reálnom čase.

Proces vykresľovania sa začína metódou `RenderHistogram`, ktorá najprv overí, že sieť aj komponenta pre inštancovanú sieť sú správne inicializované. Ďalej sa vypočíta počiatočná pozícia a ohraničujúci rámec (bounding box) základnej siete, ktoré sú potrebné pre správne umiestnenie binov.

Komponent potom prejde do vlákna na pozadí, kde sa vypočíta kompletný zoznam inštancií binov. Pre každý bin sa hodnota získa pomocou funkcie `GetBinContent`, ktorá vypočíta lineárny index z 3D súradníc (X, Y, Z) na základe dimenzionality histogramu. Táto funkcia pristupuje k hodnote v jednorozmernom poli hodnôt typu float, ktoré bolo načítané z dátového zdroja. Funkcia `GetBinContent` bola implementovaná ako zjednodušená obdoba podobných metód dostupných v knižnici JSRoot. Ako už bolo spomenuté, táto funkcia by sa dala použiť priamo, ak by bol JSROOT pripojený k projektu.

Ak má bin nenulovú hodnotu, jeho mierka sa vypočíta pomocou `ComputeBinSize` a jeho pozícia pomocou `ComputeBinPosition`. Na biny vo vrstve `Z=0` sa aplikuje špeciálne spracovanie s cieľom zabezpečiť, aby bola prvá vrstva presne zarovnaná so zemou. Toto zarovnanie sa dosiahne identifikovaním najvyššieho binu v základnej vrstve a následným posunom zvyšných vrstiev bez potreby ďalšieho prechodu.

Ak je povolené farebné mapovanie, pre každý bin sa vypočíta farba pomocou `ComputeColor` a uloží sa do paralelného poľa. Konečné transformácie a farby sa potom odovzdajú do herného vlákna na vykreslenie.

### Asynchrónny režim vykresľovania

Keď je komponent nakonfigurovaný na použitie asynchrónneho vykresľovania (`bUseAsyncRendering`), predpočítané dáta inštancií sa nevykreslia naraz. Namiesto toho sa dočasne ukladajú do vyrovnávacích pamätí (`PendingTransforms` a `PendingColors`) a následne sa odosielajú po menších dávkach pomocou metódy `SpawnInstancesStep`. Táto metóda je opakovane volaná pomocou časovača a pridáva pevný počet inštancií za snímku (napr. 100), čo pomáha udržať stabilnú snímkovú frekvenciu aj pri vykresľovaní desiatok až stoviek tisíc objek-

tov. Na spustenie tejto dávkovej aktualizácie slúži pomocná metóda `StartSpawningInstances()`, ktorá najprv okamžite vykreslí prvú dávku a následne nastaví časovač, ktorý v pravidelných intervaloch opakovane volá `SpawnInstancesStep()`. Týmto spôsobom sa zabezpečí plynulý prechod medzi jednotlivými dávkami bez oneskorenia.

Každá inštancia sa pridáva pomocou `AddInstance` a jej farba sa nastavuje cez `SetCustomDataValue` pre tri RGB kanály. Keď sú všetky inštancie spracované, sieť sa označí ako zmenená (dirty), aby sa zabezpečila jej aktualizácia na obrazovke, a vykresľovacia príznaková premenná `bIsGenerating` sa resetuje. Tento prístup zabraňuje blokovaniu hlavného herného vlákna a umožňuje plynulý výkon aj pri vysokej hustote údajov.

#### Finalizácia a čistenie

Po dokončení vykresľovania komponent automaticky resetuje svoj vnútorný stav a vymaže všetky dočasné údaje. To zahŕňa zastavenie asynchrónneho generovacieho časovača, vyčistenie príznaku `bIsGenerating` a vymazanie bufferov `PendingTransforms` a `PendingColors`.

Okrem toho, na konci životného cyklu komponentu (`EndPlay`) sa vykoná korektné uvoľnenie pomocných komponentov, ako napríklad `HistogramDataLoader` alebo `Broker`. Tieto sú najskôr deaktivované prostredníctvom metódy `Shutdown()`, následne označené na zničenie volaním `ConditionalBeginDestroy()` a nastavené na `nullptr`. Tento postup zaručuje bezpečné ukončenie spojení (napríklad WebSocket) a predchádza únikom pamäte alebo nekorektnému ukončeniu procesu.

### 5.3.5 Implementácia HistogramRenderer v Unity

V tejto časti je popísaná implementácia komponentu `HistogramRenderer` v prostredí Unity, pričom dôraz sa kladie len na rozdiely oproti verzii vytvorenej v Unreal Engine. Komponent je reprezentovaný triedou s rovnakým názvom — HistogramRenderer. Identické časti, ktoré fungujú analogicky, nie sú opakovane rozoberané.

#### Inicializácia a konfigurácia

Na rozdiel od implementácie v Unreal Engine, komponent `HistogramRenderer` v Unity nevyužíva špecializovaný inštančný komponent. V rámci inicializácie

sa nastaví základný `Mesh` a materiál, ktoré sú uložené ako verejné premenné a priradené cez editor.

Zatiaľ čo v Unreal stačí zadať iba `StaticMesh`, v Unity je potrebné osobitne definovať aj `Material`, keďže vykresľovanie inštancií prebieha manuálne. Navyše, pre zmenu farby inštancií sa vytvára objekt `MaterialPropertyBlock`, ktorý umožňuje priradiť individuálne vlastnosti každej skupine objektov bez nutnosti vytvárať duplikáty materiálu.

Konfigurácia parametrov ako veľkosť binu, padding alebo počet inštancií sa vykonáva v základnej metóde `Start()`.

#### Spracovanie prijatých údajov

Spracovanie údajov prebieha v Unity takmer identicky ako v Unreal Engine, vrátane výpočtu minimálnej a maximálnej hodnoty, ich uloženia a prípravy na ďalšie spracovanie.

#### Vykresľovanie histogramu

Na rozdiel od Unreal Engine, kde sa používa komponent `InstancedMesh`, Unity využíva funkciu `Graphics.DrawMeshInstanced` na priame vykreslenie viacerých kópií jedného objektu. Okrem samotného `Mesh`u je potrebné explicitne zadať aj `Material`, pretože funkcia neobsahuje implicitnú väzbu na vizuálne vlastnosti objektu.

Pre individuálne nastavenie farieb inštancií sa využíva komponent `MaterialPropertyBlock`, ktorý umožňuje priradiť vlastnosti ako napríklad pole farieb cez `SetVectorArray`. Transformácie pre každú inštanciu sú uložené ako pole typu `Matrix4x4`, ktoré obsahuje pozíciu, rotáciu a mierku.

Po vypočítaní pozícií a mierok pre všetky biny sa volá `DrawMeshInstanced`, ktorá prijíma vstupný `Mesh`, `Material`, zoznam transformácií a `MaterialPropertyBlock` obsahujúci farby. Táto funkcia vykreslí všetky objekty naraz a je optimalizovaná pre výkon, aj keď pri veľmi veľkom počte inštancií môže byť potrebné rozdeliť výstup na viacero dávok.

Celý proces vizualizácie histogramu prebieha v metóde `RenderHistogram`. Rovnako ako v Unreal Engine, na výpočet pozícií, mierok a farieb bola použitá pomocná knižnica `HistogramRenderUtils`, ktorá bola upravená s ohľadom na súradnicový systém Unity. Vertikálne zarovnanie binov so základňou bolo zabezpečené pomocou dodatočného posunu po osi `Y`, ktorý zodpovedá posunu po osi `Z` v Unreal Engine.

#### Asynchrónny režim vykresľovania

Asynchrónny režim vykresľovania nie je implementovaný. Počas vývoja sa zistilo, že pre objektívne porovnanie výkonu medzi platformami je lepšie testovať v neoptimalizovanom režime, lebo asynchrónne vykresľovanie sa pre webovú verziu zatiaľ neplánuje. Okrem toho je viacvláknové riadenie v Unity náročnejšie a jeho implementácia by si vyžadovala podstatne viac úsilia. Asynchrónne vykresľovanie sa môže pridať neskôr, ak sa Unity ukáže ako najvýkonnejšia platforma.

#### Finalizácia a čistenie

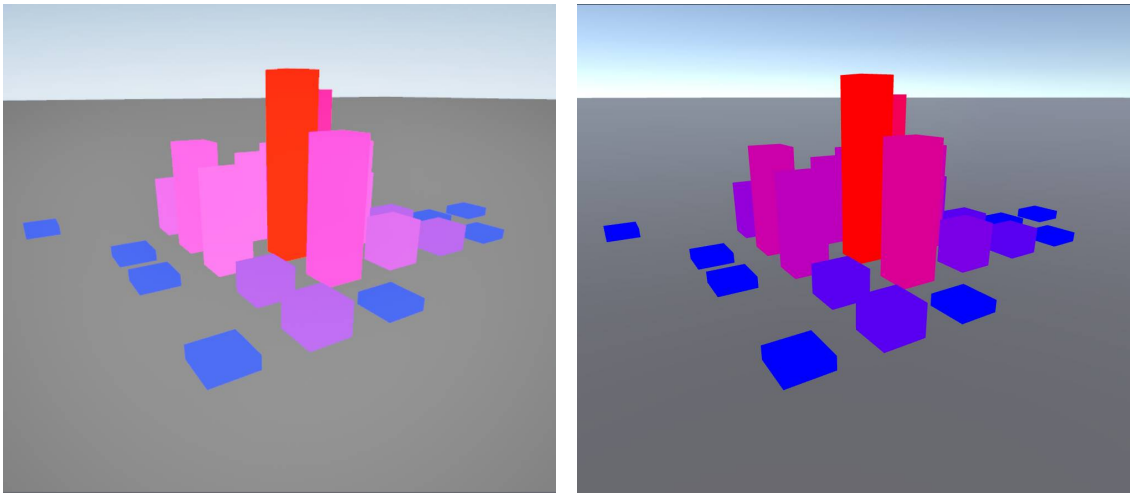
Metóda `OnDestroy()` je v tejto implementácii prepísaná tak, aby najskôr zabezpečila ukončenie a vymazanie pomocných komponentov, ako sú `Broker` alebo `HistogramDataLoader`. Až po ich odstránení sa volá základná funkcionalita prostredníctvom `base.OnDestroy()`. Tento postup zabezpečuje, že všetky závislosti sú správne uvoľnené ešte pred zničením samotného objektu a nedochádza tak k zbytočným chybám počas ukončovania scény alebo simulácie.

## 5.4 Návrh komponentu Scene a VR prostredie

Komponent „Scene“ je priestor, v ktorom sú umiestnené všetky hlavné komponenty systému a vzájomne na seba pôsobia. V oboch implementáciách (Unreal Engine a Unity) scéna obsahuje základné prvky, ako je podlaha, osvetlenie, hráč (používateľ), objekt histogramu (HistogramRenderer), ako aj ďalšie komponenty, ktoré automaticky pridáva platforma. Tieto prvky boli upravené tak, aby podporovali vizualizáciu virtuálnej reality a poskytovali základnú interakciu používateľa s údajmi.

Na obrázku 5.4 je zobrazená rovnaká scéna s identickými histogramami z rovnakého uhla pohľadu — v prostredí Unreal Engine (vľavo) a Unity (vpravo). Vďaka tomu je možné vizuálne porovnať vzhľad scén v oboch implementáciách.

Na urýchlenie vývoja a zabezpečenie kompatibility s VR boli v oboch prípadoch použité oficiálne šablóny - v Unreal Engine „Virtual Reality“ a v Unity „VR“. Vďaka nim bolo možné rýchlo integrovať hlavné prvky ovládania, priestorovej orientácie a interakcie. Niektoré aspekty však bolo potrebné vylepšiť, napríklad pohyb hráča vo všetkých smeroch vrátane vertikálnej osi. Ovládanie pohybu bolo navrhnuté pre štandardné VR ovládače s dvoma joystickmi, kde ľavý joystick ovláda pohyb dopredu, dozadu a do strán a pravý joystick slúži na vertikálny



Obrázok 5.4: Porovnanie vizualizácie histogramu v prostredí Unreal Engine (vľavo) a Unity (vpravo)

pohyb. Tieto úpravy sú podrobne opísané v nasledujúcej časti.

### 5.4.1 Úpravy pohybu hráča

Pohyb hráča vo virtuálnej realite bol prispôsobený tak, aby umožňoval voľný pohyb vo všetkých smeroch — dopredu, dozadu, do strán, ako aj vertikálne nahor a nadol. Východiskové šablóny pre VR poskytujú základné ovládanie, no pre potreby tejto práce bolo potrebné vykonať niekoľko úprav.

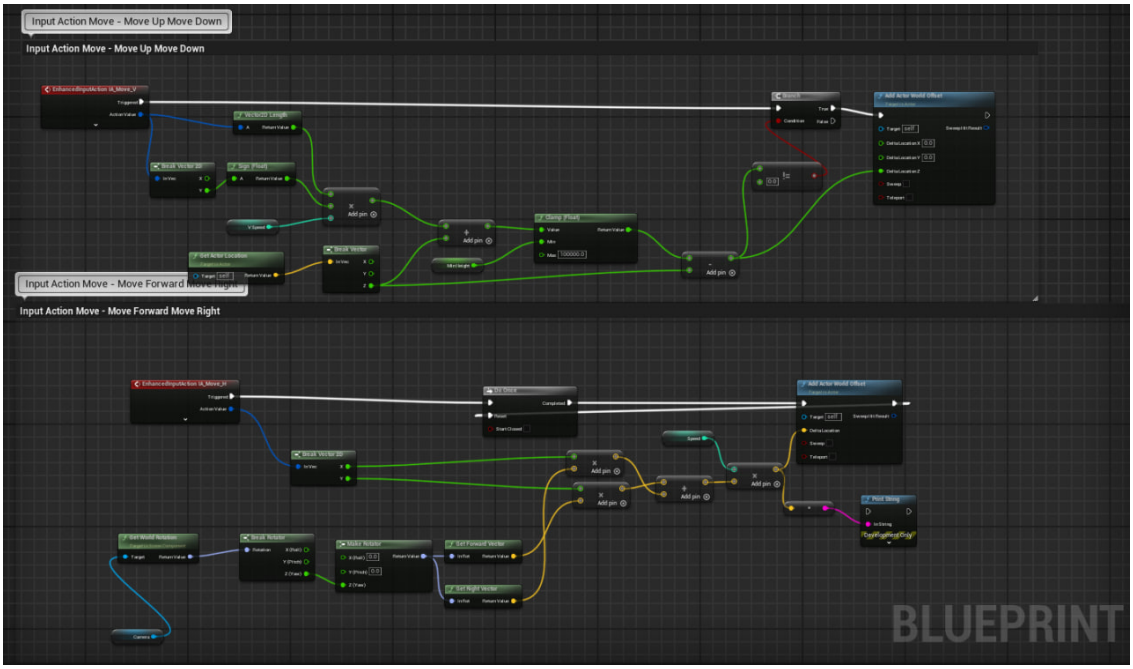
#### Implementácia zmien v Unreal Engine

V Unreal Engine bol celý pohyb a interakcia hráča definovaný v komponente `VRPawn`, ktorý predstavuje základný blueprint používateľa vo VR šablóne. Obsahuje logiku pohybu, interakcie s objektmi, ako aj vykreslenie rúk hráča v prostredí. Pôvodná implementácia pohybu pomocou ľavého joysticku bola rozšírená o vertikálnu zložku, ktorú aktivuje pravý joystick. Úprava bola vykonaná priamo v blueprint logike a je znázornená na obrázku 5.5.

#### Implementácia zmien v Unity

V Unity bol ako hráč použitý komponent `XR Origin (XR Rig)`, ktorý je súčasťou balíka `XR Interaction Toolkit`. Tento komponent poskytuje základnú podporu pre VR interakciu, no jeho štruktúra je komplexná a pozostáva z viacerých súčastí, ako sú vstupné akcie, interakcie, zariadenia, atď.

Namiesto zásahu do existujúcej logiky bolo jednoduchšie pridať vlastný skript `VerticalMovementVR.cs`, ktorý sa stará o vertikálny pohyb pozdĺž osi Y. Ten-




Obrázok 5.5: Upravený pohybový Blueprint pre VR v Unreal Engine

to skript bol pridaný priamo na objekt `Player` a reaguje na vstupy z pravého joysticku.

## 5.5 Návrh komponentu CameraMovementActor

Komponent `CameraMovementActor` bol vyvinutý ako pomocný nástroj na automatizované testovanie výkonu vizualizácie histogramov. Jeho hlavnou úlohou je simulovať pohyb kamery po vopred definovanej trajektórii a priebežne zaznamenávať výkonnostné metriky počas tohto pohybu. Tento kontrolovaný pohyb umožňuje testovať scény v rovnakých podmienkach.

Komponent podporuje manuálne definovanie trajektórie, ako aj automatické generovanie pomocou funkcie `GenerateBenchmarkTrajectory()`. V prípade testovania bola použitá druhá možnosť. Táto funkcia generuje sériu pozícií a rotácií kamery, ktoré napodobňujú typické správanie používateľa vo VR — približovanie sa k objektu, diagonálny pohyb po scéne a otáčanie okolo vlastnej osi v rôznych smeroch. Podrobný popis tejto trajektórie sa nachádza v sekcii 6.3.

Pohyb medzi bodmi je plynulý pomocou lineárnej interpolácie polohy a rotácie. Rýchlosť pohybu a otáčania je určená konfigurovateľnými parametrami. Počas celej trajektórie sa výkonnostné údaje zaznamenávajú pri každej vykreslenej snímke (frame). Tento prístup poskytuje vyššiu časovú presnosť a detailnosť dát, keďže pri vysokých FPS môže počas jednej sekundy vzniknúť aj viac ako 100

vzoriek.

Po ukončení simulácie sa údaje exportujú do súboru vo formáte CSV s nasledujúcimi stĺpcami:

```
Timestamp,FPS,RAM_MB,CPUTime,Engine,HistogramSize
```

V prostredí Unreal Engine je komponent implementovaný v triede `ACameraMovementActor`, zatiaľ čo v prostredí Unity sa používa trieda `CameraMovementActor`.

## 5.6 Kompilácia a nasadenie

V rámci bakalárskej práce boli použité Unreal Engine verzie `5.4.4` a Unity verzie `6000.0.38f1`. Na podporu VR headsetu bolo nainštalované Android Studio, konkrétne `Koala Feature Drop` vo verzii `2024.1.2`. Na nastavenie Android Studia pre Unreal Engine bol použitý oficiálny návod od Epic Games [28], ktorý bez problémov fungoval aj pre Unity - neboli potrebné žiadne dodatočné nastavenia.

Obe verzie (Unreal aj Unity) boli navrhnuté tak, aby podporovali akékoľvek zariadenie kompatibilné s OpenXR a neboli viazané na konkrétny typ headsetu. Vývoj a testovanie však prebiehali na náhlavnej súprave `Meta Quest 2`.

Export a inštalácia aplikácie z Unreal Engine prebehli bez problémov. Po stiahnutí súboru APK do zariadenia sa aplikácia okamžite spustila a fungovala podľa očakávania.

Avšak pri aplikácii Unity vznikol problém. Napriek tomu, že systém hlásil úspešnú inštaláciu a neexistenciu chýb, aplikácia sa niekedy nezobrazila v zozname dostupných aplikácií pri nahrávaní do zariadenia. Problém sa podarilo vyriešiť použitím oficiálnej aplikácie `Meta Quest Developer Hub`, ktorá umožňuje priame pripojenie náhlavnej súpravy cez USB alebo Wi-Fi, monitorovanie stavu zariadenia a inštaláciu súborov APK priamo z počítača. S pomocou tohto nástroja prebehla inštalácia bez problémov.

# 6 Metodika experimentálneho porovnania troch prostredí

Táto kapitola sa zameriava na meranie výkonu systému pri vykresľovaní veľkého množstva údajov v troch rôznych prostrediach - Unreal Engine, Unity a webová verzia. Cieľom je vytvoriť rovnaké testovacie podmienky pre všetky platformy a objektívne zhodnotiť ich možnosti z hľadiska plynulosti a škálovateľnosti.

## 6.1 Použité zariadenia

Testovanie výkonnosti bolo vykonané na dvoch typoch zariadení: osobnom prenosnom počítači a náhlavnej súprave pre virtuálnu realitu.

**Prenosný počítač**

Na vykresľovanie v desktopovom režime boli použité tieto hardvérové špecifikácie:

- **CPU:** AMD Ryzen 5 5600H s grafickými jadrami Radeon Graphics (12 vlákien, 3.3 GHz)
- **GPU:** NVIDIA GeForce RTX 3050 Laptop GPU (3965 MB VRAM)
- **RAM:** 16 GB
- **OS:** Windows 11 Pro 64-bit (build 26100)

**Náhlavná súprava**

Testovanie vo virtuálnej realite prebiehalo na zariadení **Meta Quest 2** so systémom vo verzii `76.1025`.

## 6.2 Testované veľkosti histogramov a vizualizačné nastavenia

Testovanie výkonnosti sa vykonalo na trojrozmerných histogramoch (TH3), pretože tento typ vizualizácie predstavuje najnáročnejší scenár z hľadiska počtu vykresľovaných objektov.

Každý bin bol reprezentovaný ako nepriehľadná biela kocka s rozmermi **1x1x1 meter**. Medzi jednotlivými binmi bol ponechaný rovnomerný odstup **1 meter**. Na rozdiel od štandardnej verzie aplikácie, ktorá používala priesvitné materiály bez tieňov a s dynamickým zafarbením podľa hodnoty binu, sa pri testovaní použili jednoduché nepriehľadné biele materiály so zapnutým tieňovaním — rovnako ako vo webovej verzii.

Testy boli vykonané na nasledujúcich konfiguráciách histogramov:

- 10x10x10 (1 000 binov)
- 50x20x10 (10 000 binov)
- 100x100x10 (100 000 binov)
- 100x100x50 (500 000 binov)
- 100x100x100 (1 000 000 binov)
- 150x100x100 (1 500 000 binov)
- 250x180x100 (4 500 000 binov)

Tieto hodnoty reprezentujú postupné zvyšovanie vizualizačnej náročnosti a umožňujú posúdiť škálovateľnosť jednotlivých platforiem pri vykresľovaní veľkého množstva údajov.

## 6.3 Trajektória kamery

Na zabezpečenie konzistentného testovania vo všetkých prostrediach bola použitá identická cesta kamery, ktorá sa generuje automaticky. Tým sa zabezpečí rovnaký pohyb a orientácia kamery bez ohľadu na konkrétnu platformu (Unreal Engine, Unity alebo webová verzia), čo zaručuje objektívne výsledky merania.

Pohyb kamery prebieha nasledovne:

1. **Štart:** Kamera začína 10 metrov pred histogramom, vo výške **1.7 metra**, čo približne zodpovedá výške očí používateľa.
2. **Pohľad doprava:** Bez pohybu v priestore sa kamera otočí doprava, aby získala šikmý výhľad na histogram.
3. **Návrat dopredu:** Kamera sa otočí späť na pôvodný smer.
4. **Pohľad nahor:** Kamera sa nakloní smerom nahor.
5. **Naspäť na horizont:** Kamera sa opäť narovná do horizontálnej polohy.
6. **Priblíženie k histogramu:** Kamera sa plynulo presunie priamo dopredu až na okraj histogramu.
7. **Otočenie na diagonálu:** Kamera sa otočí o **45°**, čím sa nastaví smer pohybu po diagonále histogramu.
8. **Pohyb po diagonále:** Kamera sa pohybuje v smere od začiatku histogramu po jeho diagonále. Ak je táto diagonála dlhšia ako **20 metrov**, pohyb je obmedzený na túto vzdialenosť.
9. **Pauza v strede:** V strede tejto trajektórie sa kamera zastaví a vykoná sériu otočení – najprv doľava, potom doprava, následne nahor a nakoniec nadol. Tým sa simulujú typické pohľady používateľa v priestore.
10. **Dokončenie pohybu:** Kamera sa vráti do smeru pôvodnej diagonály a pokračuje v pohybe až na koniec určeného úseku.
11. **Koniec:** Kamera sa zastaví a ukončí sa celý pohyb.

Táto trajektória bola zvolená s cieľom simulovať realistické správanie používateľa vo VR prostredí – vrátane pohybov tela a pohľadov do rôznych smerov. Automatizované generovanie zabezpečilo rovnaké podmienky pre porovnanie výkonu na všetkých testovaných platformách.

## 6.4 Zber ukazovateľov výkonnosti

Všetky ukazovatele výkonnosti boli zaznamenávané automaticky počas simulácie pohybu kamery.

Zber údajov prebiehal pri každej vykreslenej snímke a po skončení simulácie boli údaje exportované do súboru vo formáte CSV.

Pre každú vzorku boli uložené nasledujúce hodnoty: **časová značka** (**Timestamp**), **snímky za sekundu** (**FPS**), **RAM_MB**, **čas CPU** (**CPUTime**), **použitý motor** (**Engine**) a **veľkosť histogramu** (**HistogramSize**).

## 6.5 Obmedzenia a špecifiká merania

Počas experimentálneho merania sa vyskytli viaceré technické obmedzenia, ako aj vlastnosti samotného spôsobu zberu údajov, ktoré môžu ovplyvniť interpretáciu niektorých ukazovateľov:

### 6.5.1 Obmedzenia merania

- **RAM v Unity:** API poskytované prostredím Unity neumožňuje priamy prístup k celkovej spotrebe operačnej pamäte. Zaznamenávané boli len údaje o alokovanej pamäti v rámci .NET heap-u, čo predstavuje len čiastočný pohľad na reálne využitie RAM.
- **RAM vo webovej verzii:** Webové prehliadače z bezpečnostných dôvodov neposkytujú prístup k systémovej pamäti. Preto nebolo možné získať žiadne relevantné údaje o využití RAM v tomto prostredí.
- **Obmedzenie FPS v prehliadači:** Prehliadače automaticky synchronizujú vykresľovanie s frekvenciou displeja. To znamená, že webová verzia môže byť obmedzená.

### 6.5.2 Špecifiká merania

- **Zaznamenávanie údajov pri každej snímke:** Údaje boli zaznamenávané pri každom vykreslení snímky, v dôsledku čoho sa celkový počet záznamov medzi platformami líši. Rozdiely vznikajú v závislosti od dosiahnutej snímkovej frekvencie (FPS) – pri vyššom výkone sa generuje viac hodnôt, pri nižšom menej. Takto získané údaje umožňujú presnejšie vyhodnotenie vývoja výkonu v čase.
- **Zhodné časové značky:** Pri veľmi vysokom FPS sa môže stať, že viacero záznamov má rovnakú hodnotu časovej značky (Timestamp), keďže rozdiel medzi snímkami je menší ako 1 ms. To je očakávané správanie a neovplyvňuje presnosť dát.

- **Variabilita trvania simulácie:** Hoci trajektória kamery a rýchlosť pohybu boli vo všetkých implementáciách rovnaké, skutočný čas vykonávania sa mohol líšiť. Platformy s nižším výkonom (napr. pri veľkých histogramoch) vykresľovali scény pomalšie, čím sa čas simulácie predĺžil o niekoľko sekúnd. Tento rozdiel je prirodzeným dôsledkom spôsobu vykonávania pohybu v závislosti od počtu snímok.

# 7 Analýza výsledkov experimentálneho porovnania

Táto kapitola je venovaná analýze výkonu troch implementácií systému (NDMVR, Unreal Engine a Unity) pri vizualizácii histogramov rôznych veľkostí.

Testovanie bolo realizované na dvoch typoch zariadení – prenosnom počítači a náhlavnej súprave pre virtuálnu realitu (Meta Quest 2), pričom boli zachované identické vizualizačné podmienky vo všetkých prostrediach.

Analýza sa sústreďuje predovšetkým na ukazovateľ snímkovej frekvencie (FPS), ktorý poskytuje informácie o plynulosti vykresľovania.

Hoci boli počas merania zaznamenávané aj údaje o využití operačnej pamäte (RAM), tieto neboli do analýzy zahrnuté z dôvodu neúplných alebo nedostupných dát v niektorých implementáciách. Podrobnosti sú uvedené v sekcii 6.5.1.

Pre každú testovaciu konfiguráciu sú uvedené grafy priebehu FPS a tabuľka so základnými štatistickými ukazovateľmi, spolu s komentárom k pozorovaným výsledkom.

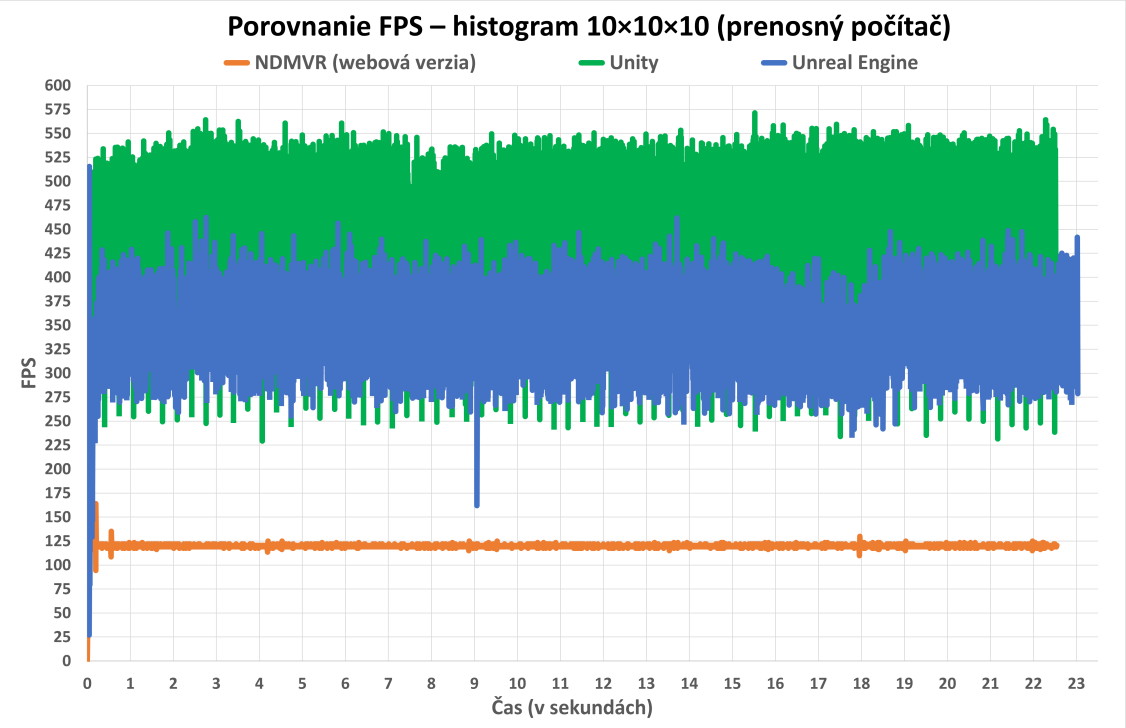
## 7.1 Výsledky testovania na prenosnom počítači

Táto časť obsahuje analýzu výsledkov testov na prenosnom počítači pre každú konfiguráciu histogramu.

### 7.1.1 Histogram 10×10×10 (1 000 binov)

Na obrázku 7.1 a v tabuľke 7.1 sú znázornené výsledky testovania výkonnosti pri vizualizácii histogramu veľkosti 10×10×10 (1 000 binov) na prenosnom počítači.

Platforma Unity dosiahla najvyšší priemerný počet snímok za sekundu, približne 425 FPS, no zároveň aj najväčšiu variabilitu (štandardná odchýlka približne 64 FPS, koeficient variability 15 %). Unreal Engine zaznamenal nižší priemer, približne 348 FPS, ale stabilnejšie hodnoty (koeficient variability 10,8%). Webo-




Obrázok 7.1: Graf FPS v závislosti od času s histogramom 10×10×10 na prenosnom počítači

Tabuľka 7.1: Základné štatistiky FPS – histogram 10×10×10 na prenosnom počítači

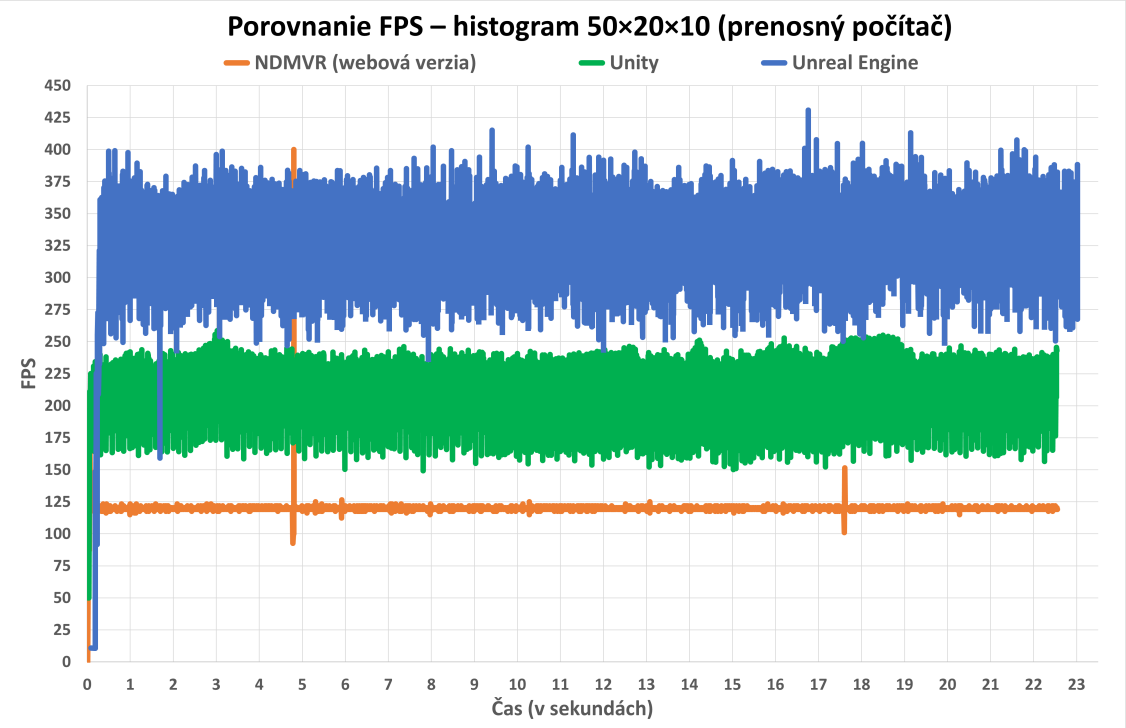
| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 424.75 | 347.90 | 120.08 |
| Medián | 397.31 | 347.74 | 120.48 |
| Minimum | 50.00 | 27.11 | 73.53 |
| Maximum | 571.53 | 515.57 | 312.50 |
| Štand. odchýlka | 63.90 | 37.54 | 4.27 |
| **Koef. variability (%)** | 15.0 | 10.8 | 3.6 |

vá verzia dosiahla konštantný výkon okolo 120 FPS s veľmi nízkou odchýlkou, čo zodpovedá typickému správaniu prehliadača s obmedzením FPS.

### 7.1.2 Histogram 50×20×10 (10 000 binov)

Na obrázku 7.2 a v tabuľke 7.2 sú zobrazené výsledky vizualizácie histogramu 50×20×10 (10 000 binov) na prenosnom počítači.

V porovnaní s predchádzajúcim testom došlo k výraznému poklesu výkonu platformy Unity (219 FPS oproti 425 FPS), no zároveň sa znížila aj jej variabilita.




Obrázok 7.2: Graf FPS v závislosti od času s histogramom 50×20×10 na prenosnom počítači

Tabuľka 7.2: Základné štatistiky FPS – histogram 50×20×10 na prenosnom počítači

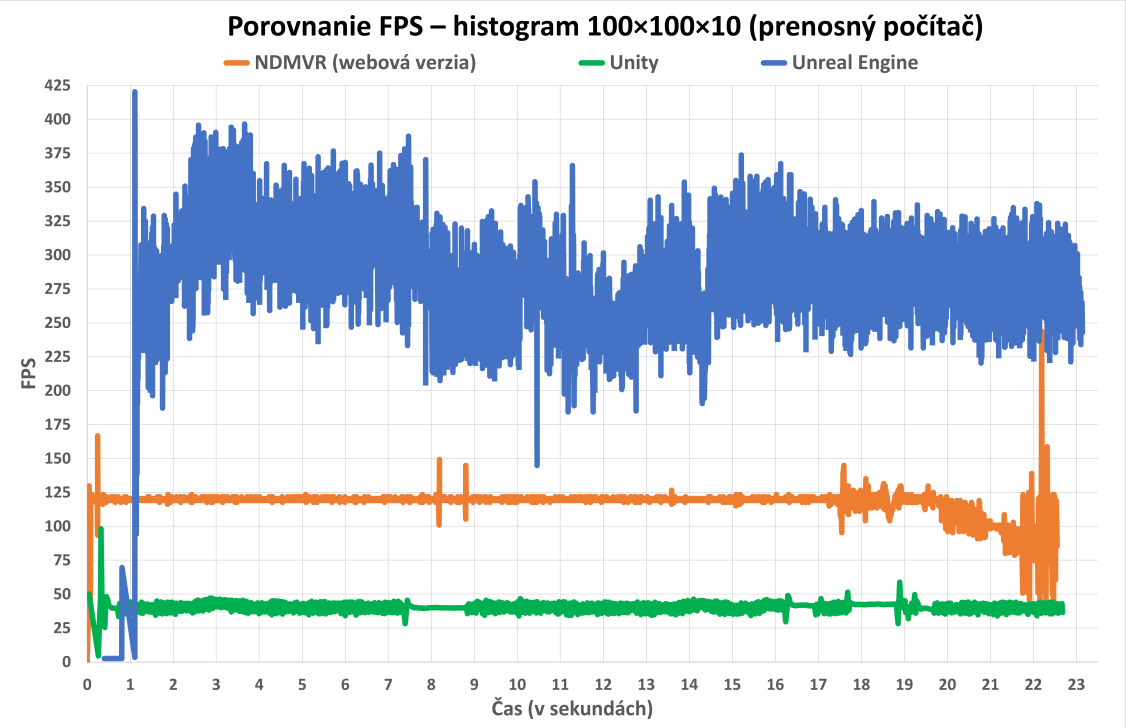
| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 218.50 | 330.35 | 120.12 |
| Medián | 220.26 | 332.51 | 120.48 |
| Minimum | 50.00 | 10.66 | 59.52 |
| Maximum | 258.66 | 430.79 | 400.00 |
| Štand. odchýlka | 19.46 | 28.96 | 6.17 |
| **Koef. variability (%)** | 8.9 | 8.8 | 5.1 |

Výsledky Unreal Engine zostali na rovnakej úrovni. Webová verzia opäť vykazuje konštantný výkon.

### 7.1.3 Histogram 100×100×10 (100 000 binov)

Na obrázku 7.3 a v tabuľke 7.3 sú znázornené výsledky testovania pri vizualizácii histogramu veľkosti 100×100×10 na prenosnom počítači.

Oproti predchádzajúcim konfiguráciám možno pozorovať výrazný pokles výkonu platformy Unity — priemerný FPS klesol na hodnotu približne 40.



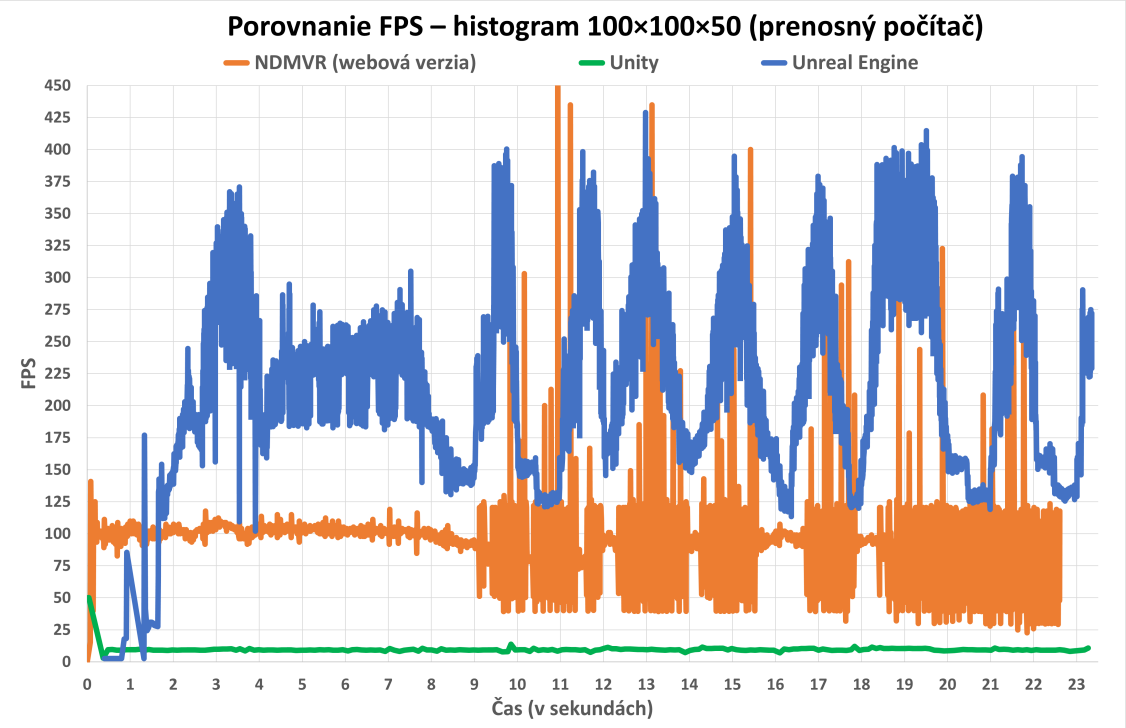

Obrázok 7.3: Graf FPS v závislosti od času s histogramom 100×100×10 na prenosnom počítači

Tabuľka 7.3: Základné štatistiky FPS – histogram 100×100×10 na prenosnom počítači

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 40.43 | 289.26 | 118.14 |
| Medián | 41.22 | 290.51 | 120.48 |
| Minimum | 4.53 | 2.50 | 39.53 |
| Maximum | 98.09 | 420.26 | 243.90 |
| Štand. odchýlka | 4.23 | 34.22 | 8.62 |
| **Koef. variability** (%) | 10.47 | 11.83 | 7.30 |

Unreal Engine si udržal vysoký výkon okolo 289 FPS, no stabilita sa zhoršila — viditeľné sú výkyvy aj poklesy, najmä na začiatku simulácie.

Webová verzia opäť vykazovala konzistentné hodnoty približne 118 FPS, no v poslednej fáze testu došlo k náhlemu nárastu variability, pravdepodobne spôsobenému preťažením v prehliadači.




Obrázok 7.4: Graf FPS v závislosti od času s histogramom 100×100×50 na prenosnom počítači

Tabuľka 7.4: Základné štatistiky FPS – histogram 100×100×50 na prenosnom počítači

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 9.83 | 236.72 | 96.46 |
| Medián | 9.35 | 231.46 | 100.00 |
| Minimum | 3.00 | 2.50 | 15.24 |
| Maximum | 50.00 | 428.85 | 500.00 |
| Štand. odchýlka | 3.95 | 67.69 | 33.42 |
| **Koef. variability (%)** | 40.21 | 28.59 | 34.64 |

### 7.1.4 Histogram 100×100×50 (500 000 binov)

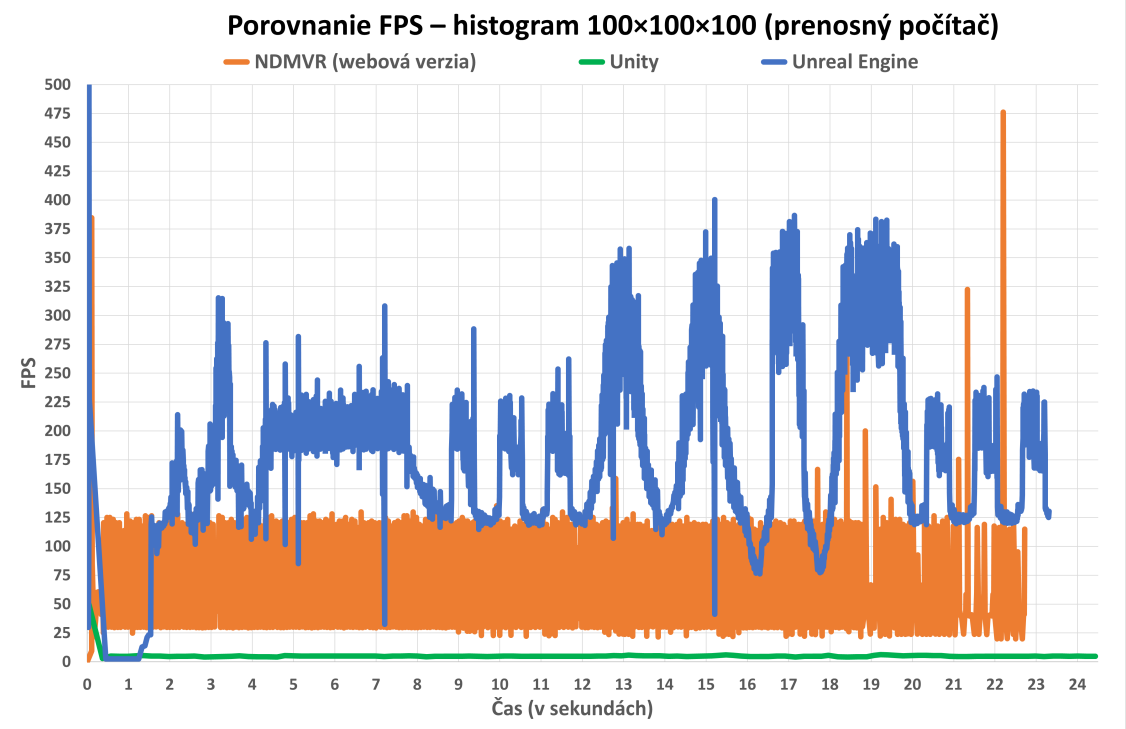
Na obrázku 7.4 a v tabuľke 7.4 sú zobrazené výsledky pre konfiguráciu 100×100×50 (500 000 binov).

Unreal Engine si napriek výraznejšiemu kolísaniu výkonu udržal prijateľný priemerný FPS približne 237, aj keď s vyššou variabilitou približne 28,6 %.

Unity opäť vykázal výrazný pokles výkonu s veľmi nízkym priemerom približne 10 FPS.

Webová verzia si dlhodobo udržiavala stabilný výkon okolo 96 FPS, ale v druhej polovici simulácie sa objavili značné výkyvy — pravdepodobne v dôsledku zaťaženia pamäte alebo spracovania v prehliadači.

### 7.1.5 Histogram 100×100×100 (1 000 000 binov)




Obrázok 7.5: Graf FPS v závislosti od času s histogramom 100×100×100 na prenosnom počítači

Tabuľka 7.5: Základné štatistiky FPS – histogram 100×100×100 na prenosnom počítači

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 5.66 | 206.20 | 68.69 |
| Medián | 4.89 | 191.27 | 51.28 |
| Minimum | 3.00 | 2.50 | 9.45 |
| Maximum | 50.00 | 578.17 | 476.19 |
| Štand. odchýlka | 5.76 | 69.80 | 41.50 |
| **Koef. variability (%)** | 101.77 | 33.85 | 60.42 |

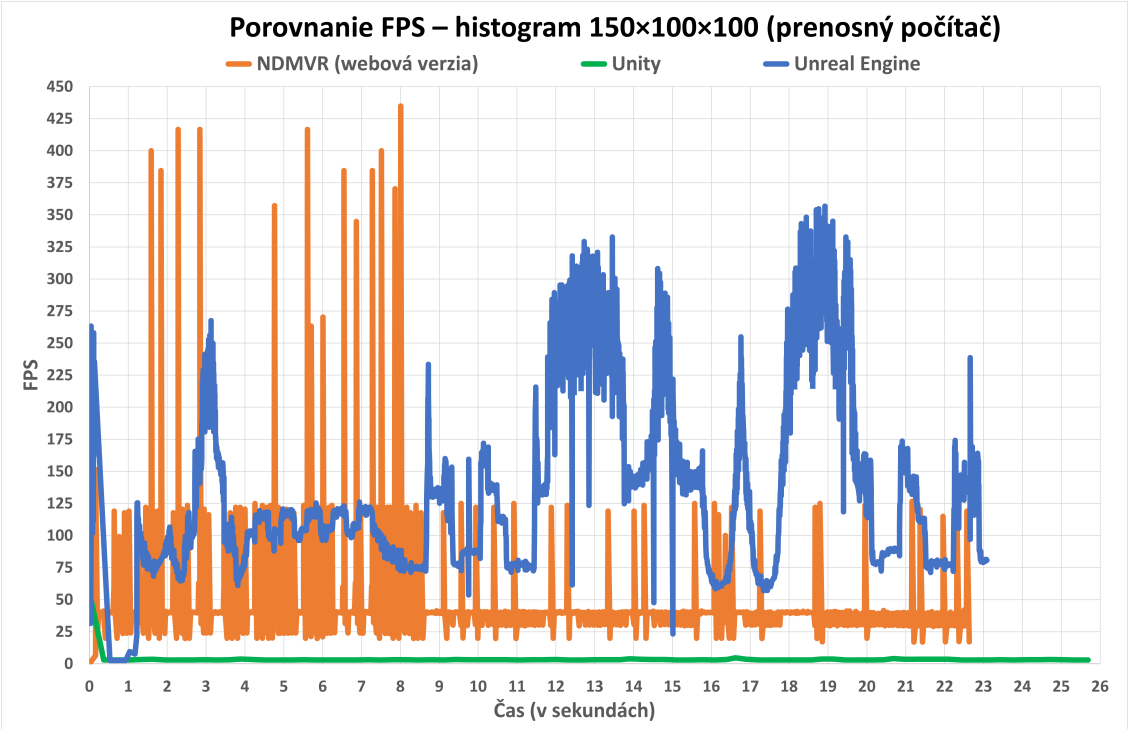
Na obrázku 7.5 a v tabuľke 7.5 sú zobrazené výsledky pre histogram 100×100×100 (1 000 000 binov).

Platforma Unity dosahovala veľmi nízku snímkovú frekvenciu približne 5,7 FPS a extrémne vysoký koeficient variability okolo 102 %.

Unreal Engine vykresľoval scénu s vysokým priemerom približne 206 FPS, ale výskyt veľkých výkyvov sa zvýšil, najmä v druhej polovici simulácie.

Vo webovej verzii došlo k výraznému poklesu stability — FPS kolísal počas celej simulácie, čo naznačuje preťaženie renderovacieho procesu pri takomto počte objektov.

### 7.1.6 Histogram 150x100x100 (1 500 000 binov)




Obrázok 7.6: Graf FPS v závislosti od času s histogramom 150×100×100 na prenosnom počítači

Na obrázku 7.6 a v tabuľke 7.6 sú zobrazené výsledky testovania pri vizualizácii histogramu 150×100×100.

Platforma Unity opäť vykazovala extrémne nízky výkon približne 4,3 FPS a veľmi vysokú variabilitu 167 %.

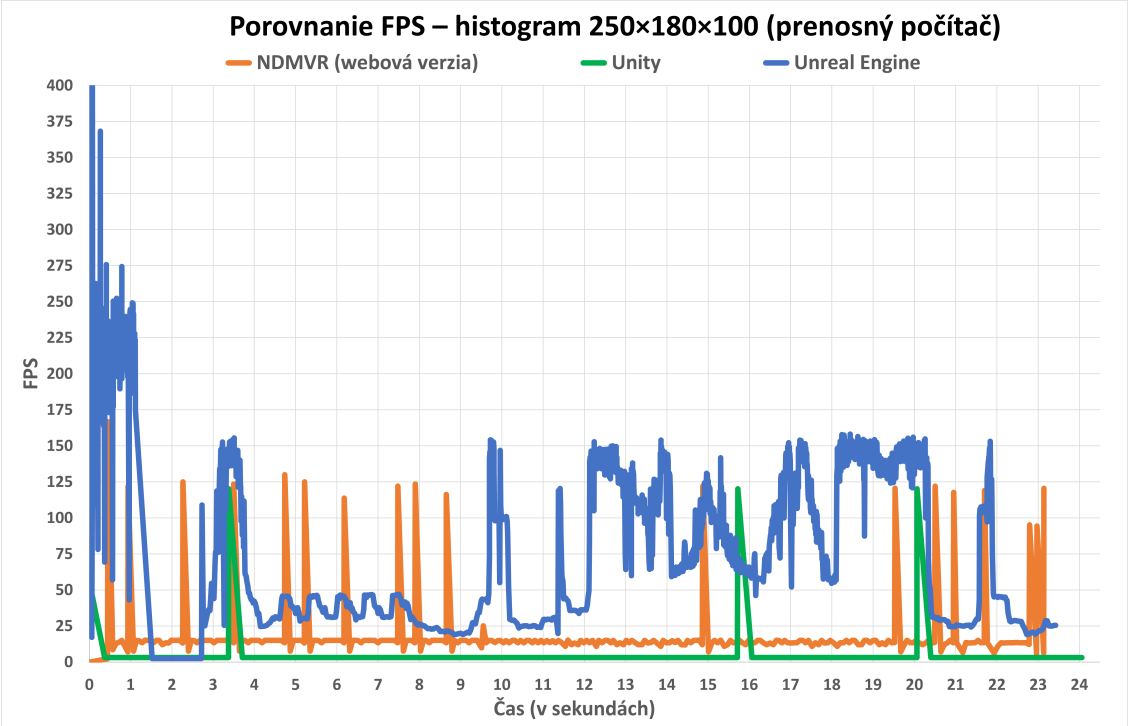
Unreal Engine si zachoval vyššiu priemernú hodnotu FPS 170 FPS, ale dochádza k pravidelným výkyvom, ktoré sú pravdepodobne spôsobené zložitými úsekmi trajektórie.

Webová verzia dosiahla nižší priemerný výkon okolo 49 FPS a tiež vykazovala zvýšenú nestabilitu, najmä v druhej polovici simulácie.

Tabuľka 7.6: Základné štatistiky FPS – histogram 150×100×100 na prenosnom počítači

| Ukazovateľ | Unity | Unreal Engine | Webová verzia |
|---|---|---|---|
| Priemer | 4.31 | 169.96 | 48.99 |
| Medián | 3.07 | 146.31 | 39.84 |
| Minimum | 3.00 | 2.50 | 6.52 |
| Maximum | 50.00 | 356.80 | 434.78 |
| Štand. odchýlka | 7.19 | 76.63 | 48.71 |
| **Koef. variability (%)** | 166.83 | 45.09 | 99.43 |

### 7.1.7 Histogram 250×180×100 (4 500 000 binov)




Obrázok 7.7: Graf FPS v závislosti od času s histogramom 250×180×100 na prenosnom počítači

Na obrázku 7.7 a v tabuľke 7.7 sú zobrazené výsledky pri najnáročnejšej konfigurácii (4,5 milióna binov).

Unreal Engine dosiahol najvyšší priemerný FPS približne 114 FPS, no pri výrazných výkyvoch.

Webová verzia dosiahla len približne 20 FPS, a to s veľmi vysokou variabilitou okolo 130 %.

Tabuľka 7.7: Základné štatistiky FPS – histogram 250×180×100 na prenosnom počítači

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 8.78 | 114.24 | 20.15 |
| Medián | 3.00 | 125.04 | 14.90 |
| Minimum | 3.00 | 2.50 | 2.31 |
| Maximum | 120.12 | 503.93 | 166.67 |
| Štand. odchýlka | 23.63 | 58.45 | 26.28 |
| **Koef. variability** (%) | 269.01 | 51.17 | 130.43 |

Unity vykazuje extrémne nízky a nestabilný výkon.

## 7.2 Výsledky testovania na náhlavnej súprave Meta Quest 2

Táto časť obsahuje analýzu výsledkov testov na náhlavnej súprave Meta Quest 2 pre každú konfiguráciu histogramu.

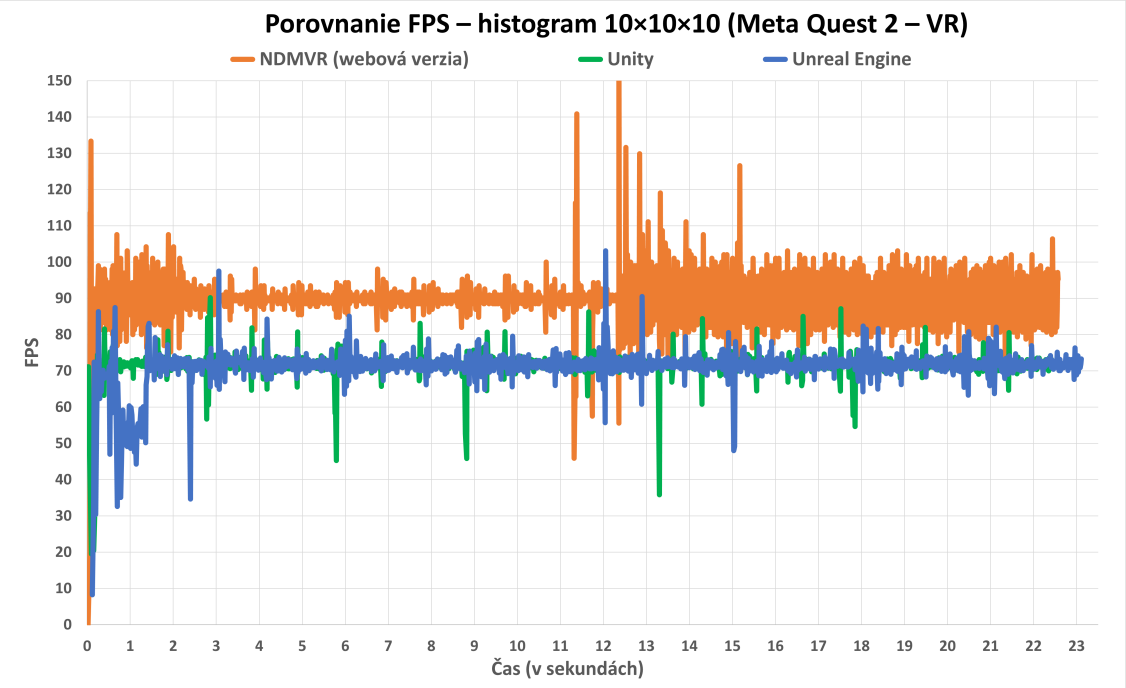
### 7.2.1 Histogram 10×10×10 (1 000 binov)

Tabuľka 7.8: Základné štatistiky FPS – histogram 10×10×10 na Meta Quest 2

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 71.59 | 71.28 | 90.04 |
| Medián | 71.67 | 71.69 | 90.09 |
| Minimum | 19.49 | 8.24 | 20.12 |
| Maximum | 90.14 | 103.11 | 161.29 |
| Štand. odchýlka | 3.29 | 4.60 | 6.08 |
| **Koef. variability** (%) | 4.59 | 6.45 | 6.75 |

Na obrázku 7.8 a v tabuľke 7.8 sú zobrazené výsledky testovania výkonu pri najjednoduchšej konfigurácii histogramu.

Všetky implementácie vykazovali stabilný výkon bez výrazných výkyvov. Webová verzia dosiahla najvyšší priemer 90 FPS , no v druhej polovici testu možno




Obrázok 7.8: Graf FPS v závislosti od času s histogramom 10×10×10 na Meta Quest 2

pozorovať mierne kolísanie. Unity aj Unreal Engine sa správali veľmi podobne a udržiavali plynulé zobrazenie okolo 71 FPS.

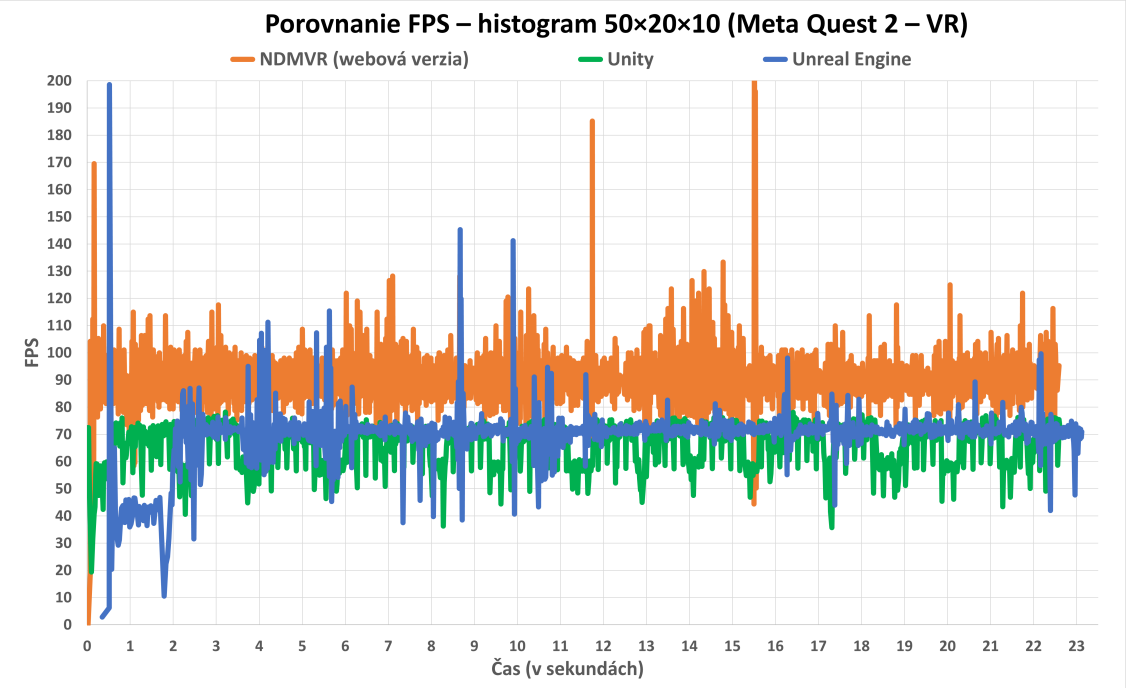
### 7.2.2 Histogram 50×20×10 (10 000 binov)

Tabuľka 7.9: Základné štatistiky FPS – histogram 50×20×10 na Meta Quest 2

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 66.88 | 70.34 | 90.47 |
| Medián | 69.52 | 71.41 | 90.09 |
| Minimum | 19.41 | 2.83 | 19.16 |
| Maximum | 78.17 | 198.60 | 243.90 |
| Štand. odchýlka | 7.28 | 9.54 | 9.96 |
| **Koef. variability (%)** | 10.89 | 13.57 | 11.01 |

Na obrázku 7.9 a v tabuľke 7.9 sú zobrazené výsledky pri konfigurácii histogramu 50×20×10.

Všetky platformy si zachovali podobné priemerné hodnoty FPS ako v predchádzajúcom teste. Zaznamenané bolo mierne zvýšenie koeficientu variability,




Obrázok 7.9: Graf FPS v závislosti od času s histogramom 50×20×10 na Meta Quest 2

čo znamená, že snímková frekvencia bola o niečo menej stabilná, ale stále v prijateľnom rozsahu.

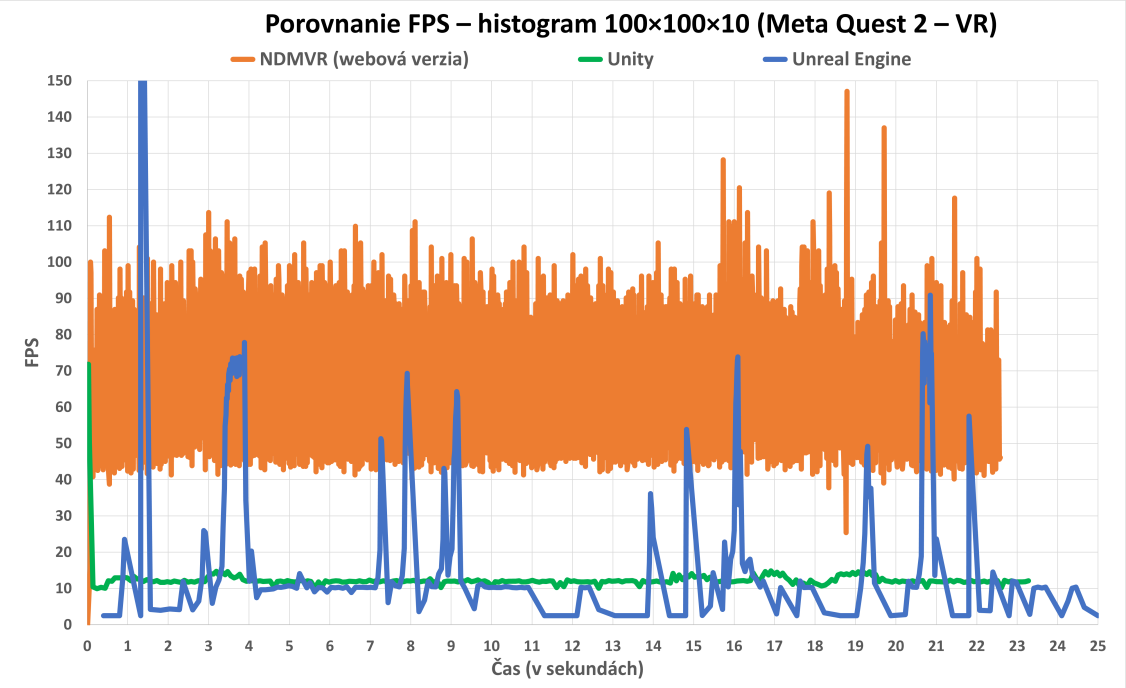
### 7.2.3 Histogram 100×100×10 (100 000 binov)

Tabuľka 7.10: Základné štatistiky FPS – histogram 100×100×10 na Meta Quest 2

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 12.75 | 29.83 | 70.22 |
| Medián | 12.02 | 13.22 | 79.37 |
| Minimum | 9.92 | 2.50 | 15.36 |
| Maximum | 71.82 | 243.24 | 147.06 |
| Štand. odchýlka | 5.52 | 29.73 | 21.91 |
| **Koef. variability** (%) | 43.27 | 99.65 | 31.21 |

Na obrázku 7.10 a v tabuľke 7.10 sú zobrazené výsledky pre histogram s veľkosťou 100×100×10.

Všetky platformy zaznamenali pokles výkonu. Priemerný FPS v Unity aj Unreal Engine výrazne klesol a zároveň sa prudko zvýšil koeficient variability, čo




Obrázok 7.10: Graf FPS v závislosti od času s histogramom 100×100×10 na Meta Quest 2

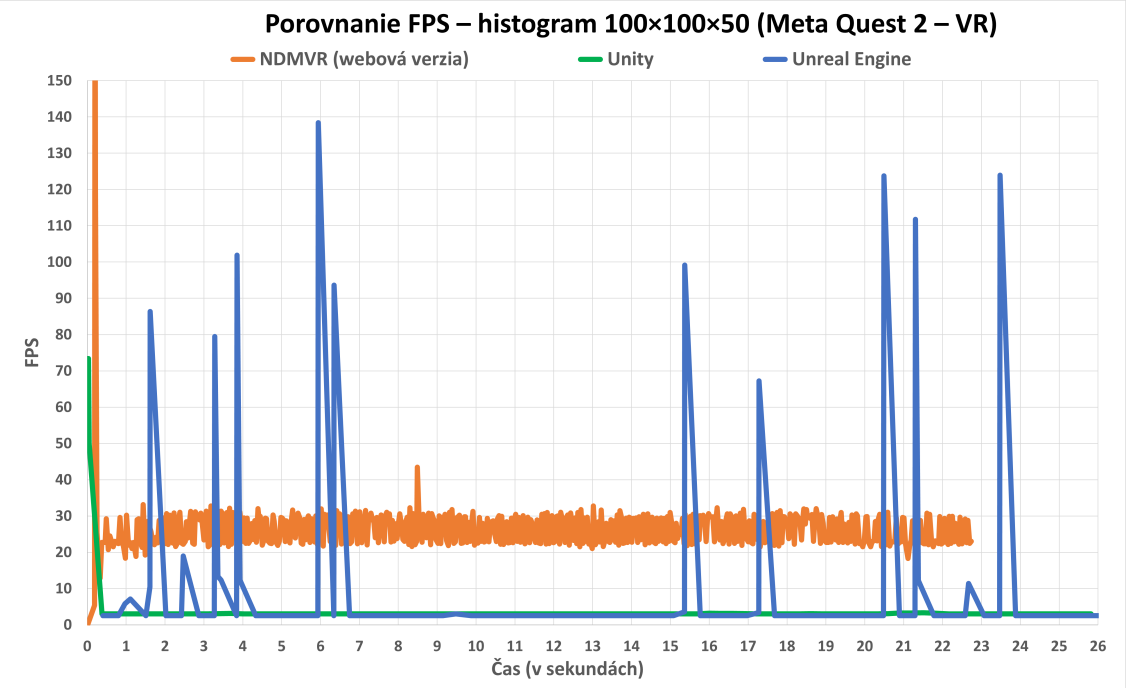
znamená zhoršenú stabilitu. Hoci Unreal Engine dosiahol vyšší priemer než Unity, jeho výkyvy boli oveľa väčšie, čo naznačuje, že koeficient variability dosiahol takmer 100%, čo poukazuje na výraznú nestabilitu výkonu. Webová verzia si zachovala stabilný výkon, aj keď bol zaznamenaný mierny pokles a zvýšená variabilita FPS.

### 7.2.4 Histogram 100×100×50 (500 000 binov)

Tabuľka 7.11: Základné štatistiky FPS – histogram 100×100×50 na Meta Quest 2

| **Ukazovateľ** | **Unity** | **Unreal Engine** | **Webová verzia** |
|---|---|---|---|
| Priemer | 5.35 | 15.57 | 26.85 |
| Medián | 3.00 | 2.50 | 28.17 |
| Minimum | 3.00 | 2.50 | 5.40 |
| Maximum | 73.40 | 138.39 | 208.33 |
| Štand. odchýlka | 12.01 | 33.15 | 9.25 |
| **Koef. variability** (%) | 224.62 | 212.93 | 34.47 |

Na obrázku 7.11 a v tabuľke 7.11 sú zobrazené výsledky pre konfiguráciu




Obrázok 7.11: Graf FPS v závislosti od času s histogramom 100×100×50 na Meta Quest 2

100×100×50.

Unity aj Unreal Engine vykazujú extrémne nestabilný výkon – koeficient variability presahuje 200%. Na rozdiel od Unity generuje Unreal Engine občas výrazné skoky vo FPS, čo spôsobuje zvýšenie priemernej hodnoty napriek celkovej nestabilite výkonu. V dôsledku výrazného zaťaženia sa celková dĺžka simulácie v prípade Unity a Unreal Engine predĺžila približne o dve sekundy oproti webovej verzii.

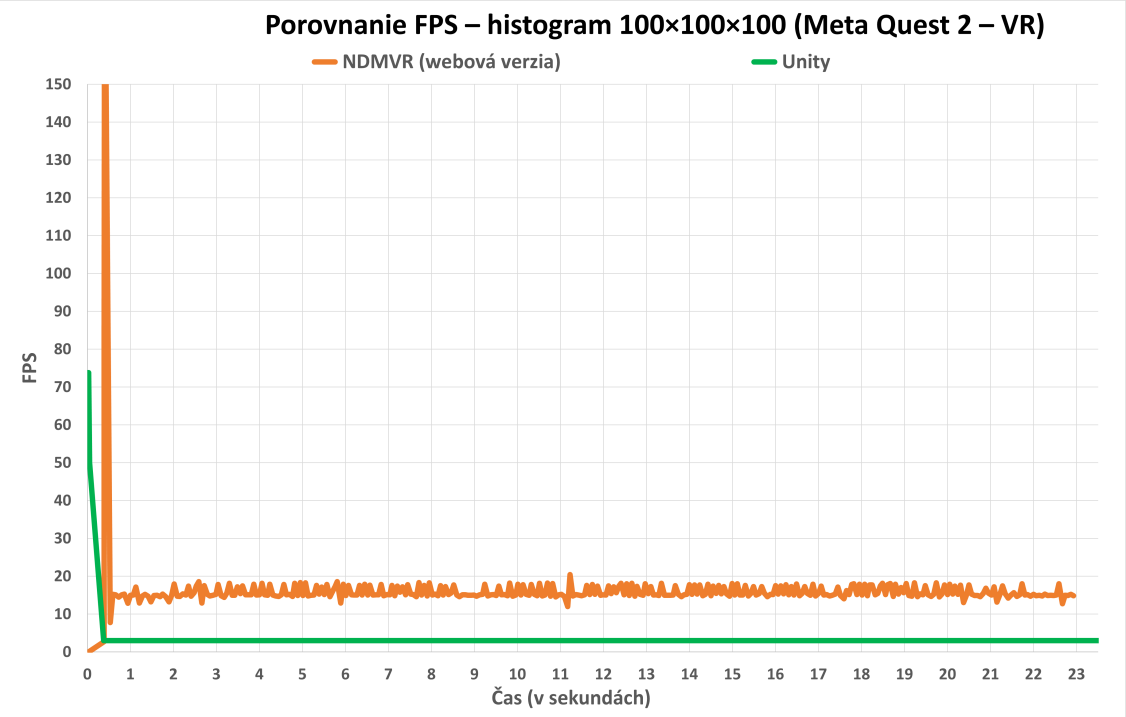
Webová verzia vykazuje nižší výkon než v predchádzajúcich prípadoch, no zostáva relatívne stabilná s FPS pohybujúcim sa v rozmedzí 20 až 30 výrazným skokom FPS na začiatku.

### 7.2.5 Histogram 100×100×100 (1 000 000 binov)

Pri tejto konfigurácii už nebolo možné spustiť testovanie v Unreal Engine. Pokus o spustenie aplikácie skončil zamrznutím ešte pred načítaním scény.

Unity sa podarilo spustiť, no výkon bol extrémne nízky. Po krátkom úvodnom skoku sa FPS ustálil na minimálnej hranici okolo 3 snímok za sekundu.

Vo webovej verzii po krátkom počiatočnom skoku FPS výkon opäť klesol. Priemerná snímková frekvencia dosiahla 16.48 FPS s variačným koeficientom približne 64%.




Obrázok 7.12: Graf FPS v závislosti od času s histogramom 100×100×100 na Meta Quest 2

Tabuľka 7.12: Základné štatistiky FPS – histogram 100×100×100 na Meta Quest 2

| **Ukazovateľ** | **Unity** | **Webová verzia** |
|---|---|---|
| Priemer | 5.48 | 16.48 |
| Medián | 3.00 | 15.17 |
| Minimum | 3.00 | 2.75 |
| Maximum | 73.77 | 200.00 |
| Štand. odchýlka | 12.44 | 10.55 |
| **Koef. variability** (%) | 226.94 | 64.01 |

### 7.2.6 Histogram 150×100×100 (1 500 000 binov)

Pri testovaní s histogramom 150×100×100 sa už aplikácia Unity nepodarilo spustiť, rovnako ako v predchádzajúcom prípade s Unreal Engine.

Webová verzia vykazovala podobný priebeh ako pri histogramoch s rozmermi 100×100×100: po krátkom počiatočnom skoku FPS nasledovalo výrazné zníženie snímkovej frekvencie. Priemerná hodnota predstavovala 11.87 FPS.

### 7.2.7 Histogram 250×180×100 (4 500 000 binov)

Rovnako ako v predchádzajúcom prípade bola spustená len webová verzia. Jej priebeh opäť kopíroval známy vzor – po úvodnom náraste nasledoval trvalý pokles snímkovej frekvencie. Priemerná hodnota FPS ďalej klesla na 7.04.

## 7.3 Zhrnutie výsledkov analýzy

Na prenosnom počítači dosiahla najvyšší výkon implementácia v prostredí **Unreal Engine**. Tento výkon je pravdepodobne podporený aj využitím optimalizačných nástrojov ako `HierarchicalInstancedStaticMeshComponent`. Stabilita výstupu však nebola vždy rovnomerná — s narastajúcim počtom objektov sa vyskytovali značné výkyvy. Napriek tomu boli hodnoty FPS dostatočne vysoké na to, aby neznižovali subjektívny pocit plynulosti až do približne jedného milióna binov.

Webová verzia (NDMVR) obsadila v desktopovom testovaní druhé miesto. Výkon začal výraznejšie klesať až pri najväčších záťažiach, no aj napriek tomu bola schopná úspešne dokončiť všetky simulácie bez zlyhania. Jej správanie bolo konzistentné a dobre škálovateľné v kontexte desktopového prostredia.

Unity sa umiestnila ako najmenej výkonná platforma v desktopovom prostredí. Výkon prudko klesal už pri stredných záťažiach. Najpravdepodobnejšou príčinou sú limity instančného vykresľovania a menej optimalizované spracovanie veľkého množstva objektov.

V prostredí virtuálnej reality (Meta Quest 2) bola **webová verzia** jedinou platformou, ktorá úspešne zvládla všetky testovacie konfigurácie vrátane najnáročnejších. Jej výkon síce postupne klesal, no systém ostal funkčný a predvídateľný.

Unity a Unreal Engine sa na VR zariadení správali podobne – zvládli len menej náročné vizualizácie (do 100 000 binov). Pri vyššej záťaži došlo k výraznému poklesu FPS alebo úplnému zlyhaniu spustenia aplikácie. Je zrejmé, že tieto platformy vyžadujú viac systémových prostriedkov, ktoré už mobilné zariadenie nevedelo zabezpečiť.

Tieto zistenia poukazujú na rozdiely v správaní platforiem pri rôznych typoch zariadení a záťaží. Unreal Engine poskytuje vysoký výkon pri dostatočných zdrojoch, zatiaľ čo webová verzia sa ukazuje ako najspoľahlivejšie riešenie v obmedzených podmienkach.

# 8 Záver

Táto bakalárska práca bola zameraná na návrh, implementáciu a experimentálne porovnanie alternatívnych platforiem pre vizualizáciu experimentálnych údajov vo virtuálnej realite (Unreal Engine a Unity) s existujúcou webovou verziou systému NDMVR. Hlavným cieľom bolo vytvoriť riešenia schopné zvládnuť vizualizáciu veľkého množstva údajov a následne objektívne vyhodnotiť ich výkonnosť v rôznych prostrediach.

V rámci implementácie boli vytvorené dva samostatné prototypy v Unreal Engine a Unity, pričom obe riešenia podporovali automatické načítavanie údajov zo servera, instančné vykresľovanie objektov, ako aj nástroj na zber výkonnostných metrik (napr. FPS).

Testovanie prebehlo na prenosnom počítači a autonómnom VR zariadení Meta Quest 2, pričom boli vizualizované histogramy od veľkosti 1 000 po 4 500 000 binov. Získané výsledky potvrdili výrazné rozdiely vo výkone jednotlivých platforiem.

Na prenosnom počítači dosiahol najvyšší výkon Unreal Engine. Hoci boli výsledky vo všeobecnosti veľmi dobré, pri vyšších záťažiach sa objavovali výraznejšie výkyvy. Za vysoký výkon pravdepodobne zodpovedajú použité optimalizačné techniky, ako napríklad `HierarchicalInstancedStaticMeshComponent`.

Webová verzia (NDMVR) preukázala konzistentný výkon. Až do veľkosti jedného milióna binov si udržiavala relatívne stabilné hodnoty FPS a zvládla dokončiť všetky testy vrátane najnáročnejších konfigurácií, čím sa ukázala ako spoľahlivé riešenie, najmä v obmedzených podmienkach.

Unity vykazovalo slabší výkon už pri stredných záťažiach. Systém dosahoval nízku snímkovú frekvenciu, pričom hlavným obmedzením bola pravdepodobne menej efektívna správa veľkého množstva objektov a obmedzené možnosti optimalizácie v porovnaní s Unreal Engine.

Testovanie na náhlavnej súprave Meta Quest 2 odhalilo ešte väčšie rozdiely. Ako jediná platforma bola schopná zvládnuť všetky testy len webová verzia NDMVR. Výkon síce postupne klesal, ale systém ostával funkčný a predvídateľ-

ný. Naopak, aplikácie v Unreal Engine a Unity zlyhávali pri väčšej záťaži – buď zamŕzali pri štarte, alebo dosahovali extrémne nízku a nestabilnú snímkovú frekvenciu. To naznačuje, že tieto platformy sú vo VR prostredí výrazne náročnejšie na systémové zdroje.

Na základe získaných výsledkov nie je možné jednoznačne určiť univerzálne najlepšiu platformu pre všetkých používateľov. Výber vhodného riešenia závisí najmä od dostupného hardvéru a požiadaviek na rozsah vizualizácie.

V prípade nedostatku výpočtových prostriedkov alebo použitia mobilných zariadení, ako je náhlavná súprava pre VR, sa ako najspoľahlivejšia ukázala webová verzia NDMVR. Vďaka svojej multiplatformovej povahe, nízkym nárokom na výkon a stabilnému správaniu je vhodnou voľbou pre široké spektrum používateľov.

Unreal Engine síce vyžaduje výkonnejšie zariadenia, no ponúka rozsiahle možnosti optimalizácie a pokročilého renderovania. V budúcnosti sa môže stať ešte vhodnejšou voľbou, najmä pri využití hybridného prístupu – kde by samotné vykresľovanie prebiehalo na výkonnom počítači a výsledok by sa zobrazoval vo VR zariadení.

Unity sa v tejto práci neosvedčilo ako vhodná platforma pre veľké vizualizačné záťaže bez dodatočných optimalizačných techník. Na efektívne využitie Unity pri väčších záťažiach by bolo potrebné implementovať pokročilejšie optimalizačné postupy.

Odporúčania pre ďalší vývoj:

- Preskúmať a integrovať ďalšie optimalizačné techniky na zvýšenie výkonu pri vizualizácii veľkého objemu údajov.
- Navrhnúť a implementovať používateľské rozhranie a zlepšiť celkový používateľský zážitok z aplikácie, najmä v oblasti konfigurácie a ovládania vizualizácie.
- V prípade rozšírenia riešenia založeného na Unreal Engine odporúčame zvážiť priamu integráciu s knižnicou JSROOT s cieľom uľahčiť spracovanie údajov.

# Zoznam použitej literatúry


1. KEHRER, Johannes; HAUSER, Helwig. Visualization and Visual Analysis of Multifaceted Scientific Data: A Survey. *IEEE Transactions on Visualization and Computer Graphics*. 2013, roč. 19, č. 3, s. 495–513. Dostupné z DOI: 10.1109/TVCG.2012.110.
2. DIXON, Steve. A History of Virtual Reality in Performance. *International Journal of Performance Arts and Digital Media*. 2006, roč. 2, č. 1, s. 23–54. Dostupné z DOI: 10.1386/padm.2.1.23/1.
3. DURUKAN, Alper; ARTUN, Huseyin; TEMUR, Atilla. Virtual Reality in Science Education: A Descriptive Review. *Journal of Science Learning*. 2020. ISSN 2614-6568. Dostupné tiež z: https://eric.ed.gov/?id=EJ1267750.
4. HEALY, Kieran. *Data Visualization: A Practical Introduction*. Princeton University Press, 2024. Dostupné tiež z: https://books.google.sk/books?id=v7gfEQAAQBAJ&printsec=frontcover&hl=uk&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false.
5. MÄKELÄ, Antti. *Data Visualisation in Virtual Reality*. JAMK University of Applied Sciences, 2018. Dostupné tiež z: https://www.theseus.fi/bitstream/handle/10024/158327/Makela_Antti.pdf?sequence=1. Bachelor's thesis.
6. EL BEHEIRY, Mohamed; DOUTRELIGNE, Sébastien; CAPORAL, Clément; OSTERTAG, Cécilia; DAHAN, Maxime; MASSON, Jean-Baptiste. Virtual Reality: Beyond Visualization. *Journal of Molecular Biology*. 2019, roč. 431, č. 7, s. 1315–1321. ISSN 0022-2836. Dostupné z DOI: 10.1016/j.jmb.2019.01.033.
7. CIEKANOWSKA, Agata; KISZCZAK-GLIŃSKI, Adam; DZIEDZIC, Krzysztof. Comparative Analysis of Unity and Unreal Engine Efficiency in Creating Virtual Exhibitions of 3D Scanned Models. *Journal of Computer Sciences Institute*. 2021, roč. 20, s. 247–253.

8. KOREČKO, Štefan; VAĽA, Martin; FEKETE, Martin. Visualization of Experimental Data in Web-Based Virtual Reality. In: *Proceedings of the CEUR Workshop*. 2021, zv. 3041, s. 149–152. Dostupné tiež z: `https://ceur-ws.org/Vol-3041/149-152-paper-27.pdf`.
9. MOZILLA. *A-Frame*. [B.r.]. Dostupné tiež z: `https://aframe.io`. Domovská stránka A-Frame.
10. ROOT PROJECT. *JSROOT*. [B.r.]. Dostupné tiež z: `https://root.cern.ch/js/`. Domovská stránka JSROOT.
11. META. *React.js*. [B.r.]. Dostupné tiež z: `https://reactjs.org`. Domovská stránka React.js.
12. *NDMVR*. [B.r.]. Dostupné tiež z: `https://ndmspc.gitlab.io/ndmvr`. Demonštračná stránka NDMVR.
13. EPIC GAMES. *Unreal Engine*. [B.r.]. Dostupné tiež z: `https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation`. Dokumentácia Unreal Engine.
14. AL-BITAR, Mahmoud; ABDAL, Fatemah; AL-JAZZAF, Kofran; AL-SAQRAN, Abdulrahman; KHANAFER, Mounib. The Use of Unreal Engine to Create Virtual Reality Fitness and Gaming Application. In: *Proceedings of the 2019 IEEE 10th GCC Conference & Exhibition (GCC)*. 2019, s. 1–5. Dostupné z DOI: `10.1109/GCC45510.2019.1570521510`.
15. HUO, Yuhao; YANG, Anran; JIA, Qingren; CHEN, Yebin; HE, Biao; LI, Jun. Efficient Visualization of Large-Scale Oblique Photogrammetry Models in Unreal Engine. *ISPRS International Journal of Geo-Information*. 2021, roč. 10, č. 10, Article 643. ISSN 2220-9964. Dostupné z DOI: `10.3390/ijgi10100643`.
16. EPIC GAMES. *Instanced Static Meshes in Unreal Engine*. [B.r.]. Dostupné tiež z: `https://docs.unrealengine.com/5.3/en-US/instanced-static-meshes-in-unreal-engine/`. Oficiálna dokumentácia Unreal Engine.
17. EPIC GAMES. *Adding Global Shaders to Unreal Engine*. [B.r.]. Dostupné tiež z: `https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Rendering/ShaderDevelopment/AddingGlobalShaders/index.html`. Oficiálna dokumentácia Unreal Engine: Compute Shader.
18. UNITY TECHNOLOGIES. *Unity Manual*. [B.r.]. Dostupné tiež z: `https://docs.unity3d.com/Manual/index.html`. Dokumentácia Unity: Základná príručka k hernému nástroju.

19. ISAR, Cosmina. A Glance into Virtual Reality Development Using Unity. *Informatica Economica*. 2018, roč. 22, č. 3, s. 14–22.

20. UNITY TECHNOLOGIES. *Unity Manual: Optimizing Graphics Performance*. [B.r.]. Dostupné tiež z: `https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html`. Dokumentácia Unity: Optimalizácia grafického výkonu.

21. UNITY TECHNOLOGIES. *Unity Scripting API: Graphics.DrawMeshInstanced*. [B.r.]. Dostupné tiež z: `https://docs.unity3d.com/ScriptReference/Graphics.DrawMeshInstanced.html`. Dokumentácia Unity: Graphics.DrawMeshInstanced — vykresľovanie inštancií meshov.

22. NEUMAN-DONIHUE, Elias; JARVIS, Michael; ZHU, Yuhao. FastPoints: A State-of-the-Art Point Cloud Renderer for Unity. *arXiv preprint arXiv:2302.05002*. 2023. Dostupné z arXiv: `2302.05002`.

23. UNITY TECHNOLOGIES. *Unity Manual: Compute Shader*. [B.r.]. Dostupné tiež z: `https://docs.unity3d.com/Manual/class-ComputeShader.html`. Dokumentácia Unity: Compute Shader.

24. SATALIN, Anton. *Improving the Game Development Process on Unreal Engine 5*. 2024. Dostupné tiež z: `https://www.theseus.fi/bitstream/handle/10024/864317/Satalin_Anton.pdf?sequence=2&isAllowed=y`. Bachelor's thesis.

25. ROOT PROJECT. *ROOT Reference Documentation*. [B.r.]. Dostupné tiež z: `https://root.cern.ch/doc/master/index.html`. ROOT Dokumentácia.

26. ABRAMOWICZ, Kamil; BORCZUK, Przemysław. Comparative Analysis of the Performance of Unity and Unreal Engine Game Engines in 3D Games. *Journal of Computer Sciences Institute*. 2024, roč. 30, s. 53–60. Dostupné z DOI: `10.35784/jcsi.5473`.

27. VUORINEN, Tuomas. *Animated Sequence Rendering Performance Comparison: Unity and Unreal Engine*. 2024. Dostupné tiež z: `https://www.theseus.fi/handle/10024/863786`. Bachelor's thesis.

28. EPIC GAMES. *Advanced Android SDK Setup*. [B.r.]. Dostupné tiež z: `https://dev.epicgames.com/documentation/en-us/unreal-engine/advanced-setup-and-troubleshooting-guide-for-using-android-sdk?application_version=5.4`. Oficiálna dokumentácia Unreal Engine.

29. EPIC GAMES. *Setting Up Visual Studio Development Environment for C++ Projects in Unreal Engine*. [B.r.]. Dostupné tiež z: `https://dev.epicgames.com/documentation/en-us/unreal-engine/setting-up-visual-studio-development-environment-for-cplusplus-projects-in-unreal-engine`. Oficiálna dokumentácia Unreal Engine: Nastavenie Visual Studio pre C++ vývoj.

30. MICROSOFT LEARN. *Getting Started with Visual Studio Tools for Unity*. [B.r.]. Dostupné tiež z: `https://learn.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/getting-started-with-visual-studio-tools-for-unity?pivots=windows`. Oficiálna dokumentácia Microsoft: Nastavenie Visual Studio pre Unity.

# Zoznam skratiek

**CPU** Central processing unit.

**FPS** Frames Per Second.

**GPU** Graphics processing unit.

**JSROOT** JavaScript ROOT.

**MB** Megabyte.

**NDMVR** N-DiMensional Virtual Reality.

**RAM** Random-access memory.

**UE** Unreal Engine.

**VR** Virtual Reality.

# Zoznam príloh

# A Používateľská príručka

Táto príručka je stručným návodom na používanie aplikácií vytvorených v prostredí Unreal Engine a Unity.

Obsahuje základné informácie o tom, ako otvoriť projekt, nakonfigurovať potrebné komponenty, spustiť simuláciu a vytvoriť build verziu aplikácie.

Návod je zameraný na praktické kroky potrebné na používanie systému. Je určený pre používateľov, ktorí chcú aplikáciu spustiť a testovať bez zásahu do zdrojového kódu.

## Používateľská príručka: Unreal Engine

Táto časť príručky sa venuje používaniu verzie aplikácie vytvorenej v prostredí Unreal Engine.

### Otvorenie projektu

Projekt je možné otvoriť dvojklikom na súbor `ndmvr.uproject`, ktorý sa nachádza v koreňovom adresári.

Po jeho spustení sa otvorí Unreal Editor a projekt by sa mal automaticky načítať a skompilovať.

V prípade, že sa pri otváraní vyskytnú chyby, odporúča sa vykonať nasledovný postup obnovy:

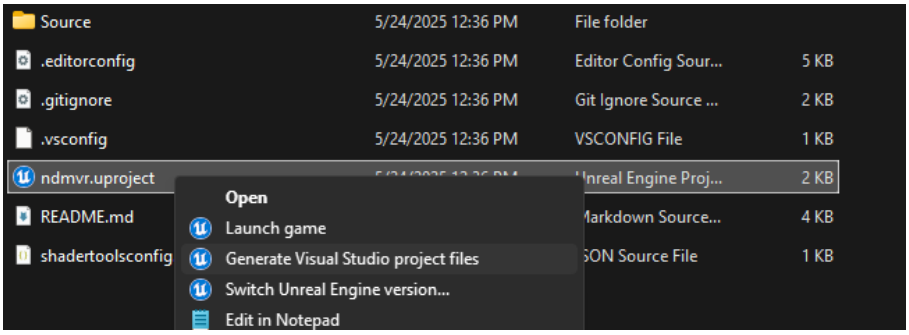
1. Odstráňte všetky priečinky, ktoré boli automaticky vygenerované editorom. V adresári by mali zostať iba tieto tri priečinky: `Config`, `Content` a `Source`. Odporúča sa taktiež odstrániť existujúci súbor `ndmvr.sln`, ak je prítomný. Ostatné súbory nie je potrebné meniť.

2. Kliknite pravým tlačidlom myši na súbor `ndmvr.uproject` a zvoľte možnosť `Generate Visual Studio project files` (obr. A.1).

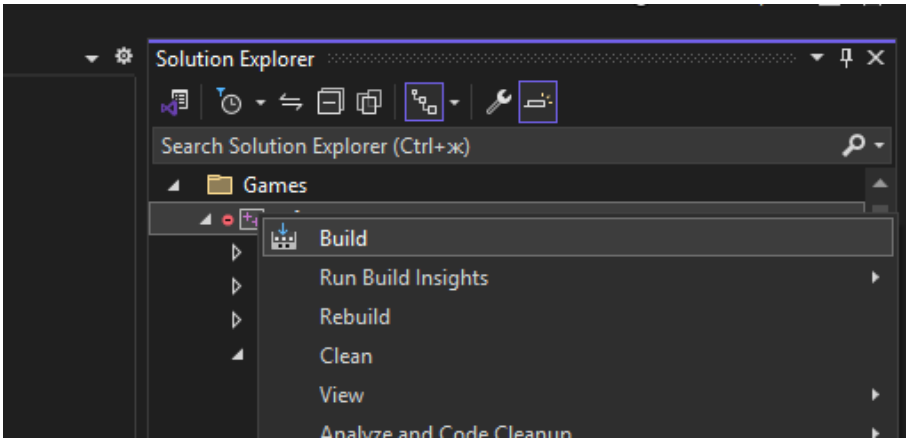
3. Po úspešnom generovaní sa v priečinku objaví súbor `ndmvr.sln`, ktorý následne otvorte v **Visual Studio**.

4. V prostredí Visual Studio kliknite pravým tlačidlom na projekt `ndmvr` a zvoľte možnosť `Build` (obr. A.2).

5. Po úspešnom zostavení je možné projekt znova otvoriť cez `ndmvr.uproject` bez chýb.




Obrázok A.1: Generovanie Visual Studio súborov zo súboru `ndmvr.uproject`




Obrázok A.2: Zostavenie projektu v prostredí Visual Studio

## Zoznámenie so scénou

Po otvorení `Scene.umap` v Unreal Engine sa načíta preddefinovaná scéna obsahujúca nasledovné komponenty:

- `DirectionalLight` – osvetlenie scény.
- `Floor` – základná plocha (Landscape).
- `Histogram` – vizualizačný komponent (`HistogramRenderer`).
- `Player` – používateľská entita (`VRPawn`).

- `SkySphere` – pozadie a obloha.

Tieto komponenty sú súčasťou základnej scény. Ďalšie voliteľné prvky, ako napríklad `CameraMovementActor`, je potrebné pridať manuálne.

## Nastavenie komponentu `Histogram`

Po kliknutí na komponent `Histogram` v okne `Outliner` sa jeho vlastnosti zobrazia v paneli `Details`. Ak tieto panely nie sú viditeľné, je možné ich aktivovať v hlavnom menu cez `Window → Outliner` a `Window → Details`. Na obrázku A.3 sú zobrazené hlavné parametre komponentu Histogram a ich význam je nasledovný:

- `Server Url` – adresa, z ktorej sa načítavajú údaje v prípade sieťovej komunikácie.
- `Use Server Data` – ak je zapnuté, histogram bude generovaný na základe údajov prijatých zo servera.
- `Use Web Socket` – ak je zapnuté, na komunikáciu so serverom sa použije WebSocket namiesto HTTP.

**Bins Settings:**

- `Bin Mesh` – výber objektu (mesh), ktorý sa použije pre jednotlivé biny.
- `Bin Padding` – vzdialenosť medzi binmi v troch smeroch (X, Y, Z) v centimetroch.
- `Max Draw Distance` – maximálna vzdialenosť, na ktorú budú biny viditeľné v scéne.

**Bins Settings for Manual Visualization:**

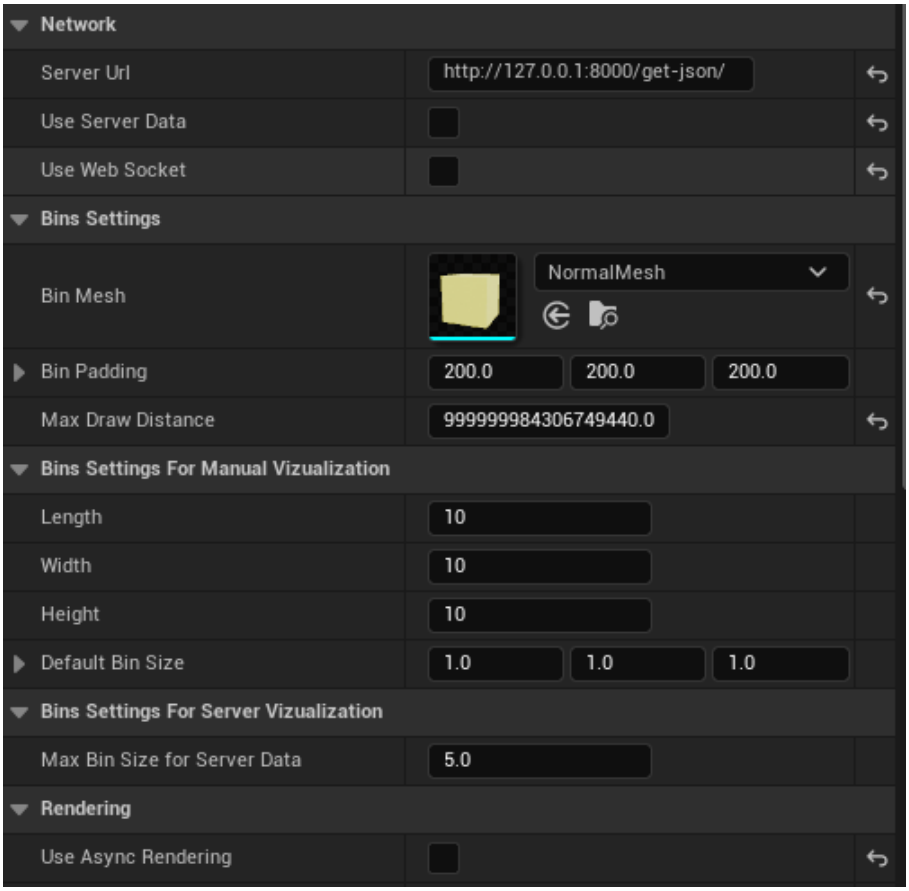
- `Length, Width, Height` – počet binov v jednotlivých smeroch.
- `Default Bin Size` – mierka (scale) každého binu v troch smeroch (X, Y, Z). Hodnota `1.0` znamená pôvodnú veľkosť objektu (napr. 1 meter), zatiaľ čo `2.0` bin zväčší dvojnásobne.

**Bins Settings for Server Visualization:**

- `Max Bin Size for Server Data` – maximálna veľkosť binu pri použití údajov zo servera. Hodnota slúži na obmedzenie zväčšovania binov – v 3D histogramoch by sa bez limitu mohli objekty prekrývať, zatiaľ čo pri 1D alebo 2D histogramoch môže byť vhodné použiť vyššiu hodnotu.

**Rendering:**

- `Use Async Rendering` – ak je zapnuté, histogram sa generuje postupne (napr. po 100 bínov), čo umožňuje plynulejšie načítanie veľkých datasetov.




Obrázok A.3: Nastavenia komponentu Histogram

## Pridanie a nastavenie komponentu `CameraMovementActor`

Komponent `CameraMovementActor` nie je súčasťou scény predvolene.

Ak ho chcete pridať, postupujte nasledovne:

1. Otvorte `Content Browser` v Unreal Engine.
2. Prejdite do sekcie `C++ Classes → ndmvr`.
3. Nájdite položku `CameraMovementActor` a presuňte ju do scény. Pozícia nie je dôležitá – komponent funguje nezávisle od umiestnenia.

Po pridaní sa v paneli `Details` zobrazia jeho parametre rozdelené do dvoch skupín:

#### `Benchmark` nastavenia

Tieto parametre sa používajú len v prípade, že je aktivované pole `Use Benchmark Trajectory`. Tu je stručný opis nastavení:

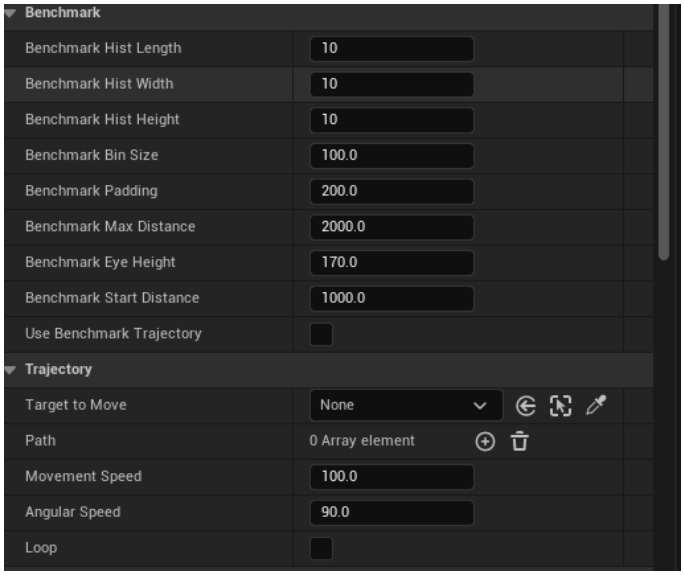
- `Benchmark Hist Length / Width / Height` – rozmery histogramu počas testovania.
- `Benchmark Bin Size / Padding` – veľkosť a vzdialenosť medzi binmi.
- `Benchmark Max Distance` – maximálna dĺžka pohybu po diagonále histogramu. Presný priebeh diagonály je popísaný v hlavnej časti práce 6.3.
- `Benchmark Eye Height` – výška kamery od podlahy počas simulácie.
- `Benchmark Start Distance` – počiatočná vzdialenosť od histogramu.
- `Use Benchmark Trajectory` – ak je aktivované, simulácia použije automaticky generovanú testovaciu trajektóriu. Podrobnosti sú uvedené v hlavnej časti práce 6.3.

#### `Trajectory` nastavenia

Táto sekcia komponentu `CameraMovementActor` umožňuje manuálne definovať trajektóriu, po ktorej sa bude pohybovať vybraný objekt v scéne.

- `Target to Move` – objekt, ktorý sa bude pohybovať (napr. `Player`).
- `Path` – zoznam bodov definujúcich trasu. Každý bod obsahuje súradnice a rotáciu (v stupňoch).
- `Movement Speed` – rýchlosť pohybu po trajektórii.
- `Angular Speed` – rýchlosť otáčania.
- `Loop` – ak je aktivované, pohyb sa bude po dokončení cykliť.

Na obrázku A.4 je znázornený príklad, ako vyzerá konfigurácia trajektórie v editore Unreal Engine.




Obrázok A.4: Nastavenia komponentu CameraMovementActor – príklad trajektórie

### Vytvorenie build verzie

Build verziu je možné vytvoriť priamo v editore Unreal Engine nasledovne:

1. V hornej lište kliknite na tlačidlo `Platforms`.
2. Vyberte cieľovú platformu (napr. `Windows` alebo `Android`).
3. Zvoľte možnosť `Package Project`.
4. Vyberte cieľový priečinok, kam sa má build uložiť.

Po dokončení procesu bude v zvolenom priečinku vytvorený spustiteľný súbor, pripravený na testovanie alebo distribúciu.

## Používateľská príručka: Unity

Táto časť príručky sa venuje používaniu verzie aplikácie vytvorenej v prostredí Unity.

### Otvorenie projektu

Projekt je možné otvoriť pomocou **Unity Hub**. Stačí vybrať koreňový adresár projektu (`ndmvr-unity/`) a otvoriť ho vo verzii **Unity 6000.0.38f1** alebo kompatibilnej.

### Zoznámenie so scénou

Po otvorení súboru `Scene.unity` sa načíta základná scéna so všetkými potrebnými komponentmi.

Zoznam hlavných prvkov zobrazený v okne `Hierarchy` zahŕňa:

- `Player` – používateľská entita, reprezentovaná pomocou XR Rig.
- `Directional Light` – osvetlenie scény.
- `Global Volume` – efekty post-process (napr. tonemapping, farby).
- `Histogram` – komponent zodpovedný za vizualizáciu údajov (Histogram-Renderer).
- `Floor` – základná podlaha scény.
- `EventSystem` – systém interakcie.
- `XR Interaction Manager` – spracováva vstupy z VR zariadenia.
- `CameraMovementActor` – komponent pre testovací pohyb kamery.

`CameraMovementActor` je súčasťou scény, ale je predvolene deaktivovaný.

Ak ho chcete použiť, kliknite na jeho názov v okne `Hierarchy`, čím sa otvorí jeho konfigurácia v paneli `Inspector`.

Potom je možné ho aktivovať a upraviť jeho nastavenia.

### Nastavenie komponentu `Histogram`

`Histogram` obsahuje rovnaké parametre ako vo verzii pre Unreal Engine.

Podrobný popis jednotlivých nastavení sa nachádza v sekcii **Nastavenie komponentu `Histogram`** v časti venovanej Unreal Engine (str. 70).

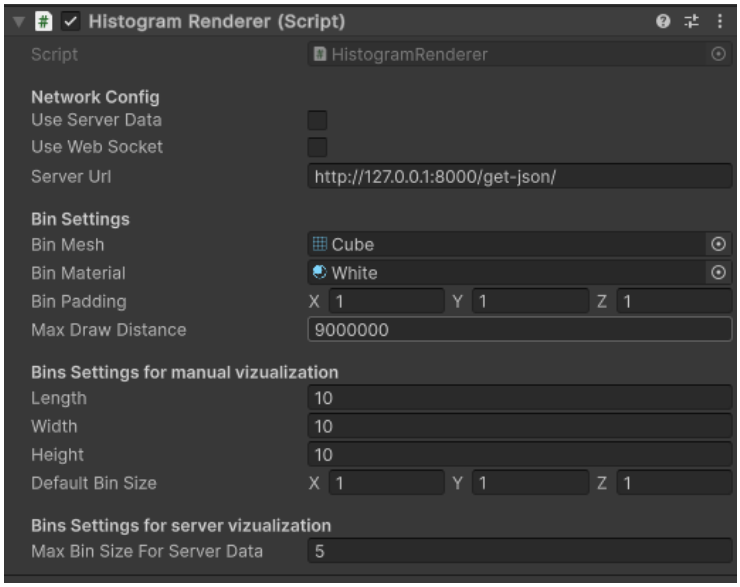
Na obrázku A.5 je zobrazený príklad konfigurácie komponentu v prostredí Unity.

### Nastavenie komponentu `CameraMovementActor`

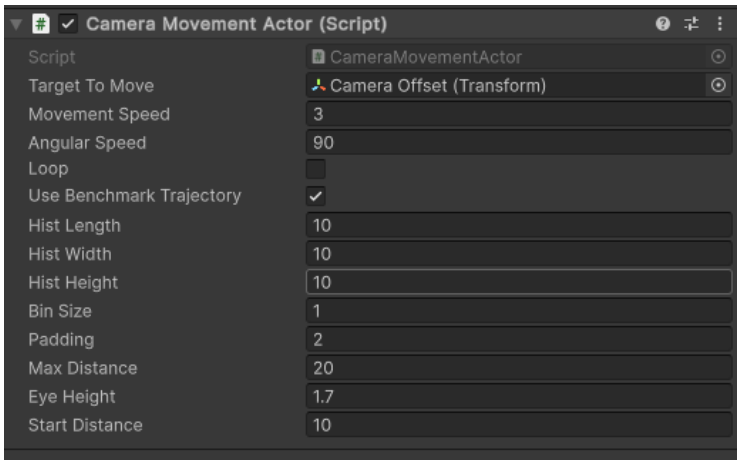
Nastavenia `CameraMovementActor` sú funkčne takmer identické s Unreal verziou, no názvy polí sa môžu mierne líšiť.

Preto je možné použiť rovnaký návod ako v časti Unreal Engine (str. 72)

Na obrázku A.6 je zobrazený príklad konfigurácie komponentu v prostredí Unity.




Obrázok A.5: Nastavenia komponentu `Histogram` v prostredí Unity




Obrázok A.6: Nastavenia komponentu `CameraMovementActor` v prostredí Unity

## Vytvorenie build verzie

Build verziu aplikácie je možné vytvoriť priamo v editore Unity pomocou nasledujúcich krokov:

1. V ovládacom paneli kliknite na `File → Build Settings`.
2. V ľavej časti okna vyberte cieľovú platformu (napr. `Windows` alebo `Android`).
3. Ak platforma ešte nebola aktívna, kliknite na `Switch Platform` a počkajte na dokončenie procesu.
4. Kliknite na `Build`.

5. Zvoľte cieľový priečinok, kam sa má aplikácia uložiť, a potvrďte.

Po úspešnom zostavení sa v zadanom priečinku vytvorí spustiteľný súbor (`.exe` alebo `.apk`), pripravený na testovanie.

# B Systémová príručka

Táto časť obsahuje systémové príručky pre dve implementácie vyvinuté v tejto práci, jednu v prostredí Unreal Engine a druhú v prostredí Unity. Cieľom je stručne opísať štruktúru projektov a objasniť účel hlavných komponentov, aby bolo možné systém v budúcnosti upravovať alebo ďalej rozvíjať. Podkapitoly sa venujú samostatne každému prostrediu.

## Systémová príručka: Unreal Engine

Táto časť dokumentácie sa venuje projektu vytvorenému v prostredí Unreal Engine. Jeho hlavnou úlohou je vizualizovať viacrozmerné histogramy v prostredí virtuálnej reality (VR). Implementácia podporuje aj pripojenie k externým dátovým zdrojom prostredníctvom internetovej komunikácie, konkrétne cez WebSocket alebo HTTP požiadavky.

Unreal Engine bol v tomto prípade použitý ako jedna z alternatívnych platforiem na vývoj VR aplikácií, ktoré umožňujú interaktívne zobrazenie veľkého množstva údajov.

### Technologické požiadavky

Projekt bol vytvorený v prostredí **Unreal Engine 5.4.4**, ktorý umožňuje vývoj pomocou:

- **C++** – hlavná logika systému sa nachádza v priečinku `Source/`.
- **Blueprint** – vizuálne programovanie použité pri tvorbe komponentu `VRPawn.uasset`, ktorý predstavuje základnú entitu používateľa vo VR prostredí – vrátane pohybu, kamery, sledovania rúk a interakcie s prostredím.

Závislosti projektu:

- **Visual Studio:** Na vývoj a zostavovanie projektu v prostredí Unreal Engine je potrebné mať nainštalované **Visual Studio 2019** (alebo novšiu verziu) s podporou vývoja v jazyku C++. Pri inštalácii je potrebné zahrnúť

komponent `Mobile development with C++`, ktorý je nevyhnutný pre zostavenie build verzie pre VR zariadenia. Podrobný návod na nastavenie vývojového prostredia je dostupný na stránke spoločnosti Epic Games [29].

- **WebSocket Plugin:** Pre komunikáciu so serverom sa využíva *Experimental WebSocket Networking Plugin*, ktorý je už aktivovaný v projekte a nevyžaduje manuálne zapínanie.

- **Android SDK:** Pre export na VR zariadenia (napr. Meta Quest 2) bolo použité **Android Studio – Koala Feature Drop 2024.1.2**. Konfigurácia prebehla podľa odporúčaného postupu od Epic Games [28].

## Štruktúra projektu

Vo výpise B.1 sú uvedené iba tie súbory a priečinky, ktoré boli priamo upravené alebo vytvorené v rámci tohto projektu.

```
ndmvr/
|--Config/
|  |--DefaultEngine.ini
|--Content/
|  |--Blueprints/
|  |  |--VRPawn.uasset
|  |--Maps/
|  |  |--Scene.umap
|  |--Materials/
|  |--Mesh/
|--Source/
|  |--ndmvr/
|  |  |--Broker.h
|  |  |--Broker.cpp
|  |  |--CameraMovementActor.h
|  |  |--CameraMovementActor.cpp
|  |  |--HistogramDataLoader.h
|  |  |--HistogramDataLoader.cpp
|  |  |--HistogramRenderer.h
|  |  |--HistogramRenderer.cpp
|  |  |--HistogramRenderUtils.h
|  |  |--HistogramRenderUtils.cpp
|  |  |--ndmvr.Build.cs
```

```
|--ndmvr.uproject
```

Výpis B.1: Štruktúra projektu ndmvr

Popis hlavných častí:

- `Config/DefaultEngine.ini` – obsahuje základné konfiguračné nastavenia pre spustenie projektu.
- `Content/Blueprints/VRPawn.uasset` – Blueprint reprezentujúci používateľa vo VR prostredí (kamera, ovládače).
- `Content/Maps/Scene.umap` – hlavná scéna použitá na testovanie vizualizácie histogramu.
- `Content/Materials/` – materiály pre vizualizáciu binov histogramu vrátane farebného mapovania.
- `Content/Mesh/` – 3D objekty predstavujúce jednotlivé biny v priestore.
- `Source/ndmvr/` – obsahuje implementáciu logiky systému:
  - `Broker.[cpp|h]` – komunikácia so serverom cez WebSocket.
  - `HistogramDataLoader.[cpp|h]` – načítavanie a spracovanie údajov a komunikácia so serverom cez HTTP.
  - `HistogramRenderer.[cpp|h]` – vykresľovanie histogramu pomocou GPU inštancií.
  - `HistogramRenderUtils.[cpp|h]` – výpočty veľkosti, farby, pozície binov.
  - `CameraMovementActor.[cpp|h]` – automatizovaný pohyb kamery a zber výkonových údajov.
- `ndmvr.Build.cs` – konfiguračný súbor, v ktorom sú definované modulové závislosti projektu pre C++ kompiláciu. Pre správnu funkcionalitu bolo potrebné aktivovať moduly ako `HTTP`, `Json`, `JsonUtilities` a `WebSockets`. Aktivácia týchto modulov je oddelená od samotného pluginu WebSocket, ktorý sa zapína cez Unreal Editor.
- `ndmvr.uproject` – projektový súbor Unreal Engine.

**Poznámky k možnému rozšíreniu**

Aktuálna verzia systému vyžaduje, aby boli všetky parametre scény nastavované manuálne priamo v prostredí Unreal Editor.

Všetky vstupné údaje – ako je URL servera, spôsob komunikácie (HTTP/WebSocket), veľkosť histogramu a režim simulácie – sa zadávajú ručne cez `Details Panel`. Po vytvorení build verzie už nie je možné tieto parametre zmeniť.

**Odporúčané zlepšenia pre budúci vývoj:**

- Implementácia **používateľského rozhrania (UI)**, ktoré by umožňovalo nastavovať parametre histogramu priamo počas používania aplikácie.
- Pridanie **desktopového režimu** – aktuálny `VRPawn` podporuje iba ovládanie pomocou VR headsetu a ovládačov. V desktopovom prostredí (bez VR) nie je momentálne podporovaný voľný pohyb používateľa.

  Navrhuje sa vytvoriť alternatívneho hráča (napr. `DesktopPawn`), ktorý by podporoval klasické ovládanie pomocou klávesnice a myši.

# Systémová príručka: Unity

Táto časť dokumentácie sa venuje projektu vytvorenému v prostredí Unity. Cieľom implementácie je vizualizácia histogramov vo VR prostredí s podporou internetovej komunikácie cez WebSocket alebo HTTP. Unity predstavuje jednu z alternatívnych platforiem, ktorá umožňuje rýchly vývoj a testovanie aplikácií pre zariadenia virtuálnej reality.

## Technologické požiadavky

Projekt bol vytvorený v prostredí **Unity 6000.0.38f1** s využitím programovacieho jazyka **C#**.

Celá aplikačná logika, vrátane vykresľovania histogramu a komunikácie so serverom, je implementovaná prostredníctvom C# skriptov.
Závislosti projektu:

- **Visual Studio:** Pre vývoj a ladenie projektu v prostredí Unity je potrebné mať nainštalované **Visual Studio** s podporou nástroja `Visual Studio Tools for Unity`. Okrem toho je potrebné počas inštalácie zahrnúť komponent `Mobile development with C++`, ktorý je potrebný pre export aplikácie na VR zariadenia. Oficiálny návod na inštaláciu a nastavenie je dostupný tu [30].

- **WebSocketSharp:** Na komunikáciu cez WebSocket sa používa knižnica `WebSocketSharp.dll`, ktorá je už súčasťou projektu v priečinku `Assets/Plugins/`. Nie je potrebná žiadna dodatočná inštalácia ani konfigurácia.
- **Android SDK:** Pre export na VR zariadenia (napr. Meta Quest 2) bolo použité **Android Studio – Koala Feature Drop 2024.1.2**. Nastavenie SDK pre Unity bolo identické ako v prípade Unreal Engine, podľa oficiálneho postupu [28].

## Štruktúra projektu

Vo výpise B.2 sú uvedené iba tie súbory a priečinky, ktoré boli priamo upravené alebo vytvorené v rámci tohto projektu.

```
ndmvr-unity/
|--Assets/
|  |--Scenes/
|  |--Scripts/
|  |  |--Broker.cs
|  |  |--CameraMovementActor.cs
|  |  |--HistogramDataLoader.cs
|  |  |--HistogramRenderer.cs
|  |  |--HistogramRenderUtils.cs
|  |  |--VerticalMovementVR.cs
|  |--Materials/
|  |--Prefabs/
|--Packages/
|--ProjectSettings/
|--README.md
```

Výpis B.2: Štruktúra projektu ndmvr-unity

Popis hlavných častí:

- `Assets/Scenes/` – katalóg so scénou, v ktorej sa vizualizácia vykonáva. Obsahuje aj ďalšie súbory, ktoré boli automaticky vygenerované pri použití scény.
- `Assets/Scripts/` – obsahuje hlavnú logiku systému:
  - `Broker.cs` – komunikácia so serverom cez WebSocket.

  - `HistogramDataLoader.cs` – načítavanie a spracovanie údajov a komunikácia so serverom cez HTTP.
  - `HistogramRenderer.cs` – vykresľovanie histogramu pomocou `Graphics.DrawMeshInstanced`.
  - `HistogramRenderUtils.cs` – výpočty pozície, veľkosti a farby jednotlivých binov.
  - `CameraMovementActor.cs` – automatizovaný pohyb kamery a zber výkonových údajov.
  - `VerticalMovementVR.cs` – vertikálne pohyby hráča vo VR prostredí.

- `Assets/Materials/` – materiály pre farebné mapovanie vizualizovaných údajov.
- `Assets/Prefabs/` – preddefinované objekty pripravené na použitie v scéne.
- `Packages/` – závislosti projektu, vrátane XR Toolkit a ďalších modulov.
- `ProjectSettings/` – globálne nastavenia Unity projektu (XR, input, build platformy).

**Poznámky k možnému rozšíreniu**

Rovnako ako v prípade verzie pre Unreal Engine, odporúča sa doplniť používateľské rozhranie (UI) na konfiguráciu parametrov počas behu aplikácie a pridať podporu desktopového režimu pre ovládanie bez VR zariadenia.

Ďalším možným zlepšením je implementácia asynchrónneho vykresľovania a optimalizácia inštancovaného zobrazovania objektov (napr. `Graphics.DrawMeshInstanced`) s cieľom dosiahnuť výkon porovnateľný s `HierarchicalInstancedStaticMeshComponent` v Unreal Engine.