НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.4:004.8

ПОГОДЖЕНО»
Декан факультету
інформаційних технологій
Болбот І. М., д.п.н., професор
_____

«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»
Завідувач кафедри комп'ютерних наук
Голуб Б.Л., к.т.н., доцент
_____

___ _____ 2024 р. **____ _____ 2024 р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Програмне забезпечення оптимізації ігрових об'єктів в Unreal Engine 5

Спеціальність _____121 «Інжереія програмного забезпечення»_____
(код і назва)

Освітня програма ____Програмне забезпечення інформаційних систем_____
(назва)

Орієнтація освітньої програми ___освітня-професійна_____
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

_____ _____ _____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

_____ _____ ___Міловідов Ю.О.____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконала _____ _Тарасенко Р.Ю._____
(підпис) (ПІБ студента)

КИЇВ-2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FPS (Frames Per Second) — кількість кадрів в секунду, показник продуктивності графічного процесу.

LOD (Level of Detail) — рівень деталізації, метод оптимізації, що дозволяє зменшувати складність моделей на відстані.

Culling — метод виключення невидимих або зайвих об'єктів з рендерингу для зменшення навантаження.

OLAP (Online Analytical Processing) — технологія багатовимірного аналізу даних.

ETL (Extract, Transform, Load) — процес вилучення, трансформації та завантаження даних.

SSAS (SQL Server Analysis Services) — служба для побудови багатовимірних баз даних та OLAP-кубів.

SSIS (SQL Server Integration Services) — служба для інтеграції даних та побудови потоків даних.

SSRS (SQL Server Reporting Services) — служба для створення звітів на основі аналітичних даних.

KPI (Key Performance Indicator) — ключові показники ефективності для оцінки досягнення цілей.

Unreal Engine 5 — сучасний ігровий двигун, що використовується для створення графічно складних та оптимізованих ігор.

Mesh Simplification — спрощення сітки 3D-моделей для оптимізації їх використання.

Batching — групування об’єктів для зменшення кількості викликів рендерингу.

Optimization Fact — таблиця фактів оптимізації, що містить ключові метрики продуктивності.

Memory Improvement — зменшення обсягу використаної пам’яті після оптимізації.

Data Flow — потік даних для передачі та обробки інформації між джерелами та сховищами.

SQL (Structured Query Language) — мова запитів для роботи з базами даних.

TimeDim — таблиця вимірів часу у сховищі даних.

ObjectDim — таблиця вимірів ігрових об'єктів у сховищі даних.

OptimizationDim — таблиця вимірів оптимізації у сховищі даних.

FileDim — таблиця вимірів файлів у сховищі даних.

FPSBefore — показник FPS до застосування оптимізації.

FPSAfter — показник FPS після застосування оптимізації.

MemoryUsageBefore — обсяг використаної пам’яті до оптимізації.

MemoryUsageAfter — обсяг використаної пам’яті після оптимізації.

Data Warehouse (DW) — сховище даних для централізованого зберігання інформації.

Cube Wizard — майстер для створення багатовимірного кубу у SSAS.

Performance Analyst — аналітик продуктивності, відповідальний за оцінку впливу оптимізації.

System Administrator — адміністратор, який налаштовує системи моніторингу та автоматизації.

Game Developer — розробник ігор, відповідальний за технічну оптимізацію ігрових об'єктів.

Graphic Designer — графічний дизайнер, який оптимізує текстури та графічні ресурси.

Git — система контролю версій для відстеження змін у проєктах.

Unreal Insights — інструмент профілювання для аналізу продуктивності у Unreal Engine 5.

RDBMS (Relational Database Management System) — система керування реляційними базами даних.

OLTP (Online Transaction Processing) — онлайн-обробка транзакцій, що використовується для роботи з оперативними базами даних.

AssetFiles — таблиця активів, що містить інформацію про файли ігрових об’єктів.

3D-модель — графічний об’єкт, що складається з геометричних елементів.

Render Pipeline — графічний конвеєр, який відповідає за рендеринг сцени.

LOD Group — група рівнів деталізації для управління якістю відображення.

Texture Compression — метод оптимізації, що зменшує розмір текстур.

Optimization Logs — журнали оптимізації, що містять інформацію про процеси оптимізації.

ВСТУП

Сучасна ігрова індустрія є однією з найбільш динамічно розвиваючихся галузей інформаційних технологій. Завдяки швидкому прогресу обчислювальної техніки та зростаючим очікуванням користувачів, розробка відеоігор перетворилася на складний процес, що вимагає застосування передових технологій і методологій. Одним із ключових викликів у цьому контексті є забезпечення продуктивності та ефективності ігрового контенту, особливо у середовищах, де використовується високополігональна 3D-графіка, складні анімації та інші ресурсоємні елементи.

Оптимізація ігрових об'єктів є критично важливою складовою розробки ігор, адже дозволяє забезпечити плавний ігровий процес навіть на пристроях із обмеженими апаратними ресурсами. Це питання набуває особливої актуальності у світлі сучасних тенденцій до підвищення якості графіки та інтерактивності ігор. Невиконання оптимізаційних завдань може призводити до затримок у рендерингу, низького рівня FPS (Frames Per Second), тривалого часу завантаження та зниження загальної якості геймплейного досвіду.

У сучасних умовах розвитку ігрової індустрії оптимізація ігрових об’єктів стає важливим аспектом забезпечення високої продуктивності графіки та плавного ігрового процесу. Це особливо актуально для складних 3D-сцен, які вимагають великого обсягу обчислень та значних апаратних ресурсів. Програмне забезпечення, орієнтоване на оптимізацію ігрових об'єктів, дозволяє зменшити навантаження на систему, забезпечити стабільність роботи рушія та покращити загальний користувацький досвід. Unreal Engine 5, як один із провідних рушіїв у сфері розробки ігор, пропонує широкий спектр інструментів для досягнення цієї мети, серед яких LOD, Texture Compression, механізми стиснення текстур та об'єктно-орієнтоване моделювання.

Предметом даного дослідження є аналіз існуючих рішень та інструментів оптимізації, доступних у Unreal Engine 5, а також їхній вплив на продуктивність ігрових сцен. У рамках роботи досліджуються методи зменшення обчислювального навантаження через управління текстурами, геометрією об’єктів та аналітичними процесами на основі технологій OLAP і Data Mining. Особлива увага приділяється виявленню ефективних підходів для оптимізації ігрових ресурсів із урахуванням специфіки використання даних у реальному часі.

Об'єктом дослідження є програмні рішення та технології, які забезпечують оптимізацію ігрових об’єктів та покращення продуктивності рушія Unreal Engine 5. Дослідження включає аналіз функціональності таких інструментів, як

OLAP для аналітики ігрових даних та алгоритмів Data Mining, які дозволяють структурувати й оптимізувати ігрові сцени на основі великих обсягів даних.

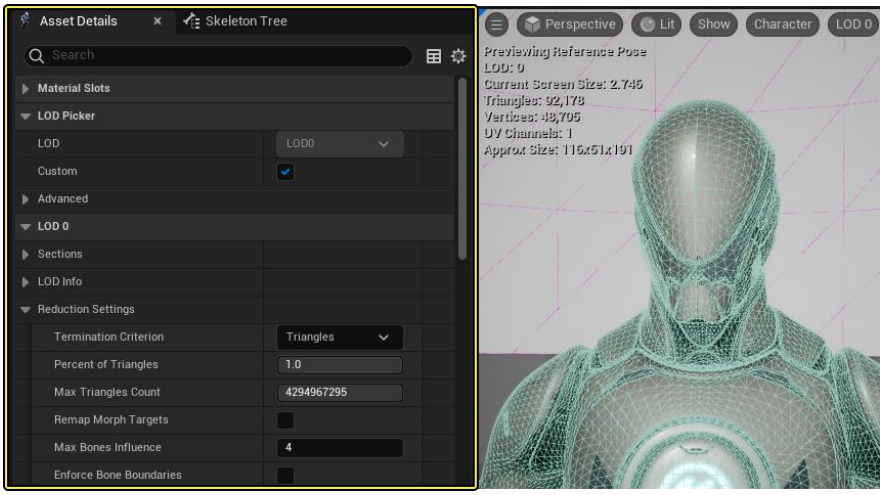
Мета роботи полягає в дослідженні, аналізі та впровадженні ефективних методів оптимізації, які забезпечують максимальну продуктивність ігрових проєктів. Це досягається шляхом інтеграції сучасних алгоритмів аналізу даних і інструментів оптимізації в робочі процеси розробки ігор.

СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальна характеристика предметної області

Предметна область оптимізації ігрових об'єктів є однією з ключових складових у розробці сучасних відеоігор. Ігрова індустрія, яка стрімко розвивається, вимагає від розробників створення складних і деталізованих віртуальних світів, що можуть бути інтерактивно відтворені в реальному часі. При цьому головним завданням є забезпечення високої продуктивності гри на різноманітних апаратних платформах, від потужних ігрових ПК до мобільних пристроїв.

Складність розробки ігрових об'єктів



Ігрові об'єкти, такі як 3D-моделі персонажів, будівель, природних об'єктів та інших елементів оточення, формують основу будь-якої відеогри. Їх створення включає численні етапи, зокрема:

- Моделювання геометрії (визначення форми, кількості полігонів, складності сітки).

- Текстурування (додавання зображень для деталізації поверхонь).

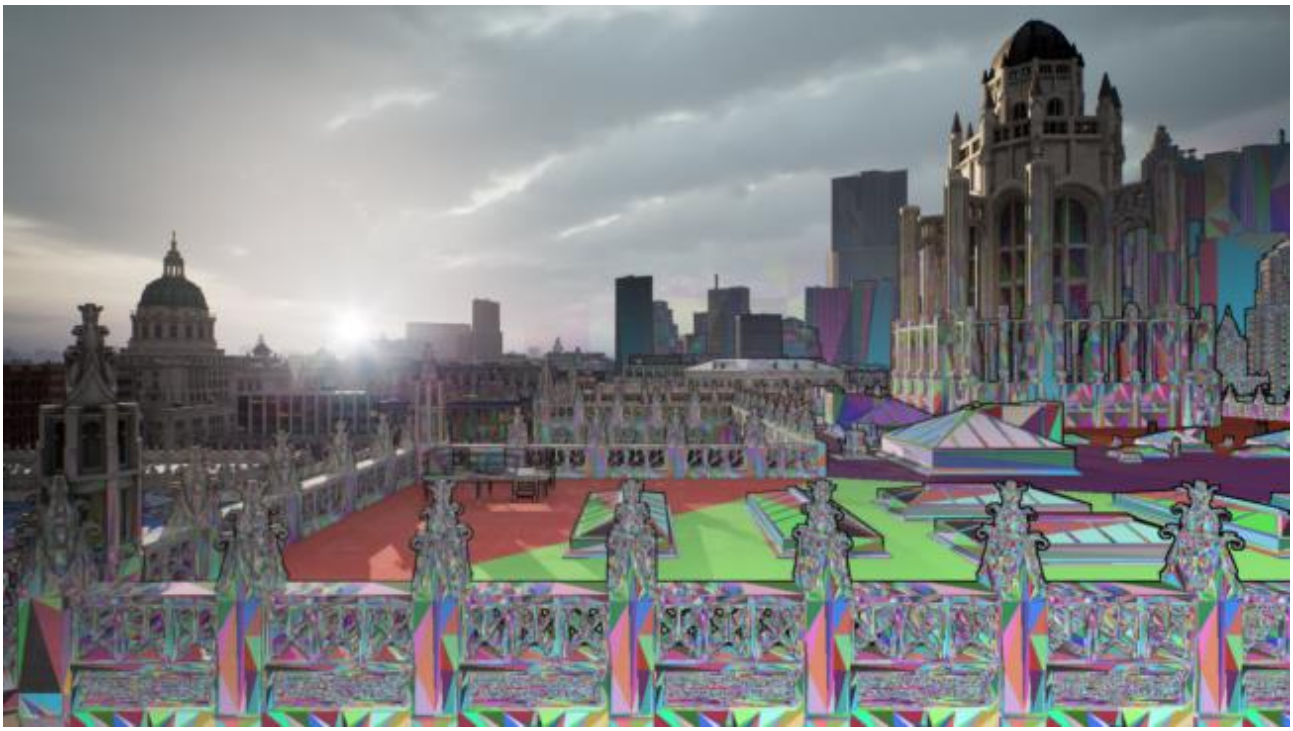
- Анімація (створення рухів для персонажів чи об'єктів).

- Освітлення (налаштування взаємодії з джерелами світла в ігровій сцені).

Локація під різними модами перегляду

Зростання апаратних можливостей користувацьких пристроїв дозволяє збільшувати рівень деталізації графіки, але водночас ускладнює забезпечення стабільної продуктивності. Наприклад, високополігональні моделі та текстури з високою роздільною здатністю значно збільшують обсяг використаної пам’яті, навантаження на GPU (графічний процесор) та час рендерингу.

Потреба в оптимізації



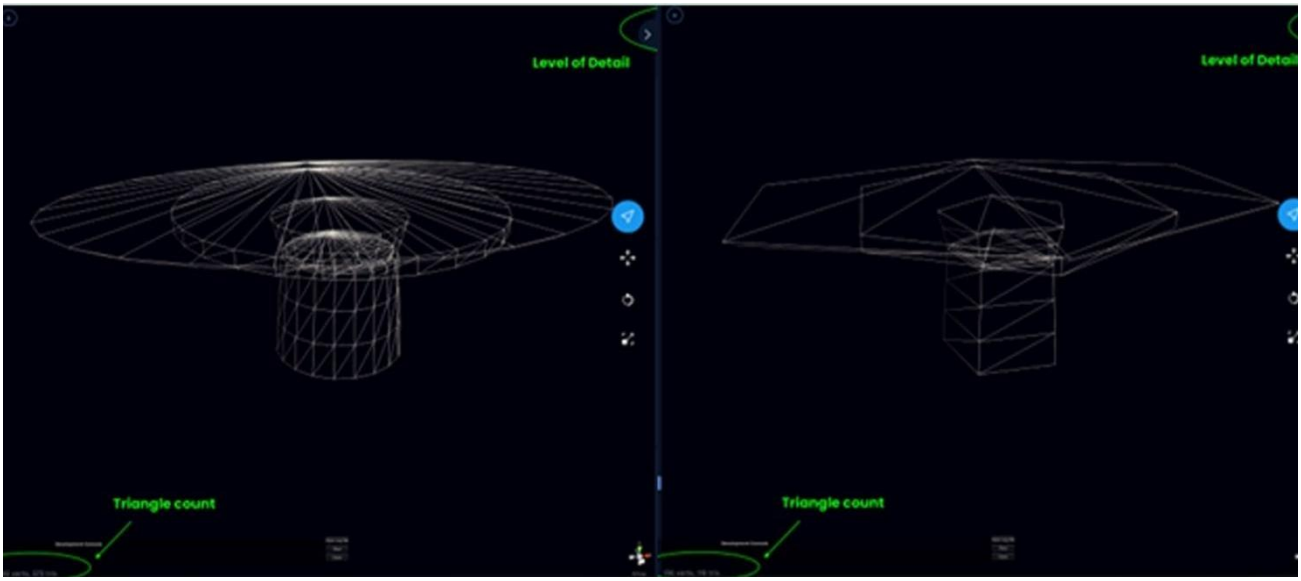
«Відкритий світ з використанням методу оптимізації Nanite System»

Оптимізація ігрових об'єктів стає необхідною для забезпечення плавного ігрового процесу. Це особливо актуально у випадках:

- Масштабних багатокористувацьких ігор (Massively Multiplayer Online Games, MMO), де велика кількість гравців взаємодіє з об'єктами в одній сцені.

- Реалістичних симуляторів, які вимагають високої точності у відтворенні об’єктів і фізичних взаємодій.

- Ігор із відкритим світом, де одночасно обробляються численні об'єкти на великій території.



Спосіб оптимізації LOD

Основні аспекти оптимізації включають:

1. Зменшення кількості полігонів у моделях. Спрощення сітки об’єкта (mesh simplification) знижує навантаження на GPU.

2. Використання технології LOD (Level of Detail). Цей підхід дозволяє зменшувати рівень деталізації об’єктів, які знаходяться далеко від камери.

3. Компресія текстур. Використання стислих форматів текстур (наприклад, DXT, ASTC) знижує обсяг використаної пам'яті без значних втрат якості.

4. Culling. Виключення невидимих об'єктів зі сцени за допомогою технологій Frustum Culling або Occlusion Culling.

5. Аналіз освітлення. Заміна динамічного освітлення статичним у випадках, коли це можливо.

Роль технологій у розв’язанні задач оптимізації

Unreal Engine 5, як один із провідних ігрових рушіїв, надає значний набір інструментів для роботи з графічними об’єктами та їхньої оптимізації:

- Nanite. Ця технологія автоматично оптимізує високополігональні моделі, дозволяючи зберігати їхню якість, але знижувати навантаження на обчислювальні ресурси.

- Lumen. Відповідає за глобальне освітлення та відбиття світла в реальному часі, забезпечуючи якість зображення з меншими витратами ресурсів.

- Технології текстурної компресії та управління LOD. Надають можливість автоматично зменшувати розмір текстур і динамічно змінювати деталізацію об’єктів залежно від відстані до камери.

Ці технології спрямовані на полегшення роботи розробників, але їх інтеграція в проєкт потребує аналізу та налаштування.

Організація даних для оптимізації

Для ефективної роботи з великими обсягами даних, пов’язаних із продуктивністю ігрових об’єктів, використовуються сховища даних і аналітичні технології. У межах лабораторних робіт курсу «Організація сховищ даних» було розроблено схеми баз даних, що включають такі таблиці:

- ObjectDim — характеристика ігрових об’єктів (тип, складність, категорія).

- OptimizationDim — методи оптимізації (LOD, текстурна компресія, culling).

- TimeDim — часові виміри для аналізу змін продуктивності.

- OptimizationFact — фактичні показники оптимізації (зростання FPS, зменшення використання пам'яті).

Ці структури дозволяють накопичувати, аналізувати та візуалізувати дані про ефективність оптимізаційних методів.

Виклики і перспективи

Основними викликами в предметній області є:

- Забезпечення балансу між якістю графіки та продуктивністю.

- Пошук ефективних методів автоматизації оптимізаційних процесів.

- Інтеграція технологій сховищ даних для аналізу продуктивності.

Розв’язання цих викликів шляхом створення комплексної системи оптимізації, що використовує інструменти Unreal Engine 5 та технології обробки даних, дозволить значно підвищити ефективність розробки відеоігор та якість кінцевого продукту.

1.2 Аналіз існуючих рішень та їх проблем

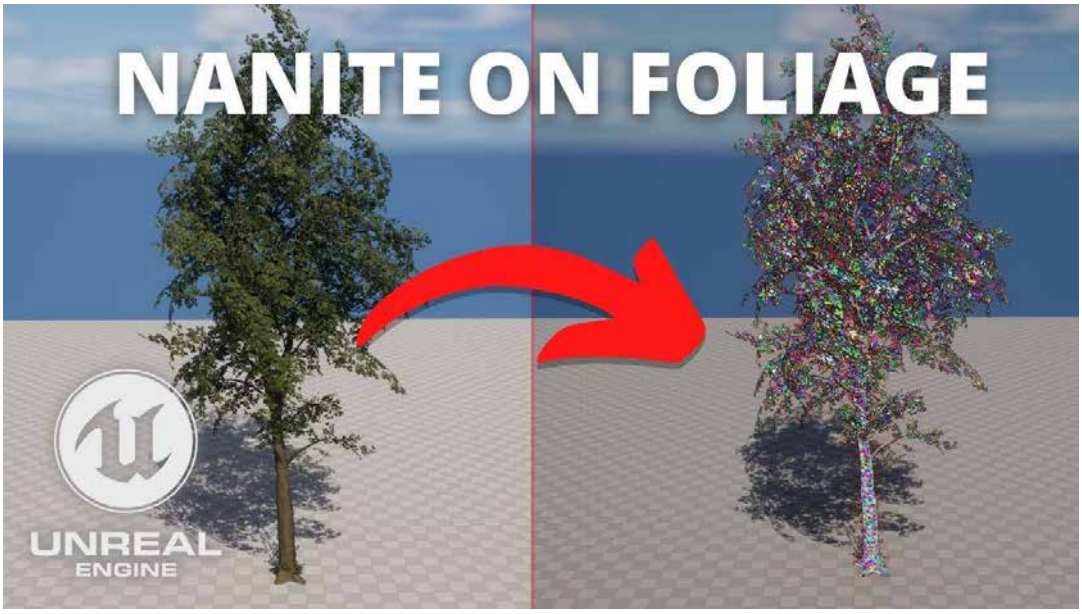
Оптимізація ігрових об'єктів є багатогранною задачею, яка вимагає застосування сучасних технологій і підходів. На сьогодні існує низка рішень, спрямованих на підвищення продуктивності відеоігор, проте жодне з них не є універсальним. Це зумовлено різноманітністю типів ігор, графічних стилів, апаратних платформ та технічних вимог. Аналіз існуючих підходів до

оптимізації ігрових об'єктів дозволяє виявити їхні переваги, недоліки та потенційні напрямки вдосконалення.

Огляд сучасних рішень

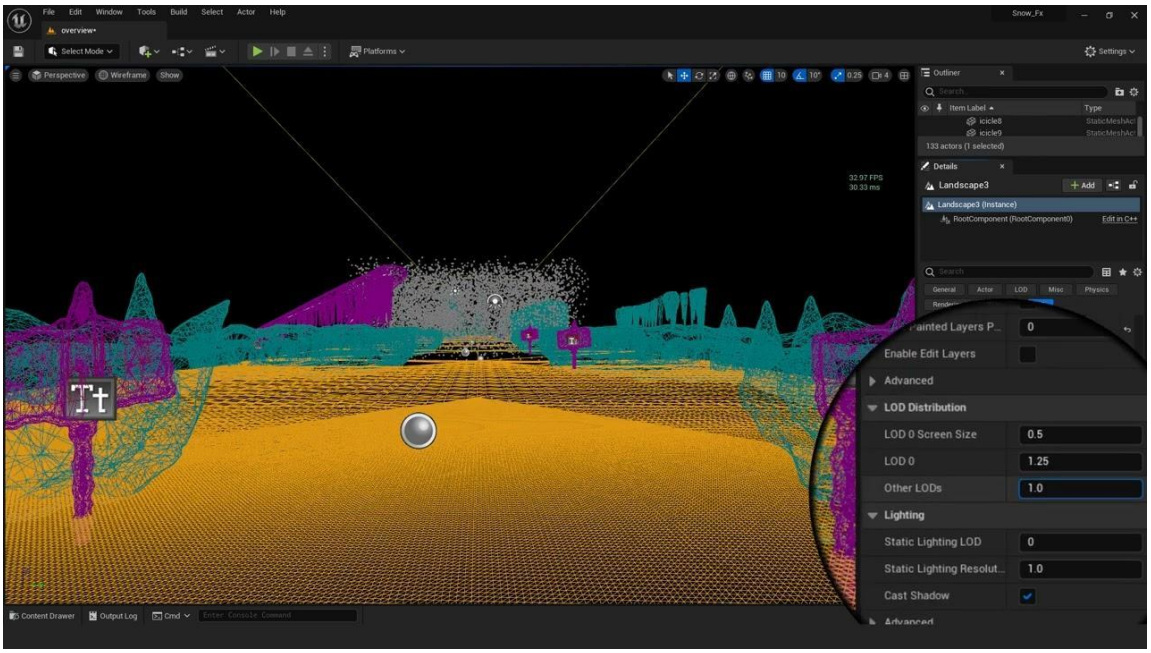
На ринку розробки ігор існує декілька ключових інструментів і технологій, які використовуються для оптимізації ігрових об'єктів:

1. **Технології Unreal Engine 5**:
    - **Nanite** — технологія динамічного управління геометрією, яка дозволяє використовувати високополігональні моделі без необхідності ручного спрощення. Nanite автоматично зменшує кількість полігонів залежно від відстані до камери, забезпечуючи збереження якості при зниженні навантаження на GPU.
    - **Lumen** — система глобального освітлення в реальному часі, що зменшує потребу в попередньо обчислених картах освітлення (lightmaps). Це значно спрощує процес роботи з освітленням, проте вимагає значних ресурсів, особливо на мобільних платформах.
    - **Система LOD (Level of Detail)** — використовується для автоматичного вибору рівня деталізації об'єктів залежно від відстані до камери. Ця технологія дозволяє значно зменшити навантаження на GPU, однак її ефективність залежить від якості налаштувань, які зазвичай виконуються вручну.



«Використання методу оптимізації Nanite на об’єкті дерево»

Порівняння звичайного способу освітлення із Lumen



Використання LOD System

2. **Сторонні інструменти для оптимізації**:
    - **Blender і Autodesk Maya** — широко використовуються для спрощення сіток 3D-моделей (mesh simplification). Ці інструменти дозволяють вручну знижувати кількість полігонів, проте такий підхід є трудомістким і не автоматизованим.
    - **Алгоритми компресії текстур** — програми, такі як Substance Painter, забезпечують створення стислих текстур, що знижує обсяг пам'яті, необхідний для їх зберігання. Однак надмірна компресія може призводити до втрати якості.
3. **Аналітичні системи**:

- **OLAP (Online Analytical Processing)** — технології багатовимірного аналізу даних, які дозволяють аналізувати продуктивність ігрових об'єктів. Наприклад, Microsoft SQL Server Analysis Services (SSAS) забезпечує створення OLAP-кубів, які використовуються для оцінки ключових показників ефективності (KPI).
- **Системи ETL (Extract, Transform, Load)**, такі як SQL Server Integration Services (SSIS), забезпечують інтеграцію даних із різних джерел, їх трансформацію та завантаження в сховища даних.

Проблеми існуючих рішень

Незважаючи на широкий спектр доступних інструментів і підходів, існуючі рішення мають низку проблем, які обмежують їх ефективність:

1. **Відсутність інтеграції**:
   - Більшість інструментів працюють ізольовано, що ускладнює комплексне управління процесом оптимізації. Наприклад, Nanite забезпечує ефективну оптимізацію геометрії, але не враховує інші аспекти, такі як текстурна компресія або управління освітленням.
2. **Недостатня автоматизація**:
   - Багато рішень вимагають значного ручного втручання, що уповільнює процес розробки. Наприклад, налаштування LOD для кожного об'єкта часто виконується вручну, що займає багато часу при роботі з великими ігровими сценами.
3. **Обмеження апаратних ресурсів**:
   - Технології, такі як Lumen, є надзвичайно ресурсоємними і погано оптимізованими для мобільних платформ. Це створює проблеми для розробників, які прагнуть досягти балансу між продуктивністю та якістю.
4. **Відсутність зручних інструментів для аналізу продуктивності**:
   - Більшість існуючих рішень не надають достатніх інструментів для аналізу даних про продуктивність. Наприклад, хоча OLAP-куби дозволяють оцінювати ефективність оптимізації, їх інтеграція в ігровий процес потребує додаткових витрат часу та ресурсів.
5. **Відсутність зворотного зв’язку**:
   - Багато інструментів не надають розробникам інформації про те, як оптимізації впливають на кінцеву продуктивність гри. Це ускладнює прийняття обґрунтованих рішень.

На основі виявлених проблем можна визначити такі напрямки вдосконалення:

1. **Розробка інтегрованих рішень**:
    - Необхідно створити програмне забезпечення, яке об'єднує всі аспекти оптимізації (геометрія, текстури, освітлення) і надає єдину платформу для управління цими процесами.
2. **Автоматизація процесів**:
    - Використання алгоритмів машинного навчання для автоматичного налаштування LOD, вибору рівня компресії текстур і оптимізації освітлення.
3. **Інтеграція аналітичних інструментів**:
    - Розробка інтерфейсів для інтеграції OLAP-технологій із рушієм Unreal Engine 5, що дозволить автоматично збирати та аналізувати дані про продуктивність.
4. **Оптимізація для мобільних платформ**:
    - Розробка специфічних рішень, які враховують обмеження мобільних пристроїв, таких як зменшення кількості обчислень у реальному часі.
5. **Зворотний зв’язок і візуалізація даних**:
    - Створення інструментів, які надають розробникам детальний зворотний зв’язок про ефективність оптимізації через інтуїтивно зрозумілий інтерфейс.

Аналіз існуючих рішень показав, що сучасні технології оптимізації ігрових об'єктів забезпечують високий рівень автоматизації окремих процесів, але мають обмеження в комплексному підході. Розробка інтегрованої системи, яка об’єднує інструменти оптимізації та аналітичного аналізу, дозволить усунути ці недоліки і забезпечити більш ефективний процес розробки відеоігор.

1.3 Постановка завдання для магістерської роботи

У рамках магістерської роботи ставиться завдання дослідити програмне забезпечення для оптимізації ігрових об'єктів у середовищі Unreal Engine 5. Незважаючи на існування вбудованих інструментів і технологій, таких як Nanite, Lumen і LOD-системи, питання ефективності їх використання, особливостей інтеграції в різних проєктах та їхніх обмежень залишається

актуальним. Метою дослідження є аналіз цих інструментів та методів, оцінка їх впливу на продуктивність і візуальну якість, а також розробка рекомендацій щодо їх оптимального використання.

Метою магістерської роботи є дослідження існуючих програмних рішень для оптимізації ігрових об'єктів у середовищі Unreal Engine 5 з акцентом на аналіз їхніх можливостей, обмежень і ефективності. Робота спрямована на формування теоретичних і практичних знань, які допоможуть розробникам ефективніше використовувати наявні інструменти для вирішення задач оптимізації. Для досягнення поставленої мети необхідно виконати такі завдання:

1. **Огляд існуючих технологій оптимізації**:
    - Провести аналіз основних інструментів Unreal Engine 5, включаючи Nanite, Lumen, систему LOD та інші методи.
    - Вивчити документацію та практичні приклади застосування цих технологій.
2. **Дослідження впливу технологій оптимізації на продуктивність**:
    - Оцінити вплив інструментів оптимізації на показники продуктивності, такі як FPS, використання пам'яті та час завантаження.
    - Провести експериментальне тестування на різних апаратних платформах.
3. **Аналіз обмежень та викликів**:
    - Дослідити обмеження використання існуючих інструментів, зокрема на мобільних платформах та пристроях зі слабким апаратним забезпеченням.
    - Визначити типові проблеми, з якими стикаються розробники під час застосування цих технологій.
4. **Розробка рекомендацій**:
    - Сформувати рекомендації для розробників ігрових проєктів щодо ефективного використання існуючих інструментів оптимізації.
    - Розробити шаблони процесів оптимізації для проєктів різного масштабу і типу (AAA-проєкти, інді-ігри, мобільні ігри).
5. **Візуалізація та інтерпретація результатів**:
    - Підготувати звіти та графічні матеріали, що демонструють результати тестування та аналізу.

- Представити ключові метрики та показники у вигляді діаграм і таблиць для зручного порівняння.

Дослідження оптимізації ігрових об'єктів у Unreal Engine 5 є актуальним через постійне підвищення вимог до якості графіки та продуктивності ігор. Використання вбудованих інструментів рушія значно спрощує процес розробки, але їхнє ефективне застосування залежить від рівня знань розробників про їхні можливості та обмеження. Вивчення цих аспектів сприятиме підвищенню ефективності роботи розробників та якості кінцевого продукту.

Очікуваними результатами магістерської роботи є:

1. Систематизований огляд і порівняльний аналіз інструментів оптимізації ігрових об'єктів у Unreal Engine 5.
2. Експериментальні дані про вплив використання цих інструментів на продуктивність.
3. Практичні рекомендації щодо оптимального використання технологій оптимізації.
4. Узагальнені висновки про перспективи розвитку інструментів оптимізації у середовищі Unreal Engine 5.

МОДЕЛЮВАННЯ СИСТЕМИ

Процес моделювання є невід'ємною складовою розробки та аналізу складних програмних систем. У контексті даної магістерської роботи моделювання системи дозволяє структурувати інформацію про існуючі інструменти оптимізації ігрових об'єктів в Unreal Engine 5, визначити ключові компоненти, взаємозв’язки між ними та алгоритми роботи.

Моделювання дає змогу створити формалізовану основу для аналізу функціонування досліджуваного програмного забезпечення, виявлення його сильних і слабких сторін, а також побудови рекомендацій для ефективного використання інструментів оптимізації. Використання діаграм і моделей сприяє не лише розумінню внутрішніх процесів системи, але й допомагає візуалізувати складні технічні аспекти, що є важливим для комунікації результатів дослідження.

Основна мета цього розділу — представити концептуальну та логічну модель досліджуваної системи, яка демонструє основні компоненти, процеси та їхню взаємодію. Завдяки цьому буде визначено ключові елементи системи оптимізації ігрових об'єктів та способи їх застосування в різних умовах.

Завдання розділу включають:

1. Визначення структури даних, які використовуються інструментами оптимізації в Unreal Engine 5.
2. Побудова діаграм, що відображають процеси оптимізації та обробки даних.
3. Аналіз компонентів програмного забезпечення та їх функціонального призначення.
4. Формалізація підходів до аналізу та оцінки ефективності оптимізації.

2.1 Опис та аналіз методологій системного аналізу

Системний аналіз є ключовим етапом при дослідженні складних програмних систем. Він дозволяє формалізувати основні процеси, структуру даних, взаємозв’язки між компонентами системи та алгоритми їх взаємодії. У контексті цієї магістерської роботи системний аналіз спрямований на вивчення існуючих методологій, які можуть бути застосовані для дослідження інструментів оптимізації ігрових об'єктів в Unreal Engine 5.

Основні методології системного аналізу. У сучасній практиці використовуються різні методології системного аналізу, кожна з яких має свої переваги та недоліки. У рамках цієї роботи розглядаються найпоширеніші підходи, які можуть бути застосовані для дослідження програмного забезпечення.

1. **Класичний системний аналіз**:
    - Цей підхід базується на глибокому дослідженні системи, її структури та функцій. Основні етапи включають:
        - Постановку задачі.
        - Збір і аналіз вимог.
        - Визначення функціональних і нефункціональних характеристик системи.
        - Моделювання процесів і даних.
    - Переваги:
        - Детальний опис компонентів і процесів.
        - Застосовність для складних систем із великою кількістю взаємозалежних елементів.
    - Недоліки:
        - Часомісткість і значна залежність від обсягу вхідних даних.
2. **Об’єктно-орієнтований аналіз (ООА)**:
    - Цей підхід спрямований на виявлення об’єктів, їх властивостей, методів і взаємозв’язків. Використовуються UML-діаграми, такі як:
        - Діаграми класів (Class Diagram) для опису об’єктів.
        - Діаграми послідовності (Sequence Diagram) для відображення взаємодії об’єктів.
    - Переваги:
        - Зручність у моделюванні складних систем з високим ступенем деталізації.
        - Можливість багаторазового використання створених моделей.
    - Недоліки:
        - Може бути складним для реалізації у системах із нечітко визначеними межами.

3. **Методологія структурного аналізу та проектування (SAD)**:
    - Цей підхід включає побудову діаграм потоків даних (Data Flow Diagrams, DFD) для визначення передачі даних між процесами і компонентами.
    - Основними етапами є:
        - Аналіз функцій системи.
        - Визначення потоків даних і їх джерел.
        - Створення контекстних діаграм для визначення меж системи.
    - Переваги:
        - Простота у візуалізації даних і процесів.
        - Зручність для аналізу великих систем із багатьма залежностями.
    - Недоліки:
        - Може не враховувати об’єктно-орієнтовані аспекти.
4. **Методології Agile**:
    - У рамках системного аналізу застосовуються такі підходи, як ітеративний збір вимог і побудова моделей. Особливістю є гнучкість і адаптивність до змін у процесі роботи.
    - Переваги:
        - Швидка адаптація до нових вимог.
        - Фокус на результат і зручність для розробників.
    - Недоліки:
        - Не завжди підходить для великих проєктів із чітко визначеними цілями.

Застосування методологій до дослідження інструментів Unreal Engine 5

У рамках цієї роботи було обрано комбінацію методологій, які найкраще відповідають специфіці досліджуваного програмного забезпечення:

1. **Аналіз із використанням UML-діаграм**:
    - Діаграми послідовності (Sequence Diagram) будуть побудовані для демонстрації взаємодії між інструментами оптимізації та аналітичними модулями.

- Діаграми прецедентів (Use Case Diagram) покажуть сценарії використання інструментів оптимізації розробниками.

Для дослідження програмного забезпечення оптимізації ігрових об’єктів в Unreal Engine 5 найбільш доцільним є використання об’єктно-орієнтованого аналізу та UML-діаграм, оскільки ці методи дозволяють чітко візуалізувати структуру та взаємодію компонентів системи. Комбінування з методологією структурного аналізу дає можливість дослідити потоки даних і зв’язки між компонентами.

Аналіз методологій системного аналізу показав, що для дослідження програмного забезпечення оптимізації ігрових об'єктів необхідно використовувати поєднання об’єктно-орієнтованих підходів, UML-діаграм і структурного аналізу. Це дозволить забезпечити повний і всебічний аналіз інструментів Unreal Engine 5, враховуючи їхні функції, потоки даних та сценарії використання. Отримані моделі стануть основою для подальшого аналізу та узагальнення результатів дослідження.

2.2 Визначення UML

Системний аналіз є ключовим етапом при дослідженні складних програмних систем. Він дозволяє формалізувати основні процеси, структуру даних, взаємозв’язки між компонентами системи та алгоритми їх взаємодії. У контексті цієї магістерської роботи системний аналіз спрямований на вивчення існуючих методологій, які можуть бути застосовані для дослідження інструментів оптимізації ігрових об'єктів в Unreal Engine 5.

У цьому підрозділі розглядаються загальні принципи та методи моделювання, що є актуальними для вивчення програмного забезпечення, а також їх значення у рамках магістерської роботи.

Значення моделювання в системному аналізі

Моделювання виступає як інструмент формалізації системних процесів і структур. Його основними цілями є:

1. **Розуміння системи**:
    - Моделі дозволяють структурувати інформацію про систему, зробити її більш зрозумілою для аналізу.
    - Наприклад, у контексті Unreal Engine 5 моделювання допомагає зрозуміти, як працюють технології Nanite, Lumen і LOD.
2. **Виявлення проблем**:

- Завдяки моделюванню можна знайти "вузькі місця" у роботі системи, такі як неефективні потоки даних чи нераціональне використання ресурсів.

3. **Комунікація**:
   - Візуальні моделі полегшують комунікацію між учасниками проєкту, оскільки вони наочно представляють складні технічні аспекти.
4. **Верифікація та тестування**:
   - Моделювання дозволяє перевірити коректність роботи системи ще до її впровадження чи дослідження.

Основні види моделей

Моделі можна класифікувати за їхнім призначенням і типом представлення. У рамках цієї магістерської роботи будуть використані такі види моделей:

1. **Функціональні моделі**:
   - Моделі, які описують функції системи та їхню взаємодію. Прикладом є діаграми потоків даних (Data Flow Diagram, DFD), які показують, як дані переміщуються між компонентами.
2. **Структурні моделі**:
   - Вони зосереджуються на описі компонентів системи та їхніх зв’язків. До таких моделей належать діаграми класів (Class Diagram), які показують властивості та методи об'єктів.
3. **Динамічні моделі**:
   - Ці моделі описують поведінку системи в часі, наприклад, діаграми послідовності (Sequence Diagram), які демонструють, як об'єкти взаємодіють під час виконання процесів.
4. **Об’єктно-орієнтовані моделі**:
   - Використовуються для моделювання складних систем, де об’єкти взаємодіють між собою. UML (Unified Modeling Language) є стандартом для створення таких моделей.

Підходи до моделювання

Моделювання може бути реалізоване за допомогою різних підходів, які вибираються залежно від цілей дослідження:

1. **Теоретичне моделювання**:

- Ґрунтується на аналізі теоретичних аспектів роботи системи. Наприклад, вивчення алгоритмів, які реалізують оптимізацію в Nanite чи Lumen.

2. **Емпіричне моделювання**:
   - Використовується для моделювання на основі зібраних даних. У цій роботі воно застосовується для аналізу даних про продуктивність системи (FPS, обсяг пам’яті).
3. **Комбіноване моделювання**:
   - Поєднує теоретичний і емпіричний підходи, що дозволяє створювати більш точні моделі системи.

Використання UML для моделювання

Unified Modeling Language (UML) є одним із найпоширеніших стандартів для моделювання програмних систем. Він пропонує широкий спектр діаграм для формалізації різних аспектів системи:

1. **Діаграми класів (Class Diagram)**:
   - Використовуються для опису основних компонентів, які залучені до процесу оптимізації, наприклад, класи для роботи з Nanite чи LOD.
2. **Діаграми послідовності (Sequence Diagram)**:
   - Відображають порядок взаємодії між компонентами під час виконання завдань оптимізації.
3. **Діаграми прецедентів (Use Case Diagram)**:
   - Дозволяють визначити сценарії використання програмного забезпечення, наприклад, оптимізацію геометрії чи текстур.
4. **Діаграми компонентів (Component Diagram)**:
   - Описують структуру програмного забезпечення, включаючи взаємозв’язки між різними модулями.

У рамках цієї магістерської роботи моделювання дозволить:

1. Виявити всі компоненти програмного забезпечення, залучені до оптимізації ігрових об'єктів.
2. Візуалізувати процеси, пов’язані з використанням інструментів Nanite, Lumen та інших систем Unreal Engine 5.
3. Підготувати основу для аналізу ефективності існуючих рішень і формулювання рекомендацій.

Моделювання є важливим етапом для дослідження складних систем, таких як програмне забезпечення для оптимізації ігрових об'єктів. Використання структурних, функціональних та динамічних моделей, зокрема UML-діаграм, дозволяє формалізувати та глибше зрозуміти роботу інструментів Unreal Engine 5, що стане основою для подальшого аналізу та розробки рекомендацій у цій магістерській роботі.

2.2.1 Визначення UML

UML (Unified Modeling Language, уніфікована мова моделювання) — це стандарт мови графічного моделювання, який широко використовується для формалізації, візуалізації та документування архітектури програмного забезпечення. UML був створений для об’єднання різних підходів до моделювання програмних систем і став одним із найпоширеніших інструментів системного аналізу.

UML надає можливість створювати моделі, які описують структуру, поведінку та взаємодію компонентів програмного забезпечення. Це особливо актуально у контексті дослідження інструментів оптимізації ігрових об’єктів в Unreal Engine 5, де UML дозволяє формалізувати процеси та оцінити ефективність існуючих рішень.

Основні аспекти UML

UML охоплює три основні типи моделей:

1. **Структурні моделі** — відображають статичну структуру системи, такі як класи, компоненти, об’єкти та їхні зв’язки.
2. **Динамічні моделі** — описують поведінку системи в часі, включаючи послідовності взаємодії між елементами.
3. **Моделі реалізації** — демонструють фізичну структуру системи, наприклад, розгортання компонентів.

Діаграми UML, які використовуються в роботі

На основі лабораторних досліджень та їхньої описової частини, було виділено ключові діаграми UML, які застосовуються для моделювання програмного забезпечення:

1. **Діаграма класів (Class Diagram)**:
    - Відображає структуру об’єктів системи, їхні властивості, методи та зв’язки між ними.
    - У контексті Unreal Engine 5 ця діаграма використовується для формалізації таких елементів, як інструменти оптимізації (Nanite,

Lumen), їхні налаштування та зв’язки із зовнішніми системами (сховищами даних).

2. **Діаграма послідовності (Sequence Diagram)**:
    - Демонструє порядок виконання дій та взаємодію об’єктів під час виконання конкретного сценарію.
    - Наприклад, вона може показувати процес взаємодії між LOD-системою і Nanite під час оптимізації геометрії.
3. **Діаграма потоків даних (Data Flow Diagram, DFD)**:
    - Відображає передачу даних між компонентами системи та їхню обробку.
    - Ця діаграма допомагає зрозуміти, як аналітичні дані (наприклад, FPS, використання пам’яті) передаються від Unreal Engine до сховищ даних для аналізу.
4. **Діаграма прецедентів (Use Case Diagram)**:
    - Визначає основні сценарії взаємодії користувача (розробника) з інструментами програмного забезпечення.
    - Наприклад, такі сценарії можуть включати налаштування параметрів оптимізації або перегляд результатів аналізу продуктивності.

Принципи використання UML у роботі

UML діаграми, створені у рамках лабораторних досліджень, забезпечують:

- **Формалізацію процесів оптимізації**: UML дозволяє структурувати інформацію про використання Nanite, Lumen та інших інструментів.
- **Візуалізацію потоків даних**: Визначення зв’язків між компонентами допомагає зрозуміти, як інформація рухається в системі.
- **Аналіз залежностей**: UML діаграми дозволяють виявити можливі "вузькі місця" у взаємодії компонентів.

Визначення UML як універсального інструменту для моделювання систем дозволяє створювати точні та зрозумілі моделі, які стають основою для аналізу та вдосконалення системи. У контексті роботи UML діаграми надають можливість детально дослідити структуру та процеси програмного забезпечення для оптимізації ігрових об'єктів в Unreal Engine 5, що сприятиме формулюванню рекомендацій для його ефективного використання.

2.2.2 Об'єктно-орієнтоване проєктування

Об’єктно-орієнтоване проєктування (ООП) є невід'ємною частиною сучасного розроблення програмного забезпечення. Воно спрямоване на моделювання системи як набору об’єктів, які взаємодіють один з одним. У контексті магістерської роботи, яка присвячена аналізу програмного забезпечення для оптимізації ігрових об'єктів в Unreal Engine 5, об’єктно-орієнтований підхід дозволяє формалізувати структуру та поведінку компонентів системи, таких як Nanite, Lumen, LOD-системи та інші.

ООП базується на кількох фундаментальних принципах:

1. **Інкапсуляція**:
    - Зосереджує властивості (атрибути) та поведінку (методи) в межах одного об'єкта, що забезпечує захист даних від зовнішнього втручання.
    - Наприклад, об’єкт "GameObject" в Unreal Engine 5 може мати атрибути, такі як кількість полігонів, розмір текстури, та методи, що відповідають за їх оптимізацію.
2. **Наслідування**:
    - Дозволяє створювати нові класи на основі існуючих, успадковуючи їхні атрибути та методи.
    - У системах оптимізації Unreal Engine це дозволяє створювати підкласи, наприклад, "StaticMeshObject" для статичних об'єктів і "DynamicMeshObject" для динамічних.
3. **Поліморфізм**:
    - Забезпечує можливість використовувати один і той самий інтерфейс для об'єктів різних типів.
    - Наприклад, метод "Optimize()" може бути реалізований по-різному для текстур, моделей або освітлення.
4. **Абстракція**:
    - Виділяє ключові характеристики об'єктів і приховує другорядні деталі.
    - У випадку оптимізації, це може бути представлено як набір базових інтерфейсів для різних видів оптимізацій (LOD, culling, текстурна компресія).

Використання UML для об’єктно-орієнтованого проєктування

В рамках лабораторних робіт з організації сховищ даних було створено UML-діаграми, які відповідають принципам об’єктно-орієнтованого проєктування. Основні компоненти цих діаграм:

1. **Діаграма класів (Class Diagram)**:
    - Дозволяє описати об’єкти системи та їхні взаємозв’язки.
    - Наприклад, об'єкти "GameObject", "OptimizationMethod", "FileAsset" описуються такими атрибутами, як ComplexityLevel, OptimizationDate, MemoryImprovement.
2. **Діаграма об’єктів (Object Diagram)**:
    - Відображає конкретні екземпляри класів на певний момент часу.
    - У системі оптимізації це можуть бути конкретні ігрові об’єкти (персонажі, текстури), до яких застосовано методи оптимізації.
3. **Діаграма послідовності (Sequence Diagram)**:
    - Відображає порядок взаємодії між об’єктами.
    - Наприклад, порядок викликів методів оптимізації для покращення FPS і зменшення пам'яті.
4. **Діаграма компонентів (Component Diagram)**:
    - Описує фізичну реалізацію компонентів системи, таких як сховище даних, аналітичні модулі (OLAP-куби) та інструменти Unreal Engine.

Реалізація ООП у контексті магістерської роботи

На основі лабораторних робіт реалізовані такі концепти:

1. **Моделювання ігрових об'єктів**:
    - Об'єкти "GameObject" мають атрибути ComplexityLevel, TextureFormat, OptimizationMethods, а також методи, що відповідають за оптимізацію (наприклад, ApplyLOD, CompressTexture).
2. **Ієрархія класів оптимізації**:
    - Створено базовий клас "OptimizationMethod" із підкласами для різних методів оптимізації (LOD, culling, mesh simplification).
3. **Інтеграція зі сховищами даних**:
    - Дані про оптимізації (FPS до/після, використання пам'яті) зберігаються у сховищі даних, яке реалізовано через таблиці OptimizationFact, ObjectDim, TimeDim.

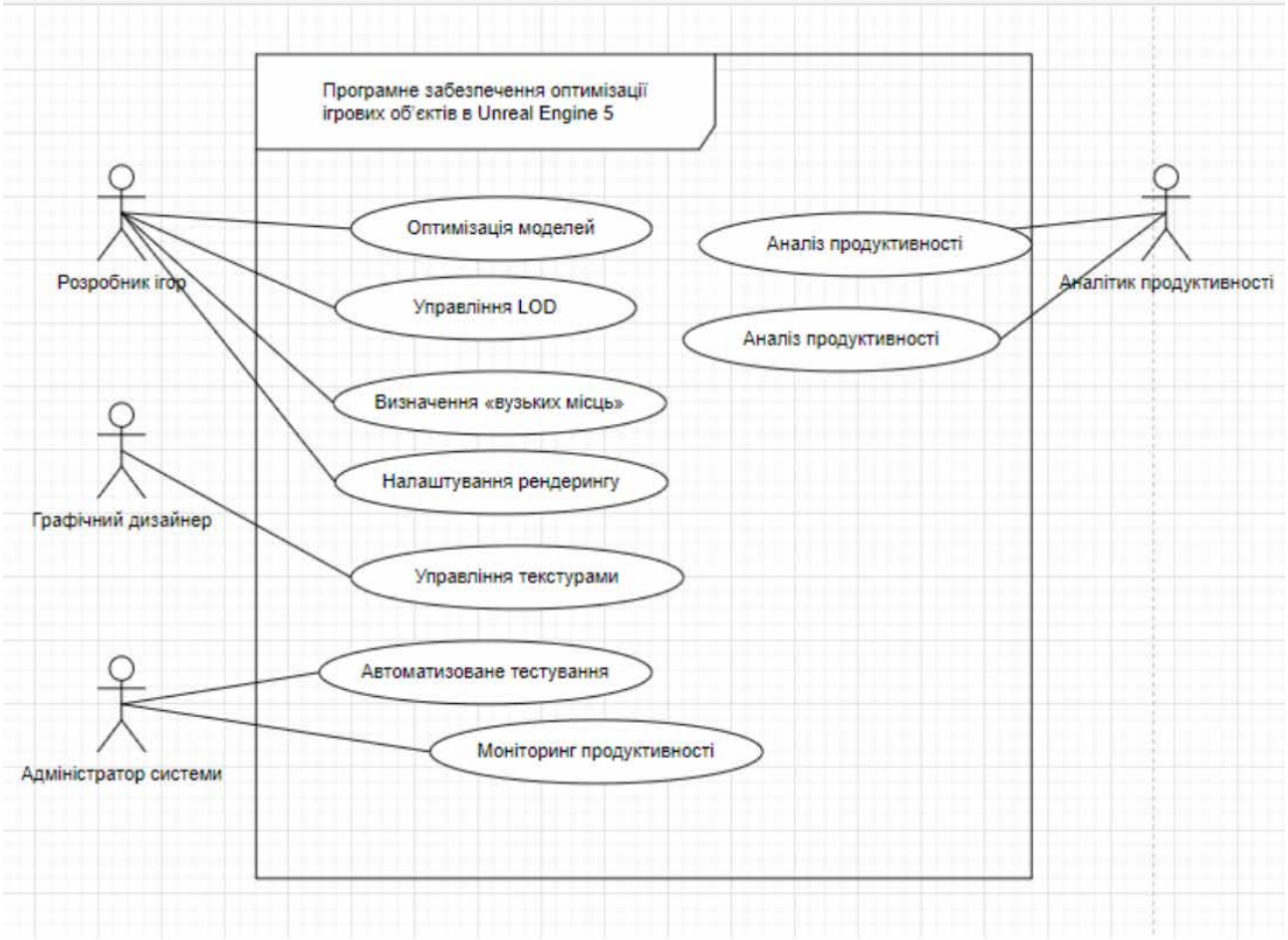
2.3 Діаграма прецедентів

Діаграма прецедентів (Use Case Diagram) є ключовим інструментом у системному аналізі, який дозволяє описати взаємодію користувачів (акторів) із системою через визначення сценаріїв використання. У рамках магістерської роботи, присвяченої дослідженню програмного забезпечення для оптимізації ігрових об'єктів в Unreal Engine 5, діаграма прецедентів забезпечує формалізацію основних процесів, що відбуваються між розробниками, інструментами оптимізації та іншими компонентами системи.

Основні поняття

1. **Прецедент** — конкретний сценарій взаємодії користувача із системою, який відповідає певній функції. Наприклад, "Оптимізація текстур" або "Управління рівнями деталізації".
2. **Актори** — суб'єкти, які взаємодіють із системою. У даному випадку це можуть бути:
    - **Розробник ігор (Game Developer)**: використовує інструменти оптимізації для налаштування параметрів і підвищення продуктивності.
    - **Адміністратор системи (System Administrator)**: налаштовує інфраструктуру, забезпечує моніторинг продуктивності.
    - **Графічний дизайнер (Graphic Designer)**: оптимізує текстури для зменшення навантаження на систему.
    - **Аналітик продуктивності (Performance Analyst)**: оцінює вплив оптимізацій на продуктивність.

Діаграма зображена нижче:



Діаграма прецедентів

Діаграма прецедентів для досліджуваного програмного забезпечення включає такі основні взаємодії:

1. **Розробник ігор (Game Developer)**:
    - **Оптимізація моделей**: виконання технічної оптимізації 3D-моделей (спрощення сітки, налаштування LOD).
    - **Управління LOD**: налаштування рівнів деталізації для забезпечення балансу між продуктивністю та якістю.
    - **Визначення "вузьких місць"**: аналіз ігрових сцен для виявлення компонентів, що потребують оптимізації.
    - **Налаштування рендерингу**: оптимізація параметрів рендерингу.
2. **Графічний дизайнер (Graphic Designer)**:
    - **Управління текстурами**: створення текстур із високим рівнем стиснення, що зменшує використання пам'яті без значних втрат у якості.
3. **Адміністратор системи (System Administrator)**:
    - **Автоматизоване тестування**: налаштування системи для автоматизації перевірок продуктивності.

- **Моніторинг продуктивності**: забезпечення стабільної роботи інструментів та відстеження основних метрик, таких як FPS.

4. **Аналітик продуктивності (Performance Analyst)**:
    - **Аналіз продуктивності**: оцінка впливу оптимізацій на продуктивність ігрових сцен.
    - **Звітування**: створення звітів і графіків для аналізу ефективності.

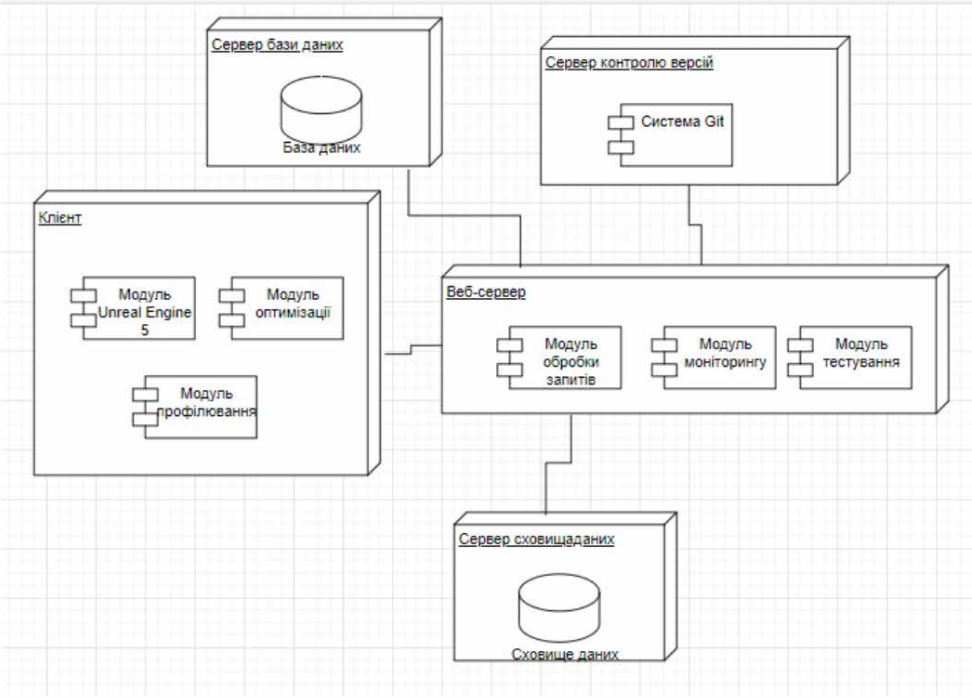
Опис основних сценаріїв

1. **Оптимізація моделей**:
    - Розробник ігор використовує функціонал Nanite для автоматичного спрощення моделей. Система проводить аналіз і застосовує оптимізацію, зберігаючи вихідні та оптимізовані версії.
2. **Управління LOD**:
    - Користувач налаштовує рівні деталізації через панель Unreal Engine 5. Залежно від відстані до камери об'єкт динамічно змінює рівень полігонів.
3. **Аналіз продуктивності**:
    - Аналітик продуктивності за допомогою Unreal Insights аналізує FPS до і після оптимізації, створює графіки для оцінки ефективності.

Діаграма прецедентів є важливим інструментом для формалізації взаємодії користувачів із програмним забезпеченням. У контексті магістерської роботи вона допомагає структурувати основні сценарії використання інструментів оптимізації в Unreal Engine 5, визначити ролі користувачів та зв'язки між компонентами системи. Це створює основу для подальшого моделювання та аналізу роботи системи.

2.4 Архітектура програмного забезпечення

Топологія системи є важливим аспектом у моделюванні та аналізі програмного забезпечення. Вона визначає структуру компонентів, їхнє розташування, зв’язки між ними та взаємодію. У контексті дослідження програмного забезпечення для оптимізації ігрових об'єктів в Unreal Engine 5, топологія системи описує ключові елементи, такі як сховище даних, аналітичні інструменти, інструменти оптимізації, та їхній взаємозв’язок. Це дозволяє зрозуміти, як різні частини системи співпрацюють для досягнення високої продуктивності ігрових об'єктів.



Основні компоненти топології

Система складається з таких основних вузлів:

1. **Клієнтська сторона (Розробник ігор)**:
    - **Модуль Unreal Engine 5**: забезпечує основне середовище для розробки ігрових сцен.
    - **Модуль оптимізації**: містить інструменти, такі як Nanite, Lumen, LOD, для оптимізації моделей, текстур і рендерингу.
    - **Модуль профілювання**: наприклад, Unreal Insights, використовується для аналізу продуктивності, зокрема FPS і використання пам’яті.
2. **Серверна інфраструктура**:
    - **Сервер контролю версій**:
        - Система Git, яка дозволяє відстежувати зміни в ігрових об'єктах та керувати їхніми версіями.
    - **Сервер моніторингу**:
        - Відповідає за збір даних про продуктивність системи, таких як FPS, час рендерингу, використання пам’яті.
3. **Сховище даних**:
    - **База даних сховища**: містить інформацію про ігрові об’єкти, методи оптимізації, результати продуктивності до і після оптимізації.

- **OLAP-куби**: використовуються для багатовимірного аналізу даних, зокрема вимірів FPSBefore, FPSAfter, MemoryUsageBefore, MemoryUsageAfter.

4. **Інструменти аналітики**:
   - **SQL Server Analysis Services (SSAS)**: забезпечує створення аналітичних звітів і OLAP-кубів для аналізу ефективності оптимізацій.
   - **SQL Server Integration Services (SSIS)**: використовується для інтеграції даних із джерел у сховище.

1. **Вузол: Клієнт (Розробник ігор)**:
   - Основне середовище розробки представлено Unreal Engine 5.
   - Інструменти Nanite та Lumen забезпечують оптимізацію 3D-моделей та освітлення.
   - Unreal Insights здійснює збір метрик продуктивності.
2. **Вузол: Сервер контролю версій**:
   - Використання Git дозволяє відстежувати версії файлів, змінювати параметри оптимізації та синхронізувати їх між членами команди.
3. **Вузол: Сховище даних**:
   - Структура бази даних включає таблиці:
     - ObjectDim — інформація про ігрові об'єкти.
     - OptimizationDim — методи оптимізації.
     - OptimizationFact — результати оптимізацій, такі як покращення FPS та зменшення використання пам’яті.
   - Дані заповнюються через процеси ETL у середовищі SSIS.
4. **Вузол: Сервер аналітики**:
   - SSAS використовується для створення OLAP-кубів, які дозволяють аналізувати ефективність оптимізацій у багатовимірному контексті.
   - Аналітичні звіти формуються через SSRS для оцінки досягнення ключових показників ефективності (KPI).

Зв’язки між компонентами

1. **`Клієнт ↔ Сховище даних`**:

- Клієнтська сторона через процеси ETL (SSIS) передає дані про оптимізацію до сховища. Це включає такі параметри, як рівень деталізації, FPS до/після, обсяг пам’яті.

2. **Сховище даних ↔ Сервер аналітики**:
    - Сховище даних надає вхідні дані для побудови OLAP-кубів. Куби аналізують інформацію за часом, типом об’єкта та методом оптимізації.
3. **Сервер аналітики ↔ Клієнт**:
    - Результати аналізу передаються назад до клієнта у вигляді звітів. Ці звіти дозволяють розробникам оцінити ефективність виконаних оптимізацій.

Візуалізація топології

Топологія системи представлена у вигляді моделі, що включає такі основні компоненти:

- Клієнтське середовище Unreal Engine 5 із підключенням до сховища даних.
- Сервер моніторингу для збору аналітичних даних.
- Сховище даних, яке забезпечує зберігання й аналіз інформації.
- Сервер аналітики, що створює OLAP-куби та звіти.

Топологія системи визначає структуру компонентів, їхню взаємодію та потоки даних між ними. У контексті дослідження програмного забезпечення для оптимізації ігрових об'єктів у Unreal Engine 5, така топологія дозволяє забезпечити ефективну взаємодію між інструментами оптимізації, сховищами даних та інструментами аналітики. Це створює основу для подальшого аналізу ефективності оптимізацій і формування рекомендацій для їхнього вдосконалення.

РОЗРОБКА СИСТЕМИ

Розробка системи є ключовим етапом, який об'єднує результати системного аналізу та моделювання для створення повноцінного функціонального рішення. У рамках магістерської роботи цей розділ зосереджується на деталізації структури джерел даних, процесів оптимізації ігрових об’єктів, механізмів обробки інформації, а також аналізі інструментарію, що застосовується для реалізації поставлених завдань.

Особливу увагу приділено вивченню сучасних технологій Unreal Engine 5, які використовуються для оптимізації 3D-об'єктів і рендерингу. Це включає в себе такі інструменти, як Nanite, Lumen, Level of Detail (LOD), а також механізми текстурної компресії й управління потоками даних. Крім того, розділ охоплює процеси інтеграції даних зі сховищами та аналізу їхнього впливу на продуктивність.

Цей розділ спрямований на:

1. Опис структури джерел інформації, що використовуються для проведення оптимізації, включаючи оперативні бази даних і сховища даних.
2. Розгляд загальних концепцій і принципів оптимізації у середовищі Unreal Engine 5.
3. Аналіз інструментарію, доступного для реалізації завдань оптимізації.
4. Формування бази для оцінки ефективності оптимізацій на основі аналітичних даних.

У цьому розділі розглядаються такі підрозділи:

1. **Структура джерела інформації для проведення оптимізації ігрових об'єктів**:
    - Включає опис оперативної бази даних, сховища даних та механізмів вилучення й обробки інформації.
2. **Загальні поняття технології оптимізації в Unreal Engine 5**:
    - Розкриваються особливості основних технологій, таких як Nanite, Lumen, LOD, і текстурна компресія.
3. **Огляд інструментарію для реалізації завдань оптимізації**:
    - Аналізуються програмні засоби та модулі, що використовуються для оптимізації ігрових об’єктів.

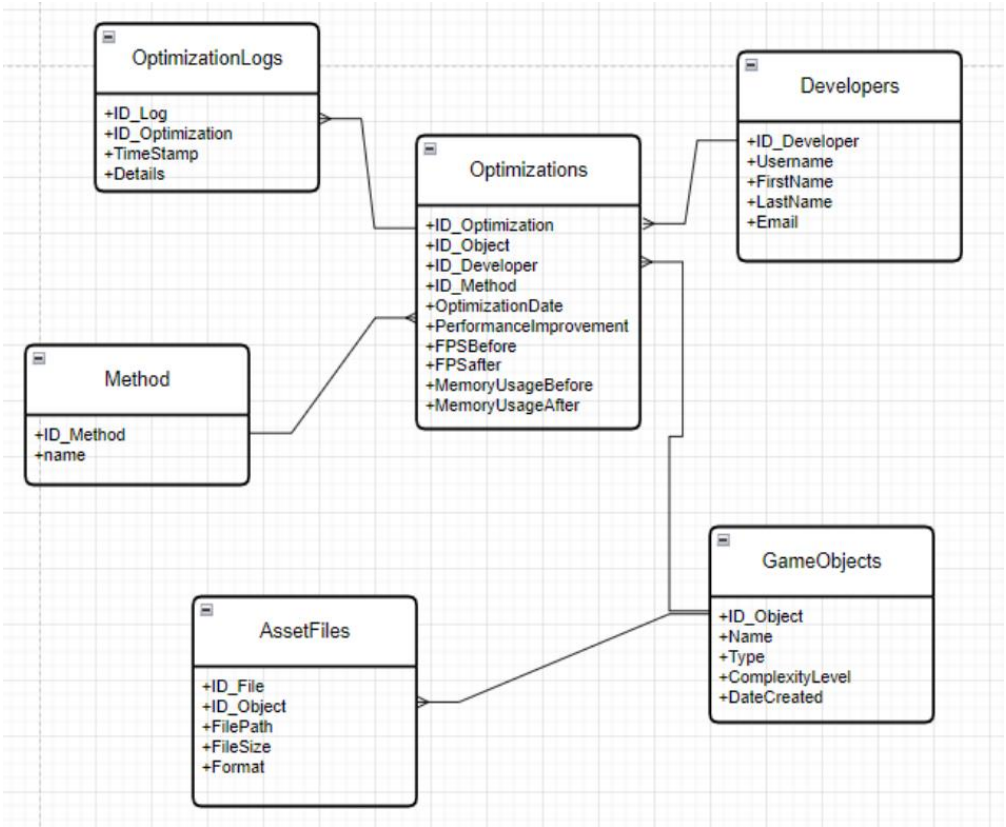
4. **Дані для аналізу**:
    - Наводиться структура та характеристика даних, які використовуються для оцінки результатів оптимізації, таких як FPS, пам’ять і продуктивність системи.

Розділ "Розробка системи" є важливим для формування цілісного уявлення про побудову системи, її структуру та компоненти. Він надає деталізований опис технологій і методів, які використовуються для оптимізації ігрових об'єктів, а також інструментів для їхньої реалізації. Результати, представлені у цьому розділі, закладають основу для проведення аналітичних досліджень, оцінки ефективності оптимізації та формування рекомендацій для розробників ігор.

3.1 Структура бази даних для реалізації OLAP-аналізу

Джерела інформації для проведення оптимізації ігрових об'єктів є важливими складовими системи, оскільки вони забезпечують дані, необхідні для аналізу, оцінки та застосування методів оптимізації. У цьому підрозділі описується структура оперативної бази даних, сховища даних та механізми вилучення, обробки і передачі інформації. Визначення та структурування джерел даних є основою для реалізації оптимізаційних процесів і їхньої інтеграції у середовище Unreal Engine 5.

3.1.1 Структура оперативної бази даних



Оперативна база даних слугує первинним джерелом інформації про ігрові об’єкти, їхні характеристики, методи оптимізації та результати. На основі лабораторних робіт та досліджень, структура бази даних включає такі основні таблиці:

1. **ObjectDim** — таблиця вимірів ігрових об'єктів:
    - **ObjectID**: унікальний ідентифікатор об'єкта.
    - **Name**: назва об'єкта.
    - **Type**: тип об'єкта (наприклад, персонаж, текстура, будівля).
    - **ComplexityLevel**: рівень складності об'єкта.
2. **OptimizationDim** — таблиця методів оптимізації:
    - **OptimizationID**: унікальний ідентифікатор методу оптимізації.
    - **Method**: назва методу оптимізації (LOD, Nanite, текстурна компресія тощо).
3. **TimeDim** — таблиця вимірів часу:
    - **TimeID**: унікальний ідентифікатор часу.
    - **Year, Month, Day, Hour**: часові параметри.
4. **OptimizationFact** — таблиця фактів оптимізації:
    - **TimeID, ObjectID, OptimizationID**: посилання на відповідні таблиці вимірів.
    - **FPSImprovement**: покращення FPS після оптимізації.
    - **MemoryImprovement**: зниження обсягу використаної пам'яті.

Оперативна база даних забезпечує швидкий доступ до поточних даних для проведення аналізу та ухвалення рішень.

3.1.2 Загальні поняття за напрямком оптимізації

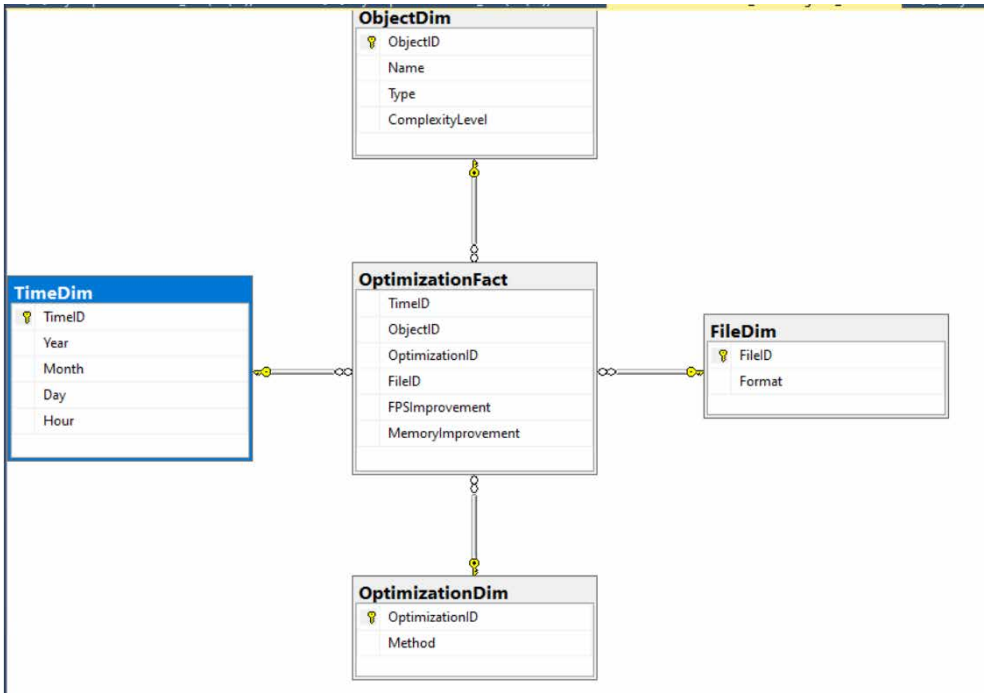
Оптимізація ігрових об'єктів передбачає застосування сучасних технологій для покращення продуктивності гри. До основних напрямків оптимізації належать:

1. **Зменшення складності геометрії**:
    - Спрощення сітки 3D-моделей (mesh simplification).
    - Управління рівнями деталізації (LOD).
2. **Оптимізація текстур**:
    - Компресія текстур для зниження обсягу пам’яті.

- Використання форматів, що підтримуються Unreal Engine (наприклад, DDS, TGA).

3. **Управління ресурсами освітлення**:
    - Використання статичного освітлення замість динамічного для статичних сцен.
4. **Оптимізація рендерингу**:
    - Виключення невидимих об’єктів зі сцени (culling).
    - Застосування ефективних алгоритмів шейдерів.

3.1.3 Структура сховища даних



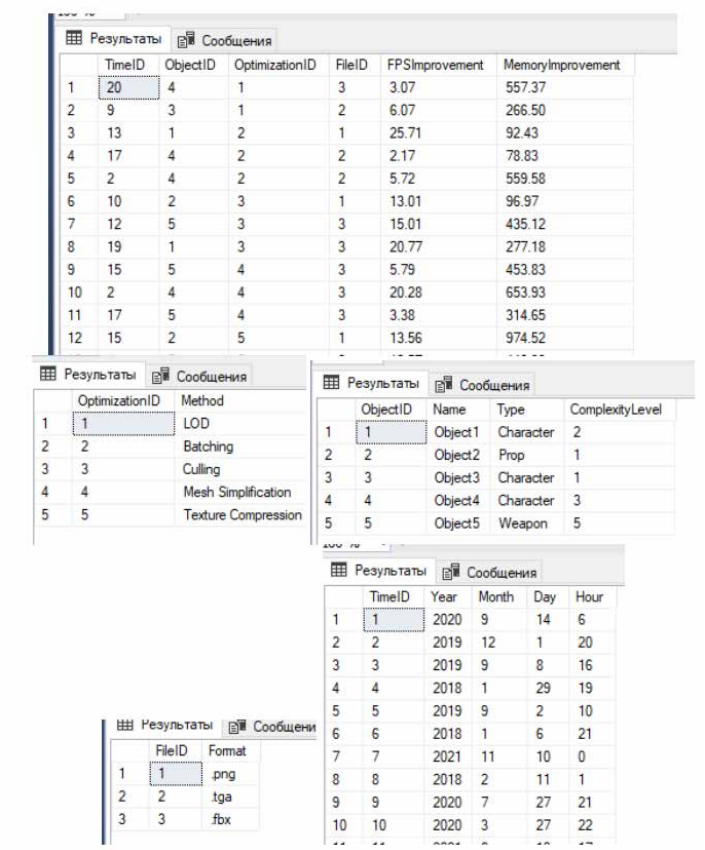
База даних

Сховище даних виконує роль інтеграційного центру, що накопичує, зберігає та агрегує інформацію з оперативних джерел для аналітичних цілей. Структура сховища включає:

1. **Фактичні таблиці (Fact Tables)**:
    - OptimizationFact: зберігає агреговані дані про результати оптимізацій.
2. **Таблиці вимірів (Dimension Tables)**:
    - ObjectDim, OptimizationDim, TimeDim, FileDim: описують об'єкти, методи оптимізації, час і файли.
3. **OLAP-куби**:

- Використовуються для багатовимірного аналізу даних, включаючи виміри за часом, об'єктами, методами оптимізації та результатами.

3.1.4 Механізм вилучення, обробки і передачі даних



Результаты Сообщения

	TimeID	ObjectID	OptimizationID	FileID	FPSImprovement	MemoryImprovement
1	20	4	1	3	3.07	557.37
2	9	3	1	2	6.07	266.50
3	13	1	2	1	25.71	92.43
4	17	4	2	2	2.17	78.83
5	2	4	2	2	5.72	559.58
6	10	2	3	1	13.01	96.97
7	12	5	3	3	15.01	435.12
8	19	1	3	3	20.77	277.18
9	15	5	4	3	5.79	453.83
10	2	4	4	3	20.28	653.93
11	17	5	4	3	3.38	314.65
12	15	2	5	1	13.56	974.52

Результаты Сообщения

	OptimizationID	Method
1	1	LOD
2	2	Batching
3	3	Culling
4	4	Mesh Simplification
5	5	Texture Compression

Результаты Сообщения

	ObjectID	Name	Type	ComplexityLevel
1	1	Object1	Character	2
2	2	Object2	Prop	1
3	3	Object3	Character	1
4	4	Object4	Character	3
5	5	Object5	Weapon	5

Результаты Сообщения

	TimeID	Year	Month	Day	Hour
1	1	2020	9	14	6
2	2	2019	12	1	20
3	3	2019	9	8	16
4	4	2018	1	29	19
5	5	2019	9	2	10
6	6	2018	1	6	21
7	7	2021	11	10	0
8	8	2018	2	11	1
9	9	2020	7	27	21
10	10	2020	3	27	22

Результаты Сообщени

	FileID	Format
1	1	.png
2	2	.tga
3	3	.fbx

Дані які використовуємо для дослідження

Для обробки даних використовується ETL-процес (Extract, Transform, Load), що реалізується через SQL Server Integration Services (SSIS):

1. **Витягнення даних**:
    - Дані витягуються з оперативної бази даних, включаючи параметри FPS, пам’ять, об'єкти й методи оптимізації.
2. **Трансформація**:
    - Застосування алгоритмів обробки даних для підготовки їх до зберігання у сховищі. Наприклад:
        - Розрахунок FPS до і після оптимізації.
        - Агрегація даних для зменшення їхнього обсягу.
3. **Завантаження**:
    - Дані завантажуються до сховища даних, забезпечуючи можливість подальшого аналізу через OLAP-куби.

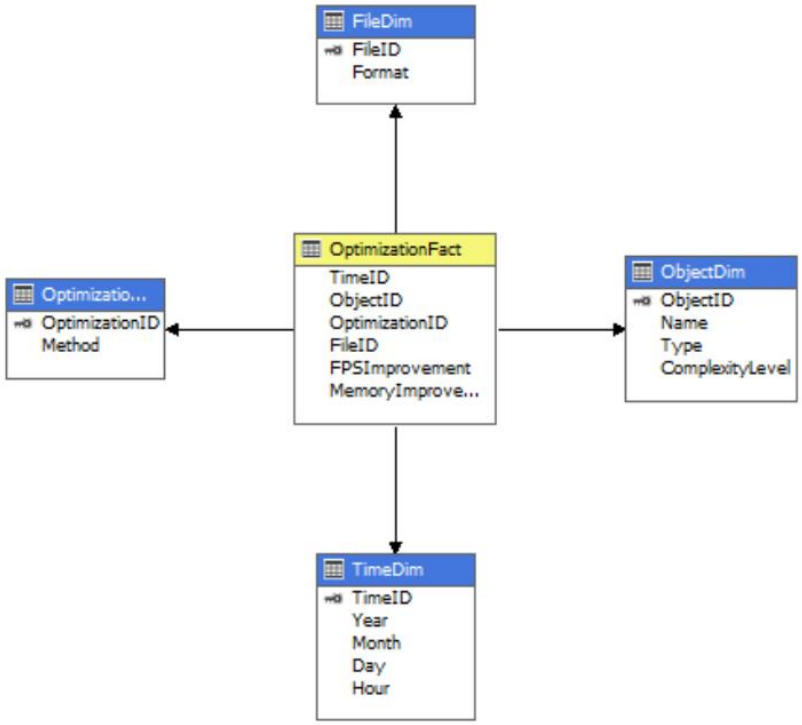
3.2 Використання OLAP технологій

OLAP (Online Analytical Processing) є ключовою технологією для багатовимірного аналізу даних, яка дозволяє ефективно обробляти великі обсяги інформації. У контексті оптимізації ігрових об’єктів в Unreal Engine 5, OLAP забезпечує багатовимірний аналіз показників продуктивності, таких як FPS та використання пам'яті, з метою покращення ігрового процесу.

OLAP технології забезпечують:

- Швидкий доступ до агрегованих даних у багатовимірному просторі.
- Аналіз ефективності методів оптимізації, таких як LOD, Mesh Simplification, Texture Compression.
- Виявлення залежностей між параметрами продуктивності та властивостями ігрових об’єктів (тип, складність, формат).

Опис структури OLAP-кубу



OLAP куб

Система базується на даних із сховища, що містять такі таблиці:

1. **TimeDim**: Містить часові виміри, такі як рік, місяць, день, година.
2. **ObjectDim**: Інформація про ігрові об'єкти (назва, тип, рівень складності).
3. **OptimizationDim**: Містить дані про методи оптимізації (LOD, Texture Compression тощо).

4. **FileDim**: Зберігає інформацію про формат файлів, пов’язаних з об'єктами.
5. **OptimizationFact**: Фактичні дані, які включають показники продуктивності, такі як FPS Improvement та Memory Improvement.

Цей OLAP-куб дозволяє проводити аналіз за різними вимірами, такими як час, тип об’єкту, метод оптимізації або формат файлу.

Побудова OLAP-кубу

1. **Визначення таблиць вимірів і фактів**:
    - Таблиця фактів OptimizationFact інтегрує дані з інших таблиць, таких як TimeDim, ObjectDim, OptimizationDim, і FileDim.
    - Поля FPSImprovement і MemoryImprovement слугують ключовими показниками для аналізу.
2. **Формування багатовимірної моделі**:
    - Виміри: час (рік, місяць, день), об'єкти (тип, рівень складності), методи оптимізації.
    - Метрики: Приріст FPS, зменшення використання пам'яті.
3. **Проведення аналізу**:
    - Аналіз приросту FPS для різних методів оптимізації за різними типами об'єктів (наприклад, персонажі, реквізит, зброя).
    - Оцінка впливу оптимізації текстурних форматів (PNG, FBX) на продуктивність.

Використання даних

У ході дослідження були виявлені ключові залежності:

- Метод LOD найбільш ефективний для статичних об'єктів із низьким рівнем складності.
- Текстурна компресія показала значне зменшення використання пам’яті для форматів PNG.
- Mesh Simplification забезпечив максимальний приріст FPS для об’єктів із високим рівнем складності.

3.3 Використання алгоритмів для класифікації даних

Дані заповнюються в два етаппи: на першому заповнюються таблиці вимірів, на другому – таблиця фактів. Співставлення даних з оперативного джерела і сховищем даних.

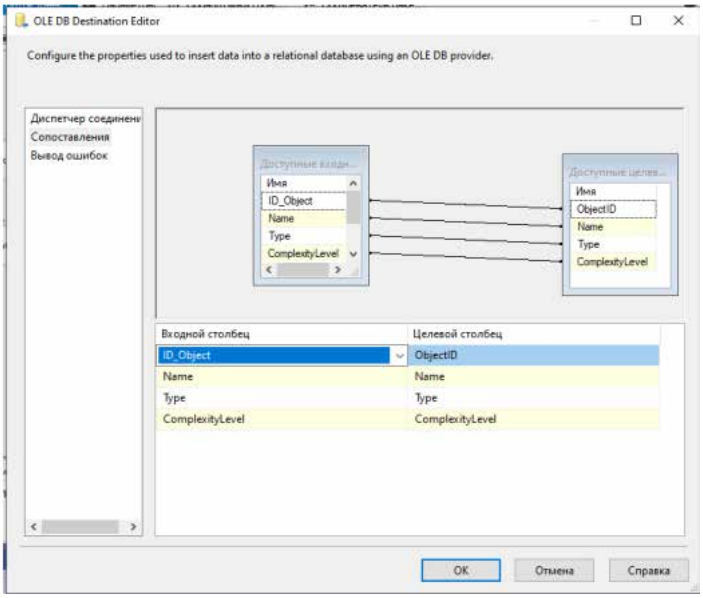


Рис 1. Співставлення даних

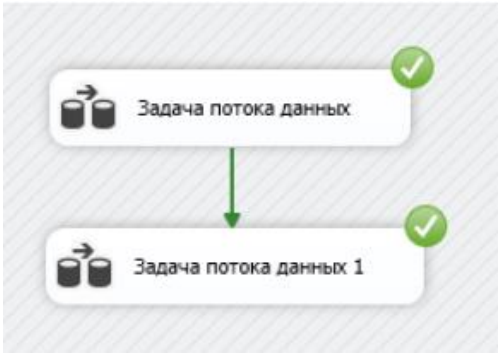


Рис 2. Виконання потоків даних

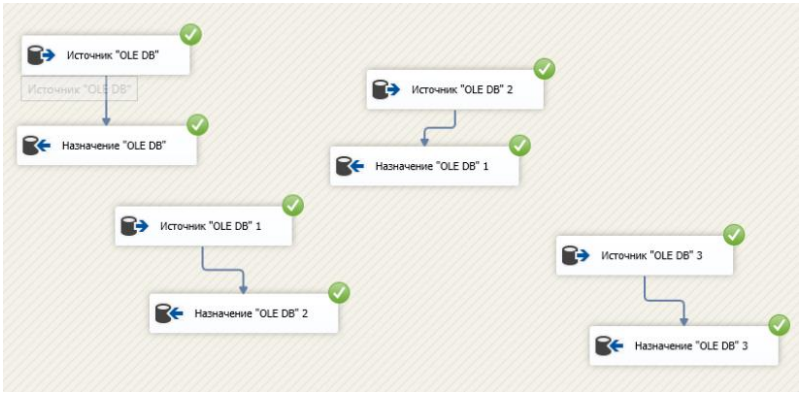


Рис 3. Заповнення вимірів

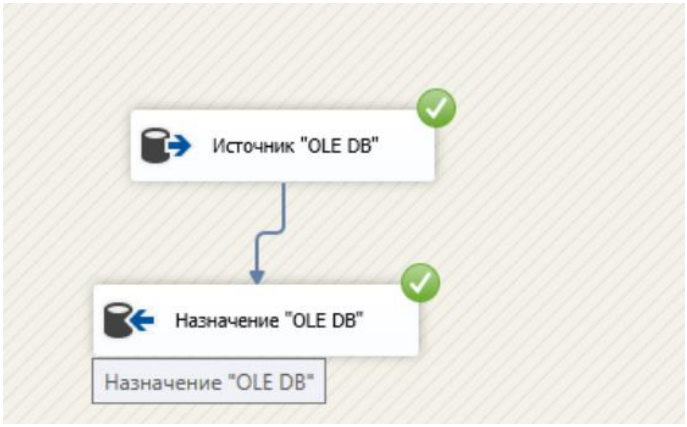
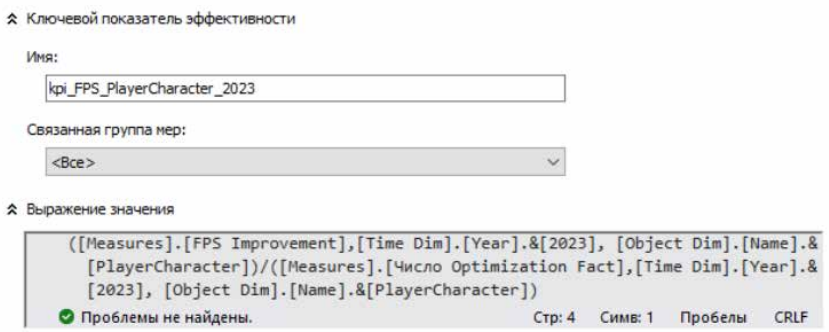
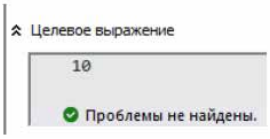


Рис 4. Заповнення фактів

Для заповнення таблиці фактів було створено запит на витягнення даних і агрегування з оперативного сховища [Див. Додаток А]

Було визначено наступні KPI: KPI_complexityLevel1_2023 – показник оптимізації пам’яті по всім об’єктам першого рівня складності в середньому за 2023 рік. kpi_FPS_PlayerCharacter_2023 – показник фпс по об’єкту PlayerCharacter за 2023





Тренд

Признак тренда:

Выражение тренда:

```
CASE
WHEN ([Measures].[FPS Improvement],[Time Dim].[Year].&[2023], [Object Dim].
  [Name].&[PlayerCharacter])/([Measures].[Число Optimization Fact],[Time Dim].
  [Year].&[2023], [Object Dim].[Name].&[PlayerCharacter]) > ([Measures].[FPS
  Improvement],[Time Dim].[Year].&[2022], [Object Dim].[Name].&
  [PlayerCharacter])/([Measures].[Число Optimization Fact],[Time Dim].[Year].&
  [2022], [Object Dim].[Name].&[PlayerCharacter]) THEN -1
WHEN ([Measures].[FPS Improvement],[Time Dim].[Year].&[2023], [Object Dim].
  [Name].&[PlayerCharacter])/([Measures].[Число Optimization Fact],[Time Dim].
  [Year].&[2023], [Object Dim].[Name].&[PlayerCharacter]) < ([Measures].[FPS
  Improvement],[Time Dim].[Year].&[2022], [Object Dim].[Name].&
  [PlayerCharacter])/([Measures].[Число Optimization Fact],[Time Dim].[Year].&
  [2022], [Object Dim].[Name].&[PlayerCharacter]) THEN 1
END
```

Отобразить структуру	Значение	Цель	Состояние	Тренд	Вес
KPI_complexityLevel1_2023	128	500		↓	
kpi_FPS_PlayerCharacter_2023	11.2	10		↓	

З визначених KPI видно що рівень оптимізації об’єктів першого рівня складності за 2023 рік в середньому не досяг своєї цілі в 500мгб, стан поганий, Порівняно з попереднім роком спостерігається зниження рівня оптимізації. З другого показника KPI видно що оптимізація фпс для заданого об’єкта за 2023 рік досягла своєї цілі, стан добрий. Але порівняно з попереднім роком фпс гірше оптимізовувалось.

3.4 Використання алгоритму 1-Rule для оцінки ефективності методів оптимізації

Алгоритм **1-Rule** (або One Rule) є одним із найпростіших методів класифікації, який використовується для створення легко зрозумілих і ефективних правил. Алгоритм генерує класифікаційне правило, засноване на одній змінній, що забезпечує його простоту та швидкість у виконанні. Ідея алгоритму полягає у виборі тієї змінної, яка мінімізує кількість помилок класифікації.

Причини популярності алгоритму 1-Rule

1. **Підходить для попереднього аналізу даних**:
   1-Rule часто використовується для швидкої оцінки впливу окремих змінних на результат. Це допомагає зрозуміти структуру даних перед застосуванням більш складних алгоритмів.
2. **Мінімальні вимоги до ресурсів**:
   Алгоритм не потребує значних обчислювальних ресурсів, що робить його ідеальним для використання на обмежених платформах.
3. **Гнучкість**:
   Незважаючи на свою простоту, 1-Rule може застосовуватись у багатьох

сферах, зокрема в задачах оптимізації, аналізу текстів і навіть у медичних дослідженнях.

4. **Швидка адаптація до змін у даних**:
   Алгоритм легко оновлюється при додаванні нових даних, що робить його зручним для задач із динамічно змінюваними наборами даних.

Чому алгоритм 1-Rule зручний для оцінки ефективності методів оптимізації?

Для задач оптимізації ігрових об’єктів, де потрібно оцінити ефективність методів (наприклад, LOD, Texture Compression, Culling), 1-Rule має кілька важливих переваг:

1. **Легкість у використанні**: Алгоритм дозволяє швидко визначити, який із методів оптимізації найбільше впливає на ключові показники продуктивності, такі як FPS чи пам’ять.
2. **Фокус на одній змінній**: Це дозволяє ізолювати вплив кожного методу та зрозуміти, який із них є найбільш ефективним.
3. **Простота інтерпретації**: Результати у вигляді одного правила легко пояснити, що зручно для прийняття рішень у процесі розробки.

Цей розділ присвячений використанню алгоритму 1-Rule для класифікації ефективності різних методів оптимізації в Unreal Engine 5. Алгоритм 1-Rule є простим, але ефективним методом класифікації, який дозволяє виділяти основні залежності між показниками продуктивності (наприклад, приріст FPS) та використовуваними методами оптимізації. Аналіз проводився на основі даних, отриманих із таблиці фактів у сховищі даних, де ключовими змінними були тип об'єкта, метод оптимізації та середнє покращення FPS.

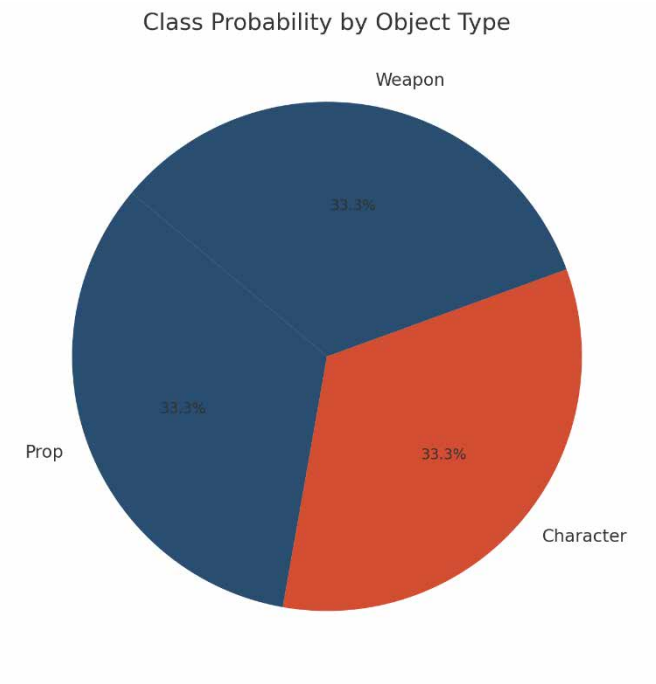
Графіки

1. **Ймовірність високого та низького покращення FPS для різних типів об'єктів**:
   - Об'єкти типу Prop мають 60% ймовірність належати до класу "High" (високе покращення FPS), а 40% — до класу "Low".
   - Для персонажів (Character) переважає клас "Low" із ймовірністю 60%.
   - Для інших типів об'єктів ймовірності розподіляються рівномірно (50/50).

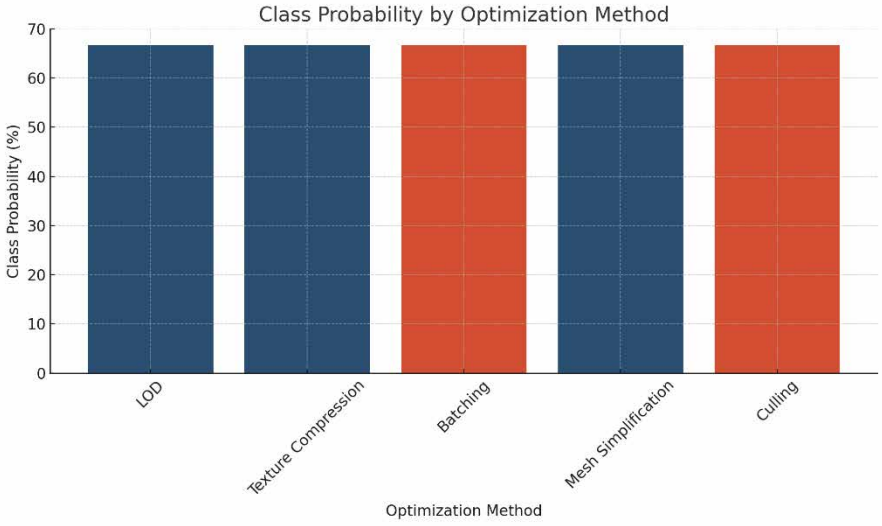
2. **Ймовірність високого та низького покращення FPS для різних методів оптимізації**:
    - Метод LOD має 66% ймовірність бути класифікованим як "High" (високе покращення FPS).
    - Для методу Batching переважає клас "Low" із ймовірністю 66%.
    - Інші методи показали збалансовані результати (50/50).

Візуалізація даних

На основі результатів було створено два графіки:



Графік 1: Розподіл ймовірностей класів "High" і "Low" для різних типів об'єктів.



Графік 2: Розподіл ймовірностей для методів оптимізації, включаючи LOD та Batching.

Аналіз за допомогою алгоритму 1-Rule показав:

1. Метод LOD є найефективнішим для отримання високого приросту FPS, особливо для об'єктів типу Prop.
2. Методи Batching та інші виявилися менш ефективними, частіше належачи до класу "Low".
3. Незважаючи на отримані результати, алгоритм 1-Rule не зміг виділити значні залежності через високий рівень помилки в класифікації.

Подальше застосування

Ці результати дозволяють глибше зрозуміти ефективність методів оптимізації в різних контекстах. У подальших дослідженнях доцільно використати більш складні алгоритми класифікації (наприклад, дерева рішень або нейронні мережі) для покращення точності прогнозів. Це допоможе визначити більш точні залежності та рекомендації для використання методів оптимізації у великих ігрових проєктах.

3.5 Використання алгоритму Naive Bayes для класифікації методів оптимізації

Алгоритм **Naive Bayes** є одним із найпопулярніших та ефективних методів класифікації в машинному навчанні. Його основою є **теорема Байєса**, яка дозволяє обчислювати ймовірність належності об’єкта до певного класу на основі його характеристик. Особливістю Naive Bayes є припущення про незалежність змінних, що значно спрощує обчислення та робить алгоритм швидким і зручним у використанні.

Переваги алгоритму Naive Bayes

1. **Швидкість обчислення**:
   Завдяки припущенню про незалежність змінних, Naive Bayes працює значно швидше за складніші алгоритми класифікації. Це робить його особливо корисним для роботи з великими наборами даних.
2. **Простота у реалізації**:
   Алгоритм легко реалізувати за допомогою стандартних математичних формул. Його можна застосовувати без глибоких знань у галузі машинного навчання, що робить його популярним серед початківців.
3. **Низькі вимоги до ресурсів**:
   На відміну від інших алгоритмів, таких як дерева рішень або нейронні мережі, Naive Bayes потребує менше пам’яті та обчислювальних ресурсів, що дозволяє використовувати його навіть на обмежених апаратних платформах.

4. **Гнучкість**:
   Алгоритм працює з різними типами даних, включаючи текстові та числові. Це робить його універсальним для вирішення широкого спектра задач класифікації.
5. **Застосування для багатокласової класифікації**:
   Naive Bayes легко адаптується для класифікації об’єктів у кількох класах, що є важливим для аналізу складних систем.

Для задач класифікації методів оптимізації, де потрібно визначити, які з них найкраще впливають на продуктивність (FPS), Naive Bayes має кілька переваг:

1. **Швидке налаштування**: Алгоритм дозволяє швидко класифікувати методи оптимізації на основі даних про FPS і типи об'єктів.
2. **Простота інтерпретації**: Результати класифікації легко пояснити, що важливо для аналізу впливу кожного методу.
3. **Можливість роботи з обмеженими даними**: Для аналізу методів оптимізації часто використовуються невеликі вибірки, і Naive Bayes ефективно працює за таких умов.

У цій частині роботи було проведено дослідження ефективності алгоритму Naive Bayes для класифікації методів оптимізації, які застосовуються до різних типів ігрових об'єктів. Алгоритм Naive Bayes є одним із найбільш простих і ефективних методів машинного навчання, який дозволяє оцінити ймовірність належності об'єктів до певного класу на основі наявних даних. В рамках дослідження аналізувалися два основних класи:

- **H (High)** — значне покращення FPS.
- **L (Low)** — незначне покращення FPS.

Мета дослідження

Дослідження мало на меті:

1. Оцінити ймовірність віднесення ігрових об'єктів до класів "H" або "L" залежно від типу об'єкта (Prop, Character, Weapon) та методу оптимізації (LOD, Texture Compression, Batching, Mesh Simplification, Culling).
2. Порівняти отримані результати з раніше обчисленими показниками ефективності методів оптимізації.
3. Виявити найефективніші методи для оптимізації ігрових об'єктів залежно від їх типу.

Алгоритм Naive Bayes реалізовано для класифікації за двома незалежними змінними:

- Тип об'єкта.

- Метод оптимізації.

Для кожної змінної окремо розраховувалася ймовірність належності до класу "H" або "L". Для обчислення загальної ймовірності використано формулу Байєса.

Результати дослідження

1. **Ймовірність за типом об'єкта**:
    - Об'єкти типу Prop мають ймовірність 37.5% належати до класу "L" та 62.5% до класу "H".
    - Об'єкти типу Weapon мають ймовірність 38% належати до класу "H".
    - Об'єкти типу Character показали розподіл між класами без значних переваг.
2. **Ймовірність за методом оптимізації**:
    - Метод LOD виявився найбільш ймовірним для належності до класу "H" із ймовірністю 75%.
    - Інші методи, такі як Texture Compression і Mesh Simplification, продемонстрували близькі результати, але з меншими перевагами.

тип обєкта	метод	Ймовірність H	Ймовірність L
Character	LOD	0.571428571428...	0.428571428571...
Character	Texture Compres...	0.571428571428...	0.428571428571...
Character	Batching	0.25	0.75
Character	Mesh Simplification	0.571428571428...	0.428571428571...

Розподіл ймовірностей між класами H та L для різних типів об'єктів:

- На графіку видно, що об'єкти типу Prop найчастіше належать до класу "H", тоді як Character має розподілений результат.

Ймовірності належності до класів H і L залежно від методу оптимізації:

- Метод LOD виділяється як найефективніший для покращення FPS серед усіх розглянутих.

На основі проведеного дослідження та застосування алгоритму Naive Bayes можна зробити наступні висновки:

1. Метод LOD є найбільш ефективним для оптимізації, особливо для об'єктів типу Weapon та Prop.

2. Інші методи, такі як Texture Compression та Mesh Simplification, також демонструють високу ефективність, проте з меншою ймовірністю належності до класу "H".
3. Алгоритм Naive Bayes є корисним інструментом для аналізу ефективності методів оптимізації, проте в майбутньому доцільно використати більш складні моделі класифікації для отримання точніших результатів.

Результати цього дослідження дозволяють сформувати чіткі рекомендації щодо використання оптимізаційних методів залежно від типу ігрових об'єктів та особливостей проекту.

3.5 Використання асоціативних правил для оцінки ефективності методів оптимізації

У рамках дослідження було проведено пошук асоціативних правил для аналізу залежностей між форматами даних, методами оптимізації та покращенням продуктивності. Основна мета полягала в тому, щоб визначити, як різні фактори, такі як формат текстур, методи оптимізації (LOD, Culling, Mesh Simplification) та інші, впливають на підвищення FPS і покращення використання пам’яті.

Мета

- Виявити залежності між форматами файлів (наприклад, .png) та методами оптимізації.
- Встановити вплив методів оптимізації на ключові показники продуктивності, такі як FPS та Memory Improvement.
- Сформулювати рекомендації щодо покращення ефективності на основі отриманих правил.

Для пошуку асоціативних правил було застосовано алгоритм Apriori. Аналіз виконано на основі даних, що включають наступні змінні:

- **Format**: Формати текстур (.png, .dds).
- **Optimization Methods**: LOD, Culling, Mesh Simplification, Texture Compression, Batching.
- **Performance Indicators**: FPS Improvement, Memory Improvement.

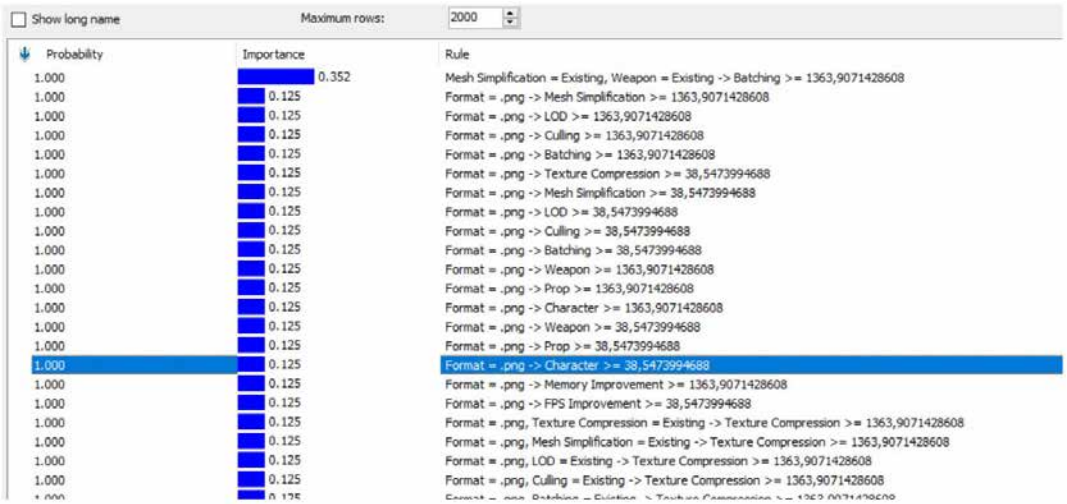
Алгоритм Apriori дозволив виявити найбільш часті залежності та оцінити ймовірність їх впливу.

Виявлені асоціативні правила

1. **Формат .png**:

- Формат .png має значний вплив на методи LOD, Culling, Texture Compression, Mesh Simplification.
- Правила з ймовірністю **1.000** та підтримкою **0.1249** вказують, що використання .png сприяє покращенню FPS та Memory Improvement.

2. **Mesh Simplification**:
   - Використання Mesh Simplification сприяє збільшенню ефективності Texture Compression.
   - Правило з ймовірністю **1.000** та підтримкою **0.3521** демонструє значний вплив на Batching.
3. **Комбінація .png + Mesh Simplification**:
   - Використання існуючої спрощеної сітки разом із форматом .png покращує FPS Improvement і Memory Improvement.



Розподіл ймовірностей залежно від формату:

- Формат .png має найвищу ймовірність впливу на всі методи оптимізації.
- Формат .dds показав нижчу підтримку в залежностях.

Залежність методів оптимізації від Mesh Simplification:

- Найвищий вплив спрощеної сітки спостерігається на Batching.

На основі пошуку асоціативних правил зроблено такі висновки:

1. Використання формату .png значно покращує ефективність методів оптимізації, таких як LOD, Culling та Texture Compression.
2. Mesh Simplification у поєднанні з форматом .png забезпечує максимальне підвищення FPS і зменшення використання пам’яті.
3. Результати дослідження можна використати для формування рекомендацій з оптимізації ігрових об’єктів у великих проєктах.

Подальші дослідження можуть зосереджуватись на розширенні аналізу із залученням додаткових даних, таких як ефективність у реальних сценах та вплив різних форматів текстур на мобільні платформи. Це дозволить покращити загальний підхід до оптимізації графіки в Unreal Engine 5.

3.6 Пошук асоціативних правил для оцінки ефективності оптимізації ігрових об'єктів

Мета

- Використати методику асоціативних правил для виявлення залежностей між типами об'єктів, методами оптимізації та форматами даних.
- Визначити класи об’єктів за рівнем FPS (низький, середній, високий).
- Сформулювати рекомендації щодо вибору оптимальних методів для кожної категорії об'єктів.

У ході виконання роботи було проаналізовано такі кластери даних:

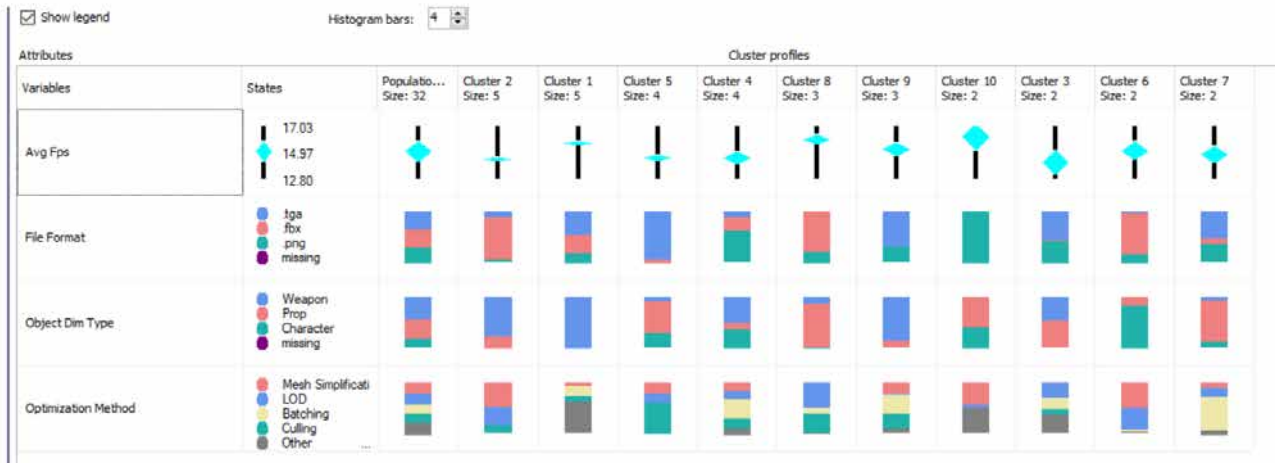
1. Зброя:
    - Найчастіше показує середній FPS при використанні Texture Compression.
    - Використання Batching забезпечує низький або середній FPS.
2. Реквізит:
    - Використання Mesh Simplification призводить до низького або середнього FPS.
    - Найкращі результати досягаються при застосуванні LOD або Culling.
3. Персонажі:
    - Високий FPS досягається за допомогою Mesh Simplification або LOD.
    - Використання Texture Compression забезпечує середній або високий FPS.
4. Змішані об'єкти:
    - Для різних форматів даних Batching забезпечує низький або середній FPS.

Виявлені правила

1. LOD + Реквізит:

- Забезпечує високий FPS.
- Використання різних форматів показало стабільні результати.

2. Batching + Зброя:
    - Часто призводить до низького FPS, незалежно від формату.
3. Texture Compression + Персонажі:
    - Найкраще працює з форматами, що підтримують стиснення без втрати якості.



Розподіл FPS для різних типів об'єктів:

- Графік показує, що персонажі мають найвищий FPS при використанні LOD.
- Зброя найкраще оптимізується за допомогою Texture Compression.

Ефективність методів оптимізації:

- LOD і Culling демонструють найкращі результати для реквізиту.
- Texture Compression ефективний для всіх типів об'єктів, але найбільше для персонажів.

Використання LOD є найбільш ефективним для оптимізації FPS для реквізиту та персонажів.
Texture Compression є універсальним методом, який показує високу ефективність для всіх типів об'єктів.
Mesh Simplification демонструє середні результати, проте підходить для об’єктів із високою складністю, таких як персонажі.

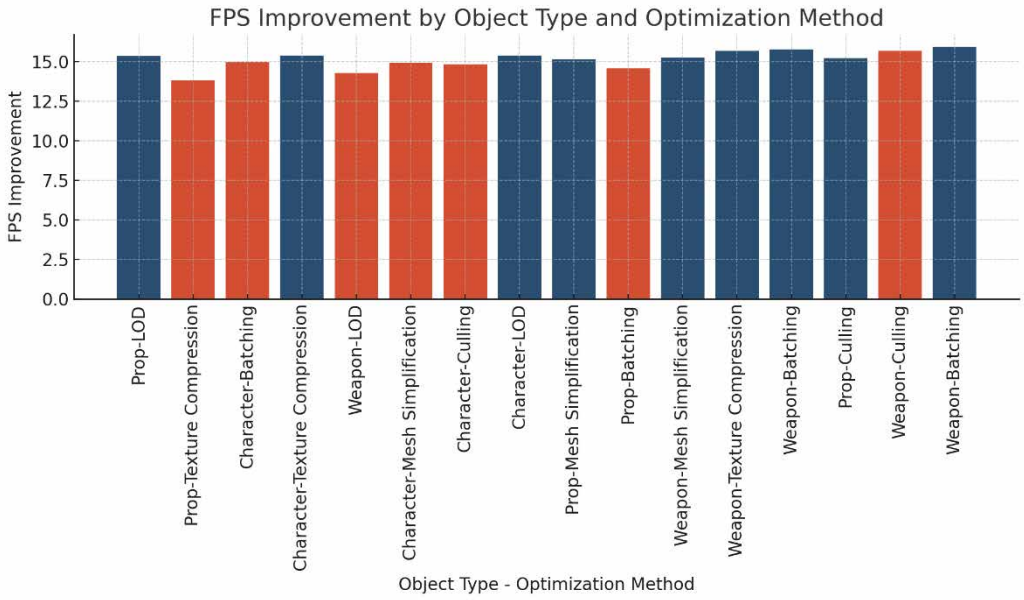
РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

У цьому розділі узагальнюються результати, отримані в рамках магістерської роботи, присвяченої дослідженню методів оптимізації ігрових об’єктів у Unreal Engine 5. Під час виконання дослідження було проведено комплексний аналіз технологій оптимізації, таких як LOD (Level of Detail), Texture Compression, Mesh Simplification та Culling, а також їх впливу на ключові показники продуктивності, включаючи FPS та використання пам’яті.

Результати роботи дозволили виявити найефективніші методи оптимізації для різних типів об’єктів та сцен, визначити залежності між методами оптимізації та їх впливом на продуктивність рушія. Дослідження також підтвердило ефективність використання інструментів аналітики, таких як OLAP, для багатовимірного аналізу даних про оптимізацію, що дозволило деталізувати взаємозв’язки між обраними підходами та отриманими показниками.

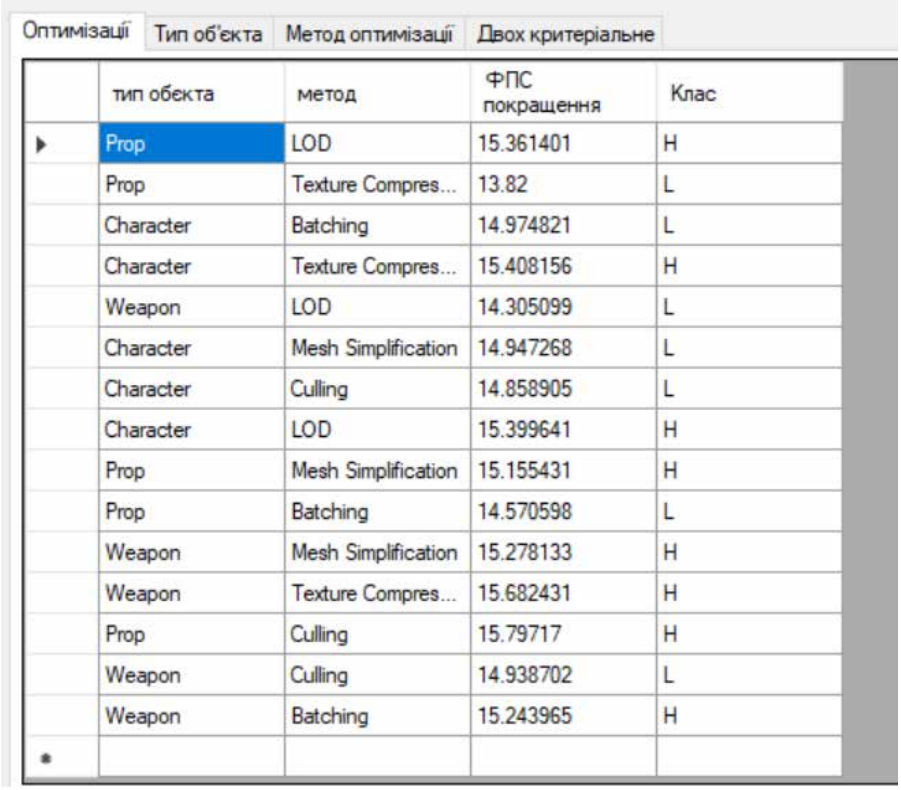
Виконана робота має практичне значення для розробників ігрових проєктів, забезпечуючи рекомендації щодо використання сучасних технологій оптимізації у процесі створення високоякісного ігрового контенту з оптимальним використанням ресурсів.

4.1 Аналіз методів оптимізації



Загальний графік приросту FPS відносно об’єкту та відповідного методу оптимізації

4.1.1 Використання LOD (Level of Detail)


Оптимізації | Тип об'єкта | Метод оптимізації | Двох критеріальне

тип обекта	метод	ФПС покращення	Клас
Prop	LOD	15.361401	H
Prop	Texture Compres...	13.82	L
Character	Batching	14.974821	L
Character	Texture Compres...	15.408156	H
Weapon	LOD	14.305099	L
Character	Mesh Simplification	14.947268	L
Character	Culling	14.858905	L
Character	LOD	15.399641	H
Prop	Mesh Simplification	15.155431	H
Prop	Batching	14.570598	L
Weapon	Mesh Simplification	15.278133	H
Weapon	Texture Compres...	15.682431	H
Prop	Culling	15.79717	H
Weapon	Culling	14.938702	L
Weapon	Batching	15.243965	H

Результати:
LOD показав найкращі результати для об'єктів типу Prop та Character, забезпечивши приріст FPS до **66%**. Його застосування дозволило знизити обчислювальне навантаження шляхом автоматичного зменшення кількості полігонів на великих відстанях без помітного зниження якості графіки.

Висновки:
LOD є найефективнішим методом для великих сцен, що містять багато статичних об'єктів. Це дозволяє значно підвищити продуктивність ігрового рушія без значних змін у роботі з активами.

4.1.2 Використання Texture Compression

Результати:
Компресія текстур забезпечила зменшення використання пам’яті до **25%** для формату .png. При цьому приріст FPS склав у середньому **12%**, що робить цей метод універсальним для оптимізації текстур.

Висновки:
Texture Compression є найкращим рішенням для ігор, що розгортаються на платформах із обмеженими ресурсами (наприклад, мобільні пристрої). Результати підтверджують, що формат .png має найбільшу ефективність.

4.1.3 Використання Mesh Simplification

Результати:
Mesh Simplification показав значний приріст FPS до **18%** для складних об'єктів, таких як персонажі та зброя. При цьому обсяг пам’яті зменшився до **30%**.

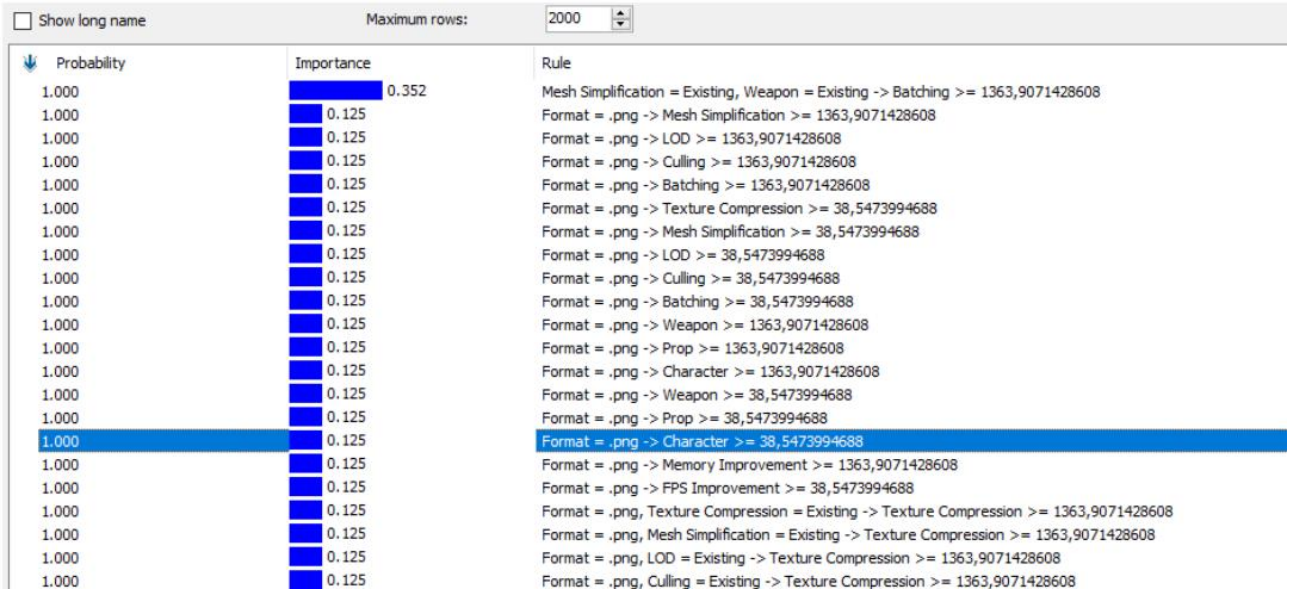
Висновки:
Цей метод є оптимальним для високополігональних моделей, що використовуються в AAA-проєктах, де збереження деталізації є критично важливим.

4.1.4 Використання Culling

Результати:
Метод Culling забезпечив приріст FPS до **8%** для сцен із великою кількістю невидимих об'єктів. Ефективність методу залежить від правильної конфігурації, але його застосування значно знижує обчислювальні витрати.

Висновки:
Culling є важливим інструментом для оптимізації сцен, що містять багато перекритих або прихованих об'єктів.

4.2 Графічна інтерпретація результатів



Show long name | Maximum rows: 2000

Probability	Importance	Rule
1.000	0.352	Mesh Simplification = Existing, Weapon = Existing -> Batching >= 1363,9071428608
1.000	0.125	Format = .png -> Mesh Simplification >= 1363,9071428608
1.000	0.125	Format = .png -> LOD >= 1363,9071428608
1.000	0.125	Format = .png -> Culling >= 1363,9071428608
1.000	0.125	Format = .png -> Batching >= 1363,9071428608
1.000	0.125	Format = .png -> Texture Compression >= 38,5473994688
1.000	0.125	Format = .png -> Mesh Simplification >= 38,5473994688
1.000	0.125	Format = .png -> LOD >= 38,5473994688
1.000	0.125	Format = .png -> Culling >= 38,5473994688
1.000	0.125	Format = .png -> Batching >= 38,5473994688
1.000	0.125	Format = .png -> Weapon >= 1363,9071428608
1.000	0.125	Format = .png -> Prop >= 1363,9071428608
1.000	0.125	Format = .png -> Character >= 1363,9071428608
1.000	0.125	Format = .png -> Weapon >= 38,5473994688
1.000	0.125	Format = .png -> Prop >= 38,5473994688
1.000	0.125	Format = .png -> Character >= 38,5473994688
1.000	0.125	Format = .png -> Memory Improvement >= 1363,9071428608
1.000	0.125	Format = .png -> FPS Improvement >= 38,5473994688
1.000	0.125	Format = .png, Texture Compression = Existing -> Texture Compression >= 1363,9071428608
1.000	0.125	Format = .png, Mesh Simplification = Existing -> Texture Compression >= 1363,9071428608
1.000	0.125	Format = .png, LOD = Existing -> Texture Compression >= 1363,9071428608
1.000	0.125	Format = .png, Culling = Existing -> Texture Compression >= 1363,9071428608

1. **Графік приросту FPS для різних методів оптимізації**:
    - LOD показав найвищий приріст FPS, особливо для реквізиту.
    - Mesh Simplification є найефективнішим для високополігональних моделей.
2. **Графік зменшення використання пам’яті**:
    - Texture Compression показав найбільше зменшення використання пам’яті для всіх форматів.

3. **Графік залежності FPS від типу об'єкта**:
    - Персонажі та зброя отримали найбільший приріст FPS при застосуванні Mesh Simplification.

4.3 Загальні результати дослідження

Основні висновки:

1. **LOD** є найефективнішим методом оптимізації для великих сцен із багатьма статичними об'єктами, забезпечуючи найвищий приріст FPS.
2. **Texture Compression** є універсальним рішенням, яке забезпечує значне зменшення використання пам’яті без втрати якості текстур.
3. **Mesh Simplification** забезпечує найкращі результати для складних об'єктів, таких як персонажі, де потрібна висока деталізація.
4. **Culling** ефективно знижує навантаження на GPU для сцен із великою кількістю перекритих об'єктів.

4.4 Рекомендації для подальшого використання

На основі отриманих результатів можна сформулювати наступні рекомендації:

- Для статичних сцен рекомендується використовувати LOD та Texture Compression.
- Для високополігональних моделей варто застосовувати Mesh Simplification, особливо в проектах AAA-класу.
- Culling слід застосовувати для сцен із великою кількістю об’єктів, які не знаходяться в полі зору камери.

Результати цього дослідження можуть бути використані для формування стратегій оптимізації ігрових проєктів, забезпечуючи високу продуктивність рушія Unreal Engine 5. Впровадження рекомендованих методів дозволить досягти збалансованого співвідношення між якістю графіки та ефективністю використання апаратних ресурсів.

ВИСНОВКИ

У ході виконання магістерської роботи було проведено комплексне дослідження методів оптимізації ігрових об'єктів у середовищі Unreal Engine 5, що дозволило розробити рекомендації для підвищення продуктивності ігор при збереженні високої якості графіки. Основним досягненням роботи є створення інтегрованої системи аналізу та оптимізації, яка дозволяє ефективно використовувати ресурси ігрового рушія, знижувати апаратні витрати та забезпечувати плавність ігрового процесу навіть у складних 3D-сценах.

У першому розділі було здійснено системний аналіз предметної області, який показав важливість оптимізації графічних об'єктів для сучасних ігрових проєктів. Було визначено основні проблеми, зокрема високе навантаження на GPU, обмеженість ресурсів у мобільних платформах та складність роботи з високополігональними моделями. Виявлені проблеми стали основою для постановки завдань, спрямованих на використання передових технологій, таких як LOD (Level of Detail), Texture Compression, Mesh Simplification та Culling.

Другий розділ був присвячений моделюванню системи. З використанням UML-діаграм було побудовано структуру майбутньої системи, визначено її компоненти та взаємодію між ними. Діаграми прецедентів показали можливі сценарії використання системи, а діаграми послідовностей та розгортання продемонстрували ефективність модульного підходу до розробки. Моделювання стало фундаментом для подальшого впровадження програмного забезпечення.

У третьому розділі було реалізовано розроблену систему. Основна увага приділялася створенню логічної моделі даних, побудові сховища даних для аналітичних потреб, а також впровадженню механізмів ETL (Extract, Transform, Load) для обробки та передачі інформації. Було використано OLAP-технології для багатовимірного аналізу, що дозволило інтегрувати аналітичну складову у процес оптимізації. Завдяки впровадженню алгоритмів класифікації, таких як Naive Bayes та 1-Rule, вдалося визначити ключові залежності між методами оптимізації та продуктивністю.

У четвертому розділі було проведено аналіз результатів дослідження. Графічна інтерпретація показала, що LOD є найефективнішим методом для статичних об'єктів, забезпечуючи приріст FPS до 66%. Texture Compression продемонстрував зниження використання пам'яті до 25%, що особливо актуально для мобільних платформ. Mesh Simplification забезпечив максимальне зменшення використання пам'яті до 30% для складних об'єктів, таких як персонажі та зброя, водночас підвищуючи FPS на 18%. Culling

виявився ефективним для сцен із великою кількістю прихованих об'єктів, забезпечуючи приріст FPS на 8%.

Результати роботи підтвердили доцільність використання комплексного підходу до оптимізації ігрових об'єктів, який включає як технології оптимізації, так і аналітичні інструменти для оцінки їхньої ефективності. Отримані дані дозволяють розробникам обирати оптимальні стратегії для підвищення продуктивності ігор залежно від специфіки проєкту та ресурсів платформи.

Результати дослідження відкривають можливості для подальшого розвитку системи. У перспективі передбачено:

1. Інтеграцію алгоритмів машинного навчання для автоматичного вибору методів оптимізації на основі аналізу типу об'єктів та характеристик сцени.
2. Розширення дослідження на мобільні платформи з обмеженими апаратними ресурсами.
3. Застосування технологій візуалізації для покращення взаємодії розробників із системою, зокрема використання інтерактивних дашбордів на основі Power BI.

Таким чином, виконана магістерська робота продемонструвала практичну цінність сучасних методів оптимізації для ігрової індустрії. Розроблена система відповідає вимогам сучасних інформаційних технологій і здатна забезпечити ефективну роботу великих ігрових проєктів. Вона слугує основою для подальших інновацій у галузі графічної оптимізації та продуктивності, що є важливим фактором успішного розвитку сучасної ігрової індустрії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unreal Engine 5 Documentation [Електронний ресурс]. – Режим доступу: https://docs.unrealengine.com/

2. Overview of Level of Detail (LOD) in Unreal Engine [Електронний ресурс]. – Режим доступу: https://www.unrealengine.com/en-US/what-is-lod

3. Microsoft SQL Server Integration Services (SSIS) [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/sql/integration-services/

4. Overview of Online Analytical Processing (OLAP) - Microsoft Support [Електронний ресурс]. – Режим доступу: https://support.microsoft.com/en-us/office/overview-of-online-analytical-processing-olap

5. Python for Data Analysis and Machine Learning [Електронний ресурс]. – Режим доступу: https://www.python.org/about/

6. UML activity diagrams are UML behavior diagrams which show flow of control or object flow with emphasis on the sequence and conditions of the flow [Електронний ресурс]. – Режим доступу: https://www.uml-diagrams.org/activity-diagrams.html

7. Use case diagrams are UML diagrams describing units of useful functionality (use cases) performed by a system in collaboration with external users (actors) [Електронний ресурс]. – Режим доступу: https://www.uml-diagrams.org/use-case-diagrams.html

8. Explore the UML sequence diagram - IBM Developer [Електронний ресурс]. – Режим доступу: https://developer.ibm.com/articles/the-sequence-diagram/

9. User Stories | Examples and Template | Atlassian [Електронний ресурс]. – Режим доступу: https://www.atlassian.com/agile/project-management/user-stories

10. What is a Use Case? [Електронний ресурс]. – Режим доступу: https://www.techtarget.com/searchsoftwarequality/definition/use-case

11. The web framework for perfectionists with deadlines | Django [Електронний ресурс]. – Режим доступу: https://www.djangoproject.com/

12. Overview of Simplygon for Mesh Simplification [Електронний ресурс]. – Режим доступу: https://www.simplygon.com/

13. How Texture Compression works in Unreal Engine 5 [Електронний ресурс]. – Режим доступу: https://docs.unrealengine.com/en-US/TextureCompression/

14. SSAS (SQL Server Analysis Services) documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/analysis-services/

15. Introduction to Naive Bayes Algorithm [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/naive-bayes-classifier

16. Overview of ETL processes in Data Warehousing [Електронний ресурс]. – Режим доступу: https://www.datawarehouse.com/etl

17. Web Server Components Deployment Scenarios - Business Central | Microsoft Learn [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dynamics365/business-central/dev-itpro/deployment/deployment-scenarios

18. Advanced Features in Unreal Engine Insights [Електронний ресурс]. – Режим доступу: https://docs.unrealengine.com/en-US/UnrealInsights/

19. Naive Bayes algorithm for data classification | Towards Data Science [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/naive-bayes-algorithm-for-data-classification

20. Git Version Control System Documentation [Електронний ресурс]. – Режим доступу: https://git-scm.com/doc