Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет Комп'ютерні технології та системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: Дослідження продуктивності ігрових рушіїв Unity та Unreal Engine

за освітньою програмою **12 Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи ПЗ2421 _____ /Ярослав ГРИВА/

Керівник: _____ /доц. Олександр ІВАНОВ/

Нормоконтролер: _____ /доц. Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань
Студент _____

Дніпро – 2026

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis

on the topic: Research on the performance of Unity and Unreal Engine game engines

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group П32421: / Yaroslav HRYVA/

Scientific Supervisor: /Oleksandr IVANOV/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2026

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ____КІТ
_____ Вадим ГОРЯЧКІН
_____ грудня 2025 р.

ЗАВДАННЯ

На кваліфікаційну роботу Магістр
студенту Грива Ярослав Васильович.
1. Тема дипломної роботи: Дослідження продуктивності ігрових рушіїв Unity та Unreal Engine
Керівник роботи: Іванов Олександр Петрович
затверджені наказом № 1401ст від 02.10.2025 року
2. Строк подання студентом роботи 09.01.2026
3. Вихідні дані до дипломної роботи:
Базові та ідентичні сцени створені на рушіях Unity та Unreal Engine.
4. Зміст пояснювальної записки (перелік питань до розробки):
4.1. Аналітична частина: Аналіз сучасного стану дослідження за науковими літературними джерелами;
4.2. Основна частина: Дослідження продуктивності ігрових рушіїв Unity та Unreal Engine;
5. Перелік демонстраційного матеріалу:
5.1. презентація;
5.2. демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	10.11.25	10%
2	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	12.11.25	
3	Постановка задачі, технічне завдання	13.11.25	30%
4	Розробка інструментальних засобів дослідження	12.02.25	
5	Виконання досліджень	05.01.26	60%
6	Оформлення пояснювальної записки	08.01.26	
7	Розробка демонстраційних матеріалів	13.01.26	100%
8	Подання кваліфікаційної роботи до кафедри	19.01.26	
9	Захист кваліфікаційної роботи на засідані Екзаменаційної комісії	21.01.26	

Студент: _____ Ярослав ГРИВА

Керівник роботи: _____ доц. Олександр ІВАНОВ

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 81 стр., 55 рис., 16 табл., 3 додатків., 8 джерел.,

У сучасному світі з величезною кількістю медіарозваг не останнє місце займають відеоігри — від малих мобільних проектів до грандіозних світів з високою деталізацією та проробленою механікою. Ці проекти дозволяють користувачам продуктивно та цікаво проводити час , заводити нові знайомства в онлайн-іграх, проявляти креативність, а також знімати стрес у складних життєвих ситуаціях.

Однак, незважаючи на їх популярність і різноманітність, більшість ігрових проектів стикаються з однією спільною проблемою — оптимізацією. Неправильна організація об’єктів у сцені, неефективне використання ресурсів процесора та відеокарти, а також невідповідний вибір рушія для розробки можуть суттєво знизити продуктивність гри та негативно вплинути на досвід користувача.

Особливо важливим є коректний вибір платформи для розробки. Від нього залежить не лише швидкість створення проекту, а й ефективність його роботи на різних пристроях, можливості оптимізації та подальшого масштабування.

Мета роботи : порівняння продуктивності двох найпопулярніших ігрових рушіїв — Unity та Unreal Engine при виконанні однакових графічних та фізичних задач.

Завданням є дослідження та визначення сильних та слабких сторін кожного рушія та необхідність оцінки ефективності їхньої роботи під різними рівнями навантаження. Та сформувати рекомендації щодо оптимального вибору платформи для розробки конкретних типів ігор (2D, 3D, мобільні, VR тощо).

Для досягнення цієї мети передбачено:

- аналіз архітектури, систем рендерингу та фізики обох рушіїв;

- розробку стандартизованих тестових сцен та інструментальних засобів для збору даних;
- проведення серії експериментів з однаковими умовами на апаратній платформі з використанням засобів фіксації від NVIDIA;
- порівняльний аналіз продуктивності за ключовими показниками (FPS, час кадру, навантаження GPU).

Результатом дослідження є створення програмно-методичного комплексу для об’єктивного порівняння продуктивності рушіїв та отримання практичних висновків щодо їхнього використання у різних типах ігрових проектів.

ЗМІСТ

Вступ 2

Розділ 1 Аналіз сучасного стану дослідження за науковими літературними джерелами 3

1.1 Основні поняття та мета дослідження 3

1.2 Сфера використання 6

Висновки до розділу 1 7

Розділ 2 Обгрунтування методів дослідження 9

2.1 Основні фунсціональні вимоги 9

2.2 Вхідні дані 9

2.3 Вихідні дані 12

Висновки до розділу 2 12

Розділ 3 Розробка інструментальних засобів для дослідження рушіїв Unity та Unreal Engine 13

3.1 Постановка задачі дослідження 13

3.2 Методология фіксації експеримента 13

Висновок до розділу 3 19

Розділ 4 Аналіз продуктивності рушіїв Unity та Unreal Engine 20

4.1 Експеримент №1 (створення великої кількості однакових об’єктів з використанням примітивів) 20

4.2 Експеримент №2 (створення великої кількості однакових об’єктів з симуляцією фізики та зіткнень) 32

4.2 Висновки часу рендерінгу кадру до експеримента №2 45

4.3 Експеримент №3 (створення великої кількості однакових багато полігональних об’єктів) 49

4.4 Експеримент №4 (створення великої кількості однакових багато полігональних об’єктів з симуляцією фізики) 62

4.5 Висновки що до часу рендерінгу кадру 73

ДОДАТОК А 78

ДОДАТОК Б 81

ДОДАТОК В 5

Вступ

У сучасному світі індустрія розваг - це потужна галузь, що поєднує творчість з бізнесом, залучаючи мільйони людей і генеруючи значні доходи.

Економічний двигун: Створює робочі місця та приваблює інвестиції.

Культурний вплив: Формує тренди, впливає на цінності та спосіб життя.

Соціальна функція: Надає можливість для відпочинку, релаксації та соціалізації.

В останні роки, ігрова індустрія вступає в нову еру. Вона не лише вийшла за межі мобільних пристроїв, але й стала нормою кросплатформної ігри, такі як PUBG або Fortnite та ігрові середовища для спілкування і хмарні ігрові сервіси на основі підписки від таких компаній - як PlayStation, Google, Amazon та інших. У багатьох відношеннях ця галузь сьогодні перебуває в центрі тенденцій соціальних мереж, електронної комерції та розваг.

Глобальний ринок вартістю понад 150 мільярдів доларів США зі зростаючою участю інвестеційного капіталу.

Історично в ігровій індустрії домінували великі гравці та інвестори, орієнтовані на ігри, але вона інтенсивно продовжила активно розвивалася в останні роки, особливо у період карантину пов'язаного з COVID-19. Швидке зростання в цьому секторі привабило безліч нових компаній, що створюються в цій сфері, а також геймерів, глядачів, фанатів і як наслідок - розширену базу інвесторів.

Практична робота полягає в створенні умов для проведення однакових експериментів та порівняння навантаження одинакових умов на різних ігрових рушіях . З отриманих результатів ми зможемо зробити висновки, щодо оптимізації рушіїв в різних умовах та взяти до уваги сильні та слабкі сторони для забезпечення оптимізації проектів .

РОЗДІЛ 1 АНАЛІЗ СУЧАСНОГО СТАНУ ДОСЛІДЖЕННЯ ЗА НАУКОВИМИ ЛІТЕРАТУРНИМИ ДЖЕРЕЛАМИ

1.1 Основні поняття та мета дослідження

Основна мета дослідження продуктивності ігрових рушіїв **Unity** та **Unreal Engine** полягає у виявленні та порівнянні їхніх технічних можливостей за допомогою ідентичних сцен та експериментів. Та виявлення ефективності використання ресурсів комп'ютера (процесора, оперативної пам'яті, відеокарти), а також визначення оптимальних умов застосування кожного рушія для створення різних типів ігор з метою вибору платформи для розробки.

Для досягнення поставленої мети треба:

- проаналізувати архітектуру і принципи роботи рушіїв Unity та Unreal Engine, зокрема системи рендерингу, управління фізикою, оптимізації та використання ресурсів;
- розробити інструментальні засоби збору продуктивності, які забезпечують однакові умови тестування для обох рушіїв ;
- створити стандартизовані тестові сцени, що імітують типові ситуації у 3D-іграх: динамічні об'єкти, освітлення, фізичні взаємодії та кількість об'єктів;
- провести серію експериментів для визначення продуктивності рушіїв при різному рівні навантаження;
- порівняти отримані результати за ключовими показниками: середній FPS, стабільність кадру, використання ресурсів відеокарти;
- зробити висновки про ефективність кожного рушія та визначити, який з них краще підходить для певних типів ігрових проектів (2D, 3D, мобільні, VR тощо).

Об'єкт експерименту

У цьому експерименті порівнюється поведінка ігрових рушіїв Unity і Unreal Engine за однакових тестових умов при виконанні типових обчислювальних та

графічних навантажень. Аналіз охоплює зміну параметрів сцени — зокрема кількість відображуваних об’єктів, складність їхньої геометрії, число джерел світла, наявність фізичних симуляцій та ефектів частинок і вплив цих факторів на продуктивність.

Мета дослідження полягає в оцінці того, як із зростанням навантаження змінюються ключові показники продуктивності (кількість кадрів за секунду, час кадру, завантаження графічного процесорів) і які внутрішні механізми рушіїв (рендеринг, управління пам’яттю, підсистеми фізики тощо) визначають ці зміни.

До об’єкта експерименту входять такі складові:

- ігрові рушії Unity та Unreal Engine — програмні платформи, що виконують рендеринг, фізичні обчислення та управління сценами;
- тестові сцени — однакові за структурою середовища, що містять 3D-об’єкти, джерела світла, текстури та системи частинок;
- інструментальні засоби збору даних — скрипти розроблені для автоматичного вимірювання та логування продуктивності;
- обчислювальні ресурси — апаратна платформа (комп’ютер), на якій проводяться тести: процесор, відеокарта, оперативна пам’ять, система зберігання даних;
- параметри експерименту — кількість об’єктів у сцені, рівень деталізації, тип освітлення, частота оновлення кадру, тощо.

Unity (ігровий рушій)

Unity — це кросплатформний ігровий рушій, розроблений Unity Technologies, вперше анонсований та випущений у червні 2005 року на конференції Apple Worldwide Developers Conference, як ігровий рушій для Mac OS X. З того часу рушій поступово розширювався для підтримки різноманітних настільних, мобільних, консольних платформ, платформ доповненої та віртуальної реальності. Він особливо популярний для розробки мобільних ігор для iOS та Android і вважається простим у використанні для розробників-початківців та популярний для розробки незалежних ігор .

Двигун можна використовувати для створення тривимірних (3D) та двовимірних (2D) ігор, а також інтерактивних симуляцій. Двигун був прийнятий у галузях промисловості поза межами відеоігор, включаючи кіно, автомобілебудування, архітектуру, інженерію, будівництво та Збройні сили Сполучених Штатів .

Використання

Unity надає користувачам можливість створювати ігри та ігровий досвід як у 2D, так і в 3D, а рушій пропонує основний API сценаріїв на C# з використанням Mono, як для редактора Unity у вигляді плагінів, так і для самих ігор, а також функцію перетягування. До того, як C# став основною мовою програмування, що використовувалася для рушія, він підтримував Boo, який було видалено з виходом Unity 5, та реалізацію JavaScript на основі Boo під назвою UnityScript, яка була застаріла у серпні 2017 року, після виходу Unity 2017.1, на користь C#.

У 2D-іграх Unity дозволяє імпортувати спрайти та використовувати розширений рендерер 2D-світу. Для 3D-ігор Unity дозволяє вказувати параметри стиснення текстур, міп-карт та роздільної здатності для кожної платформи, яку підтримує ігровий рушій, а також забезпечує підтримку карт рельєфності, карт відображення, карт паралакса, екранного простору ambient occlusion (SSAO), динамічних тіней за допомогою карт тіней, ефектів рендерингу в текстуру та повноекранної постобробки. Доступні два окремі конвеєри рендерингу : High Definition Render Pipeline (HDRP) та Universal Render Pipeline (URP, раніше LWRP), на додаток до вбудованого конвеєра старої версії. Усі три конвеєри рендерингу несумісні один з одним. Unity пропонує інструмент для оновлення шейдерів за допомогою старого рендерера до URP або HDRP.

Розробники можуть розробляти та продавати користувацькі ресурси іншим розробникам ігор через магазин Unity Asset Store. Це включає 3D- та 2D-ресурси та середовища для розробників, які вони можуть купувати та продавати. Магазин Unity Asset Store було запущено у 2010 році. До 2018 року через цифровий магазин було здійснено приблизно 40 мільйонів завантажень.

Unreal Engine (ігровий рушій)

Unreal Engine (UE) — це 3D-графічний ігровий движок, розроблений Epic Games, спочатку призначений для використання у відеогрі-шутері від першої особи Unreal 1998 року. Спочатку розроблений для шутерів від першої особи для ПК, він з того часу використовується в різних жанрах ігор та був прийнятий іншими галузями, зокрема кіно- та телеіндустрією. Unreal Engine написаний на C++ та має високий ступінь портативності, підтримуючи широкий спектр настільних комп'ютерів, мобільних пристроїв, консолей та платформ віртуальної реальності.

Останнє покоління, Unreal Engine 5, було запущено у квітні 2022 року. Його вихідний код доступний на GitHub, а комерційне використання надається на основі моделі роялті, при цьому Epic стягує 5% від доходу понад 1 мільйон доларів США, що не поширюється на ігри, опубліковані ексклюзивно в Epic Games Store. Epic включила в рушій функції від придбаних компаній, таких як Quixel, яка вважається, отримує вигоду від доходів Fortnite.

1.2 Сфера використання

Відеоігри

Спочатку Unreal Engine був розроблений для використання як базова технологія для відеоігор. Двигун використовується в низці відомих ігор з високими графічними можливостями, включаючи Hogwarts Legacy, PUBG: Battlegrounds, Final Fantasy VII Remake, Valorant та Yoshi's Crafted World, а також в іграх, розроблених Epic, включаючи Gears of War та Fortnite. Польський розробник ігор CD Projekt також планує використовувати рушій після припинення підтримки власного REDengine; їхньою першою грою, яка використовуватиме Unreal, буде ремейк The Witcher.

Використання Unreal Engine неухильно зростає з 2012 року, з приблизно 17% частки ринку до 28% у 2024 році, порівняно з 50% Unity. За обсягом продажів

Unreal становить 31% порівняно з 26% Unity, а власні двигуни складають 42% загалом, що робить Unreal двигуном найбільшим за кількістю проданих одиниць.

Кіно та телебачення

Unreal Engine знайшов застосування у кіновиробництві для створення віртуальних декорацій, які можуть відстежувати рух камери навколо акторів та об'єктів і відображатися в режимі реального часу на великих світлодіодних екранах та системах атмосферного освітлення. Це дозволяє створити композицію кадрів у режимі реального часу, негайне редагування віртуальних декорацій за потреби та можливість знімати кілька сцен протягом короткого періоду, просто змінюючи віртуальний світ позаду акторів. Загальний вигляд було визнано більш природним, ніж типові ефекти хромакей.

Інші способи використання

Unreal Engine також використовувався некреативними сферами завдяки своїй доступності та набору функцій. Він був використаний, як основа для інструменту віртуальної реальності для дослідження молекул фармацевтичних препаратів у співпраці з іншими дослідниками, як віртуальне середовище для дослідження та проектування нових будівель та автомобілів, а також використовувався для кабельних новинних мереж для підтримки графіки в реальному часі. Деякі автомобільні компанії, зокрема Rivian, використовують Unreal Engine у своїх інформаційно-розважальних системах.

Висновки до розділу 1

Дослідження продуктивності ігрових рушіїв **Unity** та **Unreal Engine** має широку сферу застосування у різних галузях, що виходять за межі безпосередньо ігрової індустрії. У контексті розробки комп’ютерних ігор результати дослідження дозволяють розробникам оптимізувати проекти під цільові платформи, балансувати між якістю графіки та швидкодією, а також підвищувати ефективність виробничих процесів. Для індустрії **VR/AR**

дослідження продуктивності забезпечує можливість досягнення стабільної частоти кадрів і мінімальної затримки, що є критичним для зручності та безпеки користувачів.

Таким чином, сфера застосування дослідження охоплює як ігрову індустрію, так і суміжні галузі — від віртуальної та доповненої реальності, до робототехніки й архітектурної візуалізації. Отримані результати мають практичне значення для розробників, науковців, освітніх установ та бізнес-середовища, сприяючи підвищенню якості, продуктивності й ефективності проектів у сфері інтерактивних технологій.

Розділ 2 Обгрунтування методів дослідження

2.1 Основні фунсціональні вимоги

Для забезпечення об’єктивності результатів була розроблена методика тестування, що включає підготовку середовища, сценарій проведення та процедуру збору даних.

Підготовка середовища:

- експерименти проводились на одному комп’ютері з однаковими умовами (графічні налаштування, роздільна здатність екрана, обмеження кадрів вимкнено);
- у кожному рушії (Unity та Unreal Engine) створено ідентичні сцени, що містили набір 3D-об’єктів, джерел світла та елементів фізики;

2.2 Вхідні дані

1. Створюємо ідентичні сцени з об’єктом Plane та одинакові об’екти кулі та складні 3D моделі з симуляцією фізики одинаковою массою та идентичною симуляцією сили тяжіння.

2. При натисканні на кнопку буде створюватись N обектів на невеликій висоті .

3. Проводимо замірювання показників FPS , Frame Time.

4. Запис відео і метрик проводився за допомогою вбудованого програмного забезпечення NVIDIA.

5. Записуємо отримані дані та створюємо графік залежності.

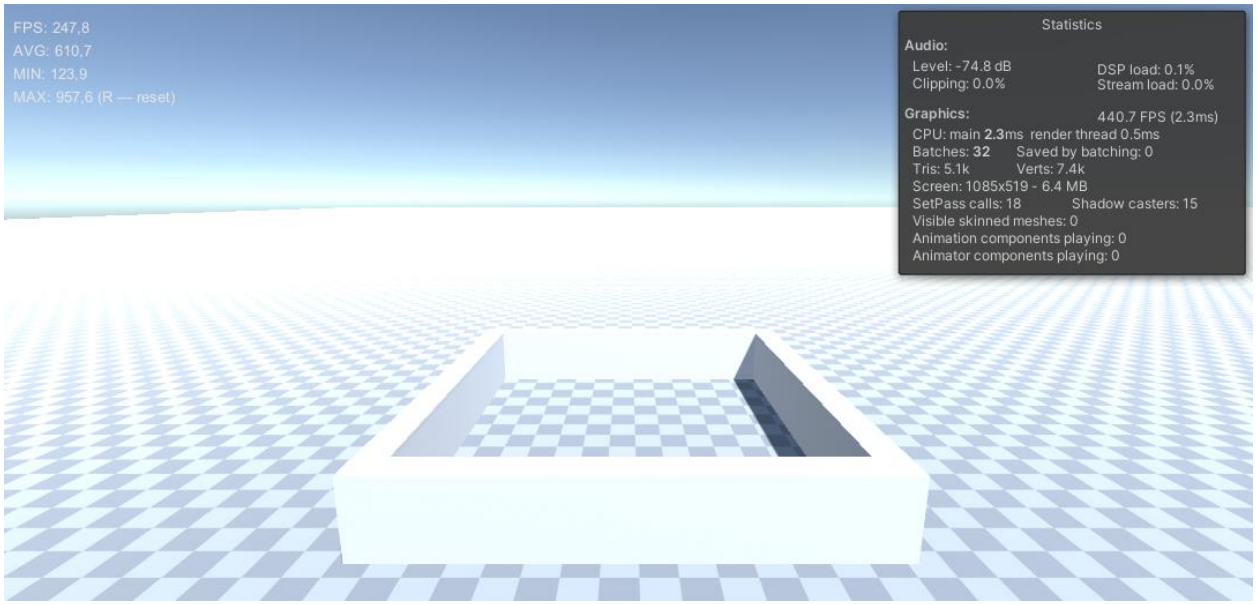

Рисунок 1.2 – Сцена для тестування на Unity

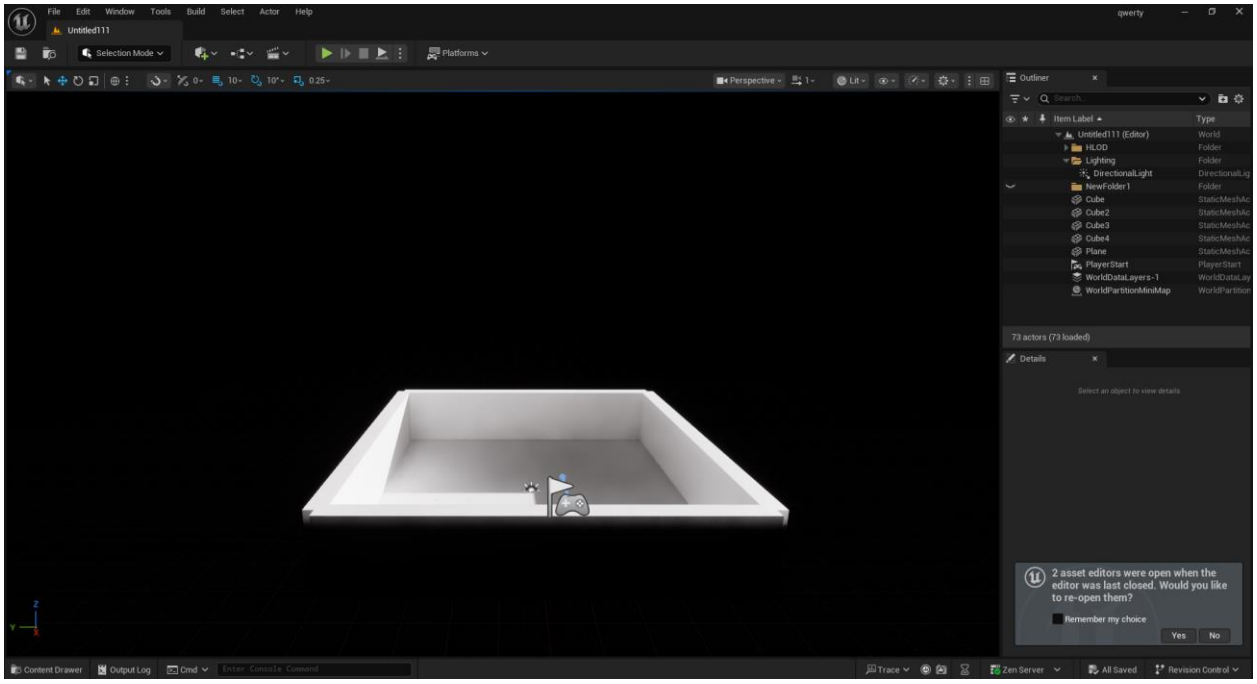

Рисунок 1.2 – Сцена для тестування на Unreal Engine

Для проведення експерименту були створені тестові сцени з мінімальною візуальною складністю, з метою ізоляції впливу кількості об’єктів та фізичних розрахунків на продуктивність рушіїв.

Сцени побудовані з використанням базових примітивів:

- Plane використовувався як підлога для розміщення та взаємодії об’єктів;
- чотири куби застосовувалися як стінки, що обмежують простір та забезпечують коректну фізичну симуляцію зіткнень.

Такий підхід дозволив:

- зменшити вплив графічних ефектів на результати;
- зосередитися на навантаженні від рендерингу та фізики об’єктів;
- забезпечити однакові умови тестування для Unity та Unreal Engine.

Усі сцени мали ідентичну структуру та налаштування, що гарантує коректність і порівнюваність отриманих результатів.

Та в подальших експериментах буде використовуватись базова 3D модель мавпи з програми Blender для перевірки різниці навантаження рушіїв з використанням багатополігональних моделей.

Рисунок 2 – 3D модель мавпи з 3936 полігонами (Suzanne)

2.3 Вихідні дані

Очікувані результати

Розроблені інструментальні засоби дають можливість:

- отримувати точні кількісні показники продуктивності;
- порівнювати рушії у рівних умовах;
- виявляти слабкі місця у роботі Unity та Unreal Engine;
- візуалізувати результати у зручній формі для подальшого аналізу.

Таким чином, створений інструментальний комплекс є основою для проведення серії експериментів і подальшого формування висновків щодо ефективності використання обох рушіїв у розробці ігор.

Висновки до розділу 2

Для досягнення поставленої мети дослідження було створено дві ідентичні сцени в ігрових рушіях Unity та Unreal Engine 5, що забезпечило однакові умови проведення експериментів. Це дозволило об’єктивно порівняти вплив кількості об’єктів і використання фізики на продуктивність, стабільність часу кадру та якість відображення.

Проведені експерименти дадуть змогу виявити особливості роботи кожного рушія та визначити їх сильні й слабкі сторони. На основі отриманих результатів буде сформовано обґрунтовані висновки щодо того, у яких сценаріях кожен рушій демонструє кращу продуктивність і стабільність, а також у яких умовах потребує додаткової оптимізації.

Розділ 3 Розробка інструментальних засобів для дослідження рушіїв Unity та Unreal Engine

3.1 Постановка задачі дослідження

Основною задачею дослідження є порівняння продуктивності ігрових рушіїв Unity та Unreal Engine при виконанні однакових обчислювальних і графічних операцій у стандартизованих умовах.

Для досягнення поставленої мети необхідно розв’язати такі підзадачі:

1. Розробити тестові сцени, які мають однакову структуру, набір об’єктів, освітлення та фізичні параметри.
2. Створити інструментальні засоби збору та фіксації показників продуктивності обох рушіїв.
3. Провести серію експериментів із поступовим збільшенням навантаження на систему.
4. Зібрати та систематизувати дані про FPS, використання процесора, графічного процесора.
5. Порівняти отримані результати та зробити висновки про ефективність кожного рушія в різних умовах.

У процесі виконання роботи робиться акцент на практичному вимірюванні продуктивності й аналізі стабільності кадру при зміні кількості об’єктів у сцені.

3.2 Методология фіксації експеримента

Фіксація та запис експериментів

Запис і фіксація результатів експерименту здійснюються за допомогою вбудованого програмного забезпечення NVIDIA

(NVIDIA GeForce Experience), що дозволяє:

- автоматично фіксувати показники FPS у реальному часі;
- записувати відео тестових сесій без втрати продуктивності;
- отримувати звіти про продуктивність у вигляді таблиць або графіків;

- використання інструментів NVIDIA забезпечує однакові умови вимірювання для обох рушіїв, а також високу точність фіксації даних, оскільки результати не залежать від внутрішніх профайлерів кожного рушія.

Методологія дослідження

Методологія дослідження продуктивності ігрових рушіїв Unity та Unreal Engine передбачає використання порівняльного аналізу на основі експериментального підходу.

Вибір об’єктів дослідження:

- ігрові рушії: Unity та Unreal Engine;
- тестові проекти (ідентичні сцени у двох рушіях);
- показники продуктивності: FPS, час рендерингу кадру, GPU.

Створення експериментальних умов:

- формування однакових тестових сцен у обох рушіях із базовими геометричними об’єктами, матеріалами, джерелами освітлення та елементами фізики;
- використання однакового апаратного забезпечення (процесор, оперативна пам’ять, відеокарта);
- збереження ідентичних налаштувань графіки (роздільна здатність, якість текстур, тіні, ефекти).

Операційна система: Windows 10 з актуальними оновленнями, включаючи оновлені драйвери пристроїв зберігання та вводу-виводу, встановлені через вбудований компонент оновлення Windows.

Відеокарта Nvidia GTX 1660 оновлено актуальним пакетом, включаючи оновлені драйвери пристроїв зберігання та вводу-виводу, інстальовані через вбудований компонент оновлення.

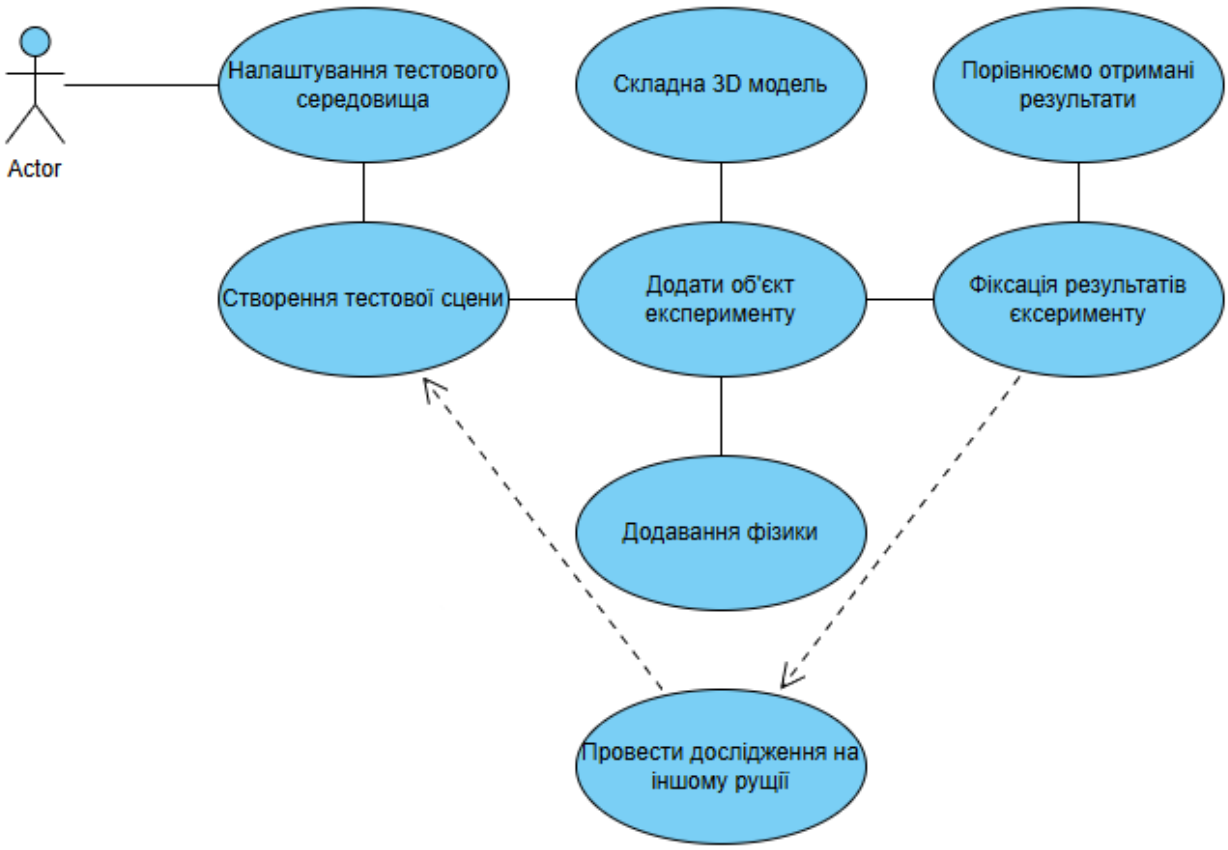

Рисунок 3.1 – Діаграма прецендентів аналізу та проведення досліджень

Розробку інструментальних засобів для проведення досліджень

Для проведення об’єктивного порівняльного аналізу продуктивності ігрових рушіїв Unity та Unreal Engine необхідно створити комплекс інструментальних засобів, які забезпечують збір, фіксацію та аналіз технічних показників у контрольованих умовах.

Такі засоби дають змогу автоматизувати процес тестування, уникнути впливу суб’єктивних чинників і гарантувати однакові умови для обох рушіїв.

Загальна концепція інструментальних засобів

Основною ідеєю розробки інструментів є створення єдиного сценарію тестування, який можна реалізувати як у Unity, так і в Unreal Engine, із застосуванням однакових сцен, об’єктів і параметрів камери.

Це забезпечує повну відтворюваність експериментів та достовірність результатів.

Інструментальний комплекс складається з кількох взаємопов’язаних модулів:

- модуль створення тестової сцени;
- модуль збору даних (профайлер);
- модуль запису результатів;
- модуль візуалізації та аналізу.

2. Модуль створення тестової сцени

Тестова сцена створюється окремо в кожному рушії але з ідентичними умовами:

- однакове розташування камери, освітлення, джерел світла та текстур;
- набір стандартних 3D-об’єктів (куби, сфери, моделі середньої полігональності);
- механізм поступового збільшення кількості об’єктів у сцені для створення контрольованого навантаження;
- наявність анімацій та фізичних взаємодій для перевірки роботи фізичного рушія.

Таке рішення дозволяє дослідити, як кожен рушій реагує на збільшення кількості елементів і складності сцени.

До тестової сцени також відносится створення Level Blueprint (вбудована в Unreal Engine) для проведення етапів екстпериментів

3. Модуль збору даних

Модуль збору даних відповідає за фіксацію ключових показників продуктивності під час виконання тесту.

До таких показників належать:

- FPS (кадри за секунду);
- Frame Time (середній час обробки кадру);
- кількість полігонів та активних об’єктів.

У Unity та Unreal Engine дані отримуються за допомогою вбудованих інструментів та через консольні команди

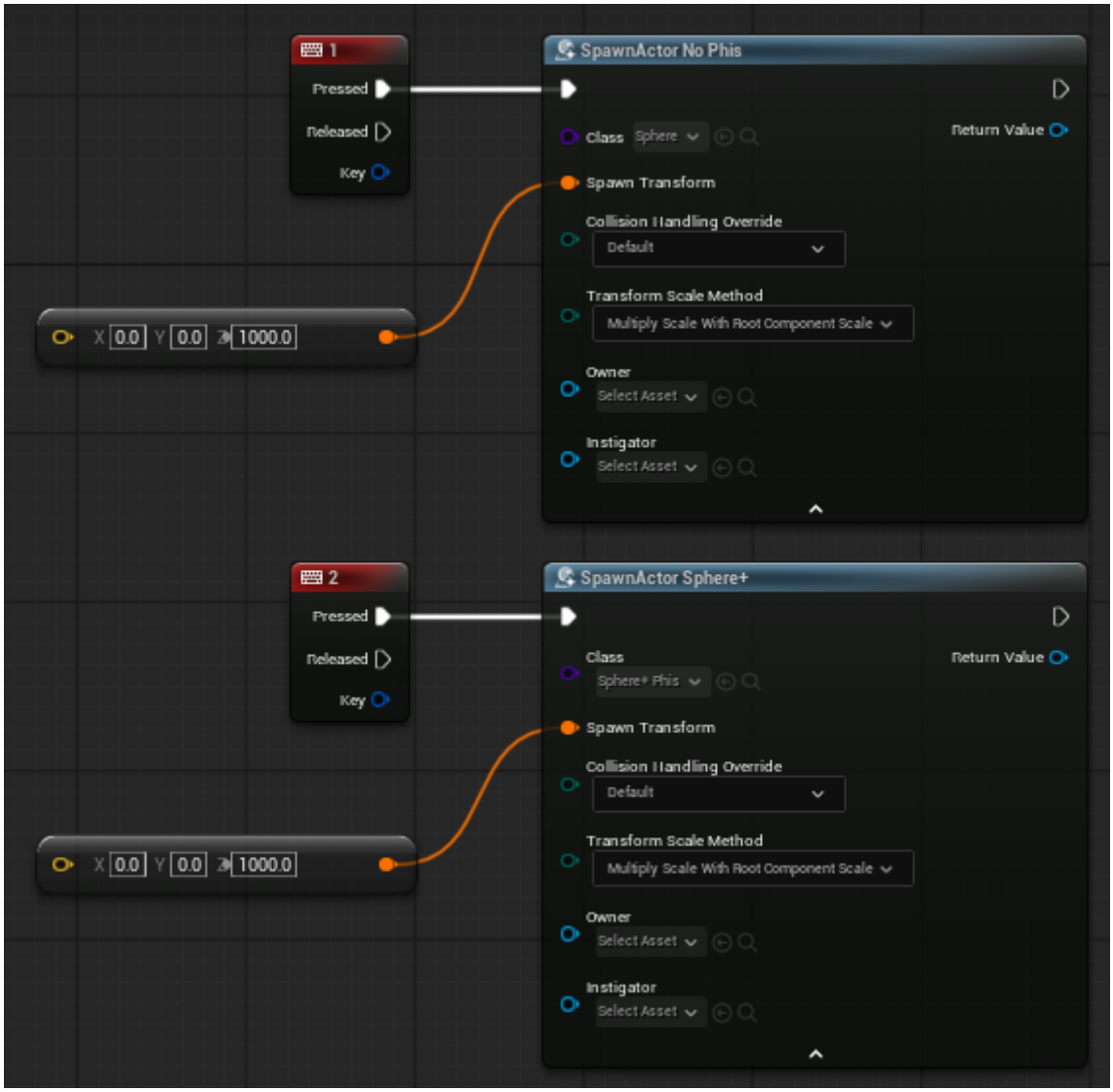

Рисунок 3.2 – Bluprint створення примитивів з фізикою та без

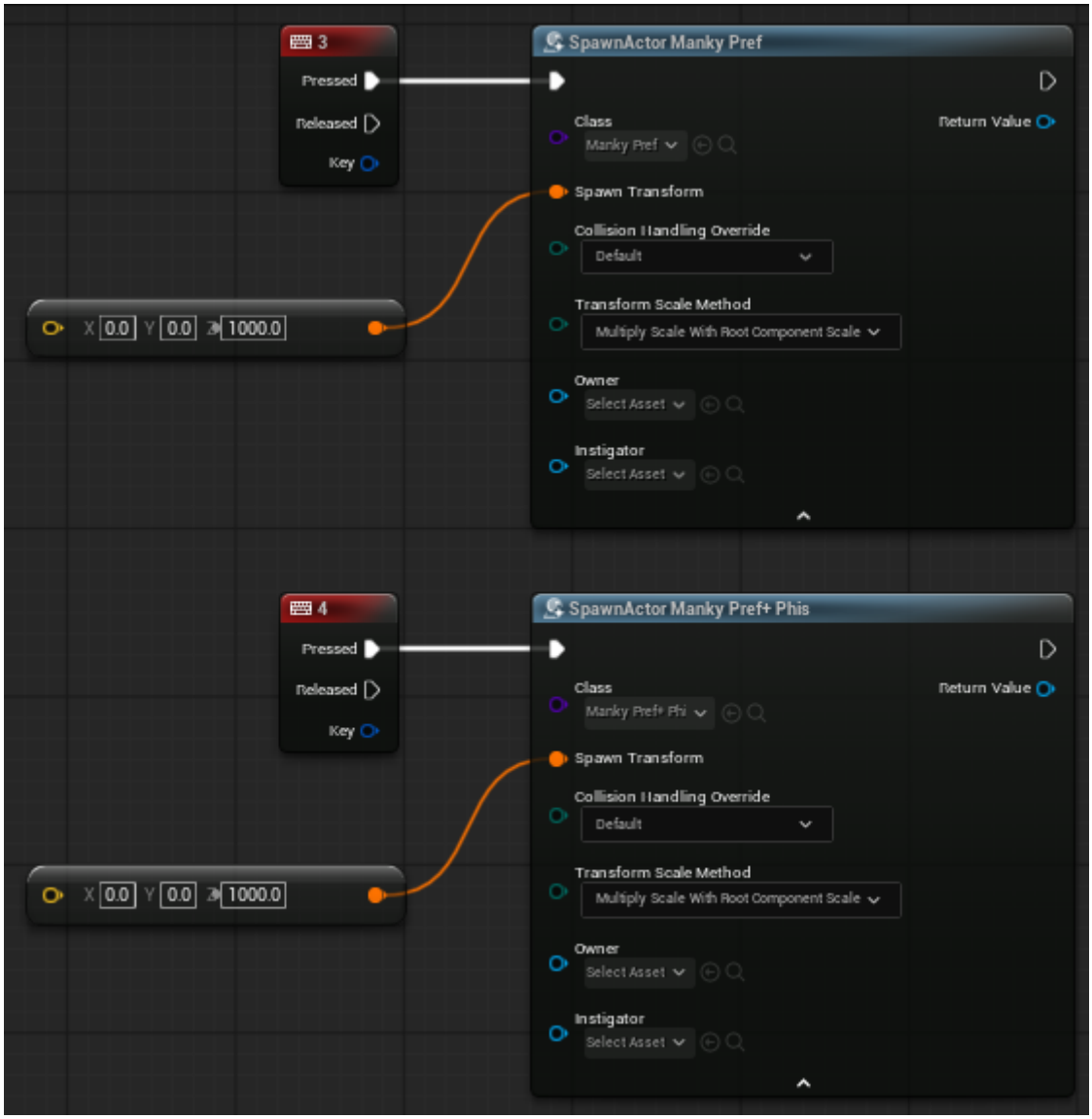

Рисунок 3.3 - Bluprint створення складних 3D моделей з фізикою та без

4. Модуль запису та фіксації результатів

Для забезпечення максимальної точності запису даних використовується вбудоване програмне забезпечення NVIDIA

(зокрема NVIDIA GeForce Experience, NVIDIA Performance Overlay або FrameView).

Ці інструменти дозволяють:

- фіксувати FPS, часу кадру;
- записувати відео тестової сесії без впливу на продуктивність;
- автоматично експортувати результати у CSV або JSON для подальшого аналізу;

- створювати графічні звіти з динамікою змін показників.

Завдяки цьому дані для Unity та Unreal Engine збираються в однакових умовах і за допомогою одного апаратно-програмного комплексу, що гарантує їх об’єктивність.

5. Модуль візуалізації та аналізу

Після завершення тестів результати обробляються за допомогою Excel для побудови графіків залежності FPS, часу кадру та споживання ресурсів від кількості об’єктів у сцені.

Також розраховуються середні, мінімальні та максимальні значення показників, що дозволяє виявити стабільність рушія під навантаженням.

Висновок до розділу 3

Обрані методи дослідження є доцільними та обґрунтованими, оскільки забезпечують об’єктивне порівняння продуктивності ігрових рушіїв Unity та Unreal Engine в однакових умовах. Використання ідентичних тестових сцен, поступового збільшення кількості об’єктів і розділення експериментів на сценарії з фізикою та без фізики дозволило ізолювати вплив окремих чинників на продуктивність і виключити сторонні фактори. Таким чином, обрана методика дослідження дозволила коректно виявити сильні та слабкі сторони кожного рушія.

Розділ 4 Аналіз продуктивності рушіїв Unity та Unreal Engine

4.1 Експеримент №1 (створення великої кількості однакових об’єктів з використанням примітивів)

Хронологія проведення експерименту

1. Базовий замір (сцена без об’єктів)

1. Створити новий проект в Unity та Unreal Engine.

2. Створити порожню сцену (мінімум об’єктів — лише камера та освітлення за замовчуванням).

3. Встановити однакові налаштування графіки (роздільна здатність, якість тіней, постобробка тощо).

4. Запустити сцену в режимі відтворення за допомогою вбудованих функцій, фіксуємо FPS.

5. Повторити замір 3–5 разів і обчислити середнє значення (для точності результатів).

Мета етапу: визначити “нульовий” рівень продуктивності — як рушії працюють без навантаження.

2. Створення об’єктів і замір середнього FPS

1. Додати на сцену однакові об’єкти .

2. Розташувати їх рівномірно, щоб усі об’єкти потрапляли в камеру.

3. Увімкнути ідентичні джерела світла та матеріали в обох рушіях.

4. Запустити сцену.

5. Виміряти:

- середній FPS;
- мінімальний FPS;
- пікове використання ресурсів.

6. Зафіксувати результати в таблиці.

7. Повторити такі ж дії в другому рушії.

Мета етапу: перевірити, як збільшення кількості об’єктів впливає на продуктивність кожного рушія.

Дані, отримані під час тестування, були зафіксовані для сцен без додавання об’єктам компонентів Rigidbody та без увімкнення фізичних обчислень.

Отримати результати, щодо навантаження рушіїв додаванням великої кількості примітивних не фізичних об’єктів.

Таблиця 1 – Зібрані дані середнього FPS на Unity з створенням примітивів

	MAX FPS	MID FPS	MIN FPS
0	1044	795	222
1000	625	443	210
2000	469	400	161
3000	363	307	138
4000	286	237	140
5000	246	206	138

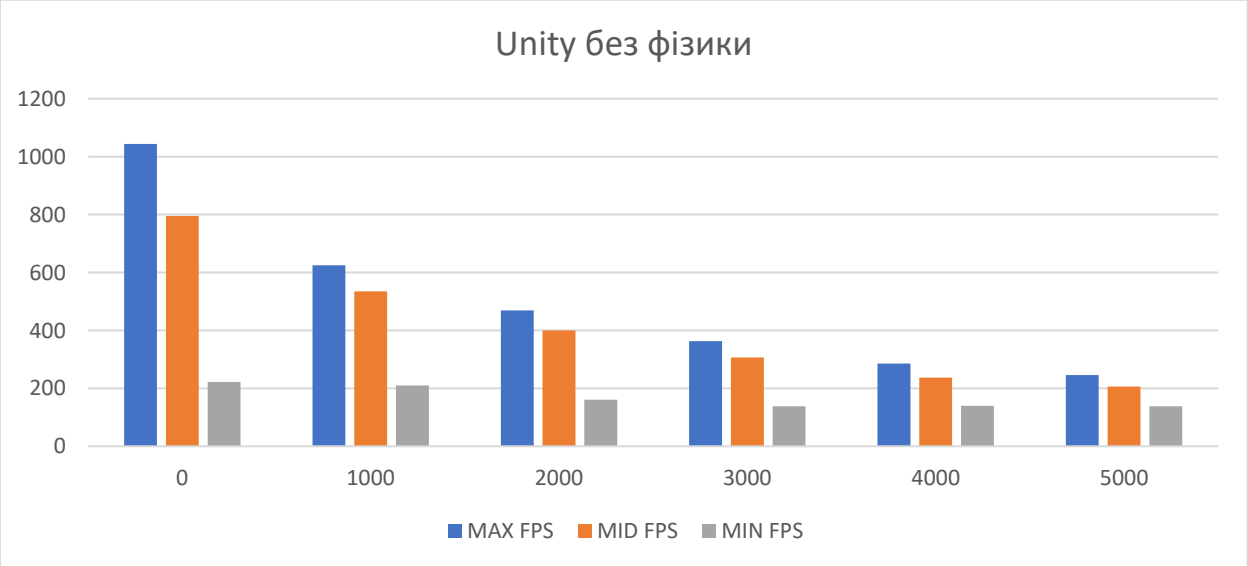

Рисунок 4.1 – Діаграма отриманих даних до експеримента № 1 на Unity

Діаграма демонструє, що Unity при додаванні кожних 1000 об’єктів показує поступове зниження FPS приблизно на 100 кадрів. Спад відбувається рівномірно, що свідчить про передбачувану продуктивність рушія під навантаженням.

Проте одночасне додавання великої кількості об’єктів викликає різкий тимчасовий спад мінімального піку FPS. Це свідчить про те, що Unity потребує додаткових ресурсів для миттєвої обробки великого числа об’єктів, хоча середній FPS залишається стабільним у динаміці зростання навантаження.

Таким чином, рушій показує стабільну продуктивність при поступовому збільшенні об’єктів, але одночасне масове додавання може призводити до короткочасних просідань FPS, які необхідно враховувати при плануванні сцени.

Таблиця 2 - Зібрані дані середнього FPS на Unreal Engine з створення примітивів

	MAX FPS	MID FPS	MIN FPS
0	90	88,5	87
1000	87	86,5	86
2000	87	85,5	84
3000	85	83	81
4000	80	78	76
5000	75	72	69

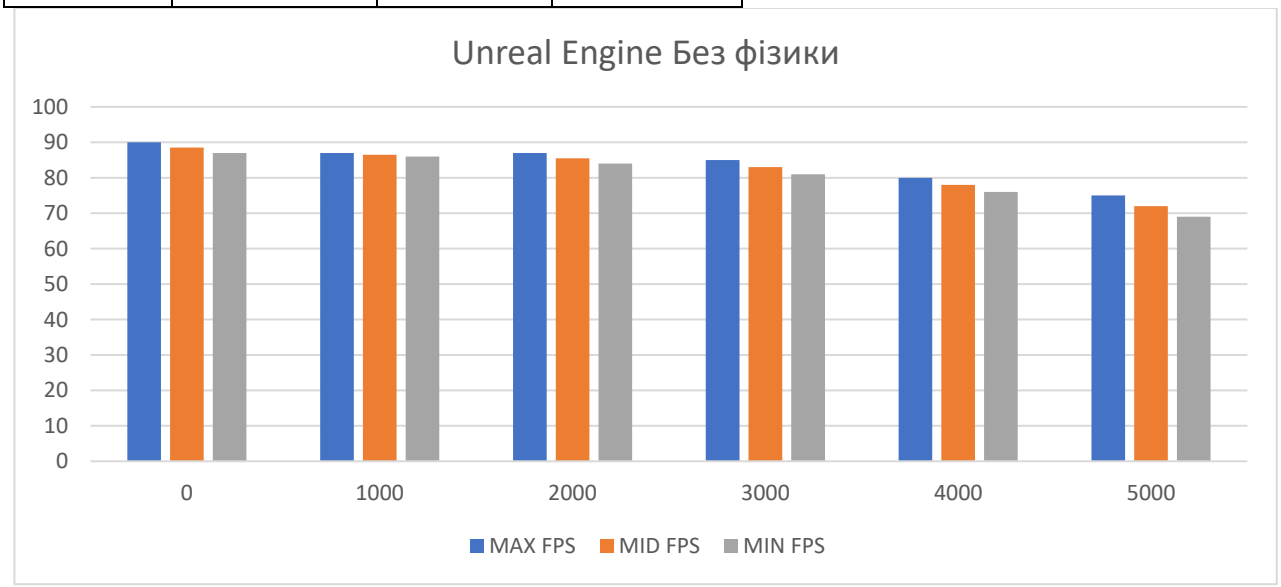

Рисунок 4.2 – Діаграма отриманих даних до експеримента № 1 на Unreal Engine

Діаграма демонструє, що Unreal Engine показує лише незначне зниження FPS при збільшенні кількості об’єктів: від 88,5 до 72 кадрів. Це свідчить про ефективну обробку великої кількості даних і стабільну продуктивність рушія навіть під значним навантаженням.

Таким чином, Unreal Engine забезпечує плавний та передбачуваний спад FPS, що робить його надійним для сцен з великою кількістю об’єктів.

Тафлиця 3 – Порівняння часу кадру екпериментy №1

	Unity ms	Unreal Engine ms
0	1,2	10,4
1000	1,9	11,6
2000	2,6	12,4
3000	3,5	13,7
4000	4,6	14,2
5000	5,4	14,9

Аналіз отриманих даних показує, що обидва ігрові рушії демонструють поступове та відносно рівномірне зростання часу кадру (ms) при збільшенні навантаження.

Unity має менші значення часу кадру в абсолютних одиницях: від 1,2 ms до 5,4 ms. Це відповідає більш високому FPS у порівнянні з Unreal Engine.

Unreal Engine показує більші значення часу кадру: від 10,4 ms до 14,9 ms, що є природним через інший підхід рушія до рендерингу та управління ресурсами.

Незважаючи на різницю в абсолютних показниках, обидва рушії демонструють однакову тенденцію росту часу кадру: із збільшенням навантаження час кадру зростає плавно та передбачувано, без різких стрибків. Це свідчить про стабільність обох рушіїв при додаванні великої кількості об’єктів, навіть якщо їхня продуктивність у ms різниться.

Таким чином, Unity може забезпечувати вищий FPS при легкому навантаженні, тоді як Unreal Engine підтримує стабільну роботу на трохи

більших значеннях часу кадру, що також є прийнятним для більшості ігрових сценаріїв. Обидва рушії показують надійність і прогнозовану поведінку при зростанні кількості об’єктів.

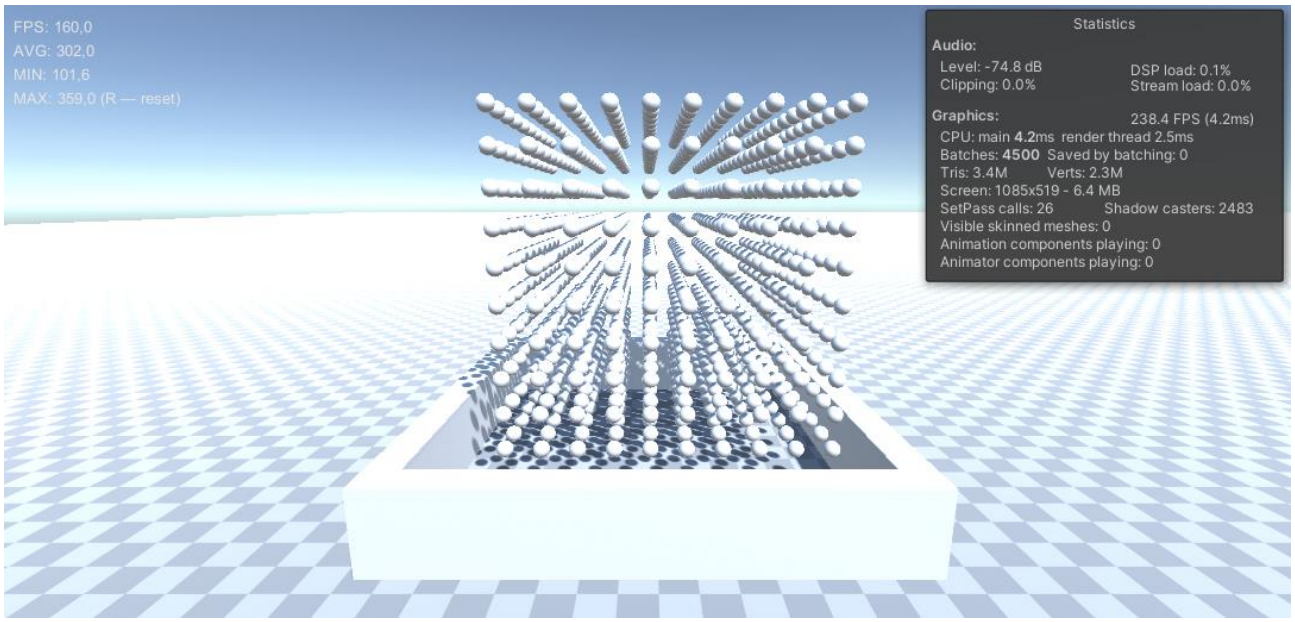

Рисунок 4.3 – 1000 простих об’єктів на Unity

Unity — 1000 об’єктів

Створення перших 1000 об’єктів без фізики супроводжується короткочасними просадками FPS до ~0,3. Однак ці просадки тривають лише в момент одночасного створення об’єктів і швидко зникають: після 1–2 секунд FPS стабілізується й повертається до середнього значення. Такий ефект пов’язаний з одноразовою ініціалізацією моделей, а не з постійним навантаженням сцени.

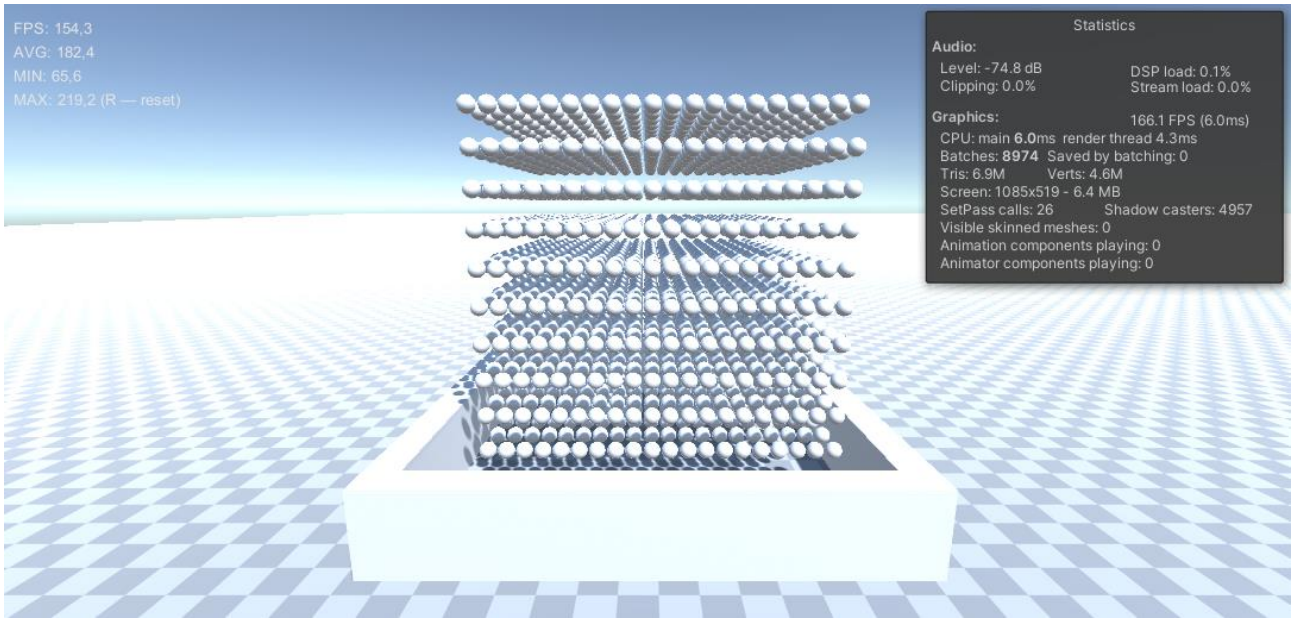

Рисунок 4.4 – 2000 простих об’єктів на Unity

Unity — 2000 об’єктів

При збільшенні сцени до 2000 об’єктів середній FPS помітно знижується, однак рушій зберігає стабільність. Просадки короткочасні, без графічних артефактів. Основне навантаження переходить на CPU, що відповідає за обробку великої кількості об’єктів у кадрі.

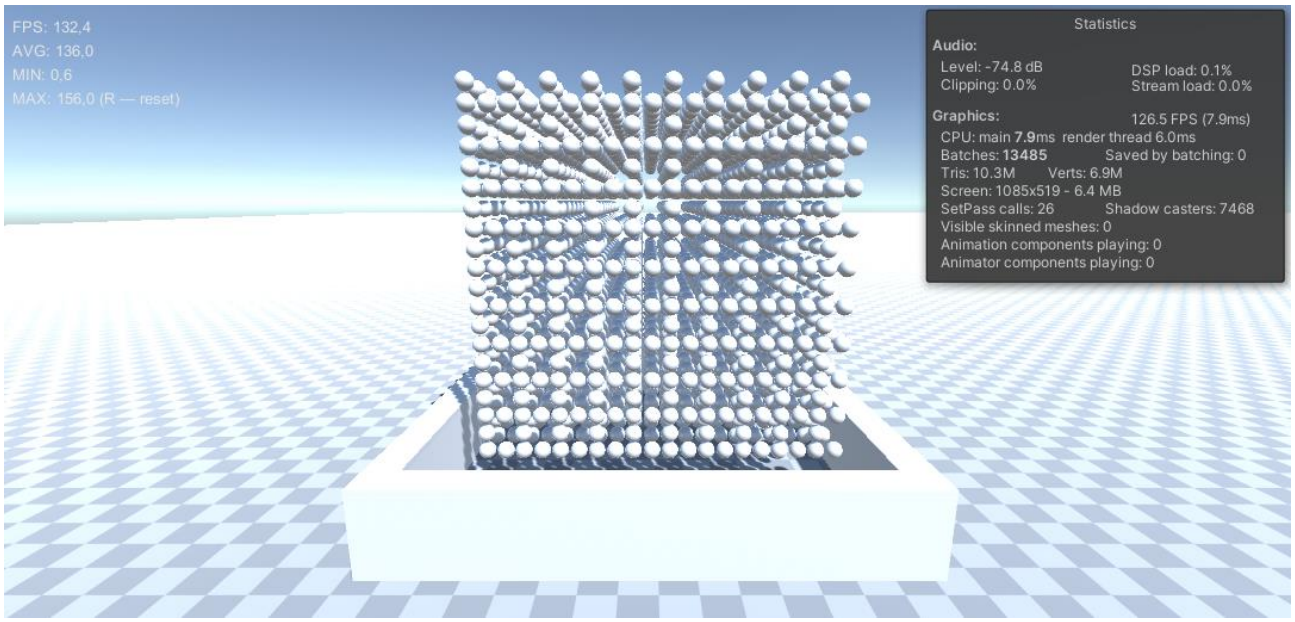

Рисунок 4.5 – 3000 простих об’єктів на Unity

Unity — 3000 об’єктів

На рівні 3000 об’єктів середній FPS продовжує поступово падати, а пікові просадки стають частішими. Незважаючи на це, картинка залишається стабільною — рендеринг не втрачає якості. На цьому етапі дедалі більше помітний вплив кількості об’єктів на час оновлення сцени.

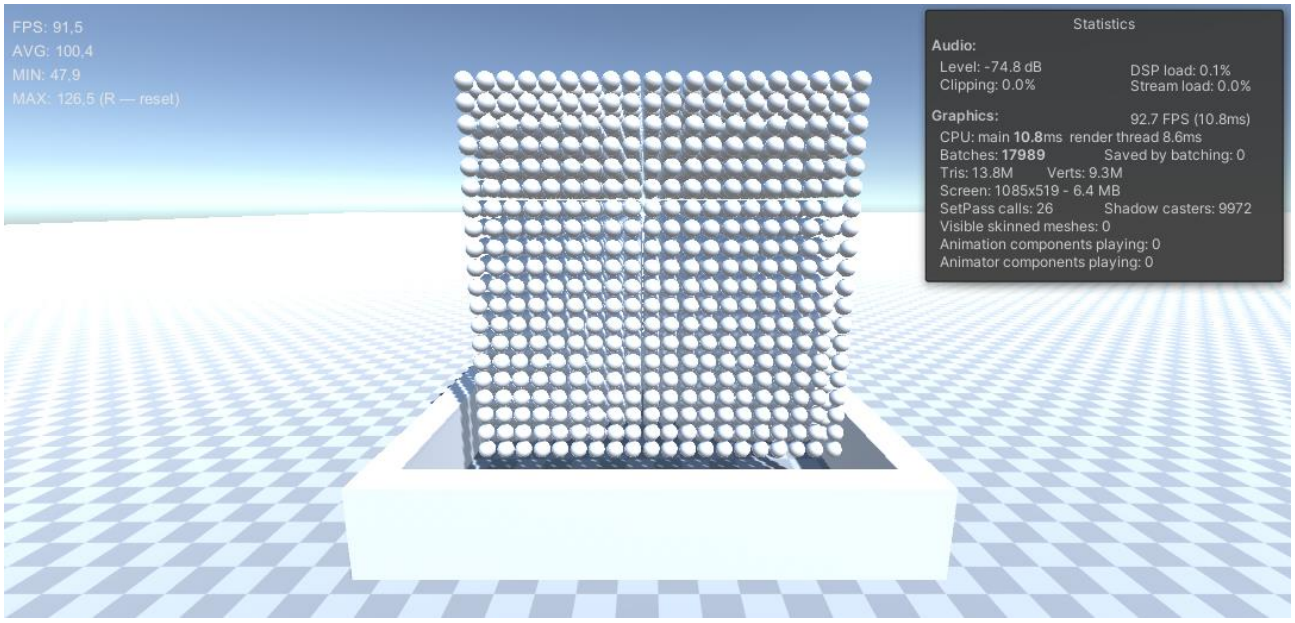

Рисунок 4.6 – 4000 простих об’єктів на Unity

Unity — 4000 об’єктів

При 4000 об’єктах продуктивність досягає межі, де навіть невеликі зміни у сцені починають відчутно впливати на плавність. Середній FPS знижується, а мінімальні значення частіше падають. Це демонструє, що без оптимізації (батчинг, LOD, відключення зайвих компонентів) масштабування сцени стає менш ефективним.

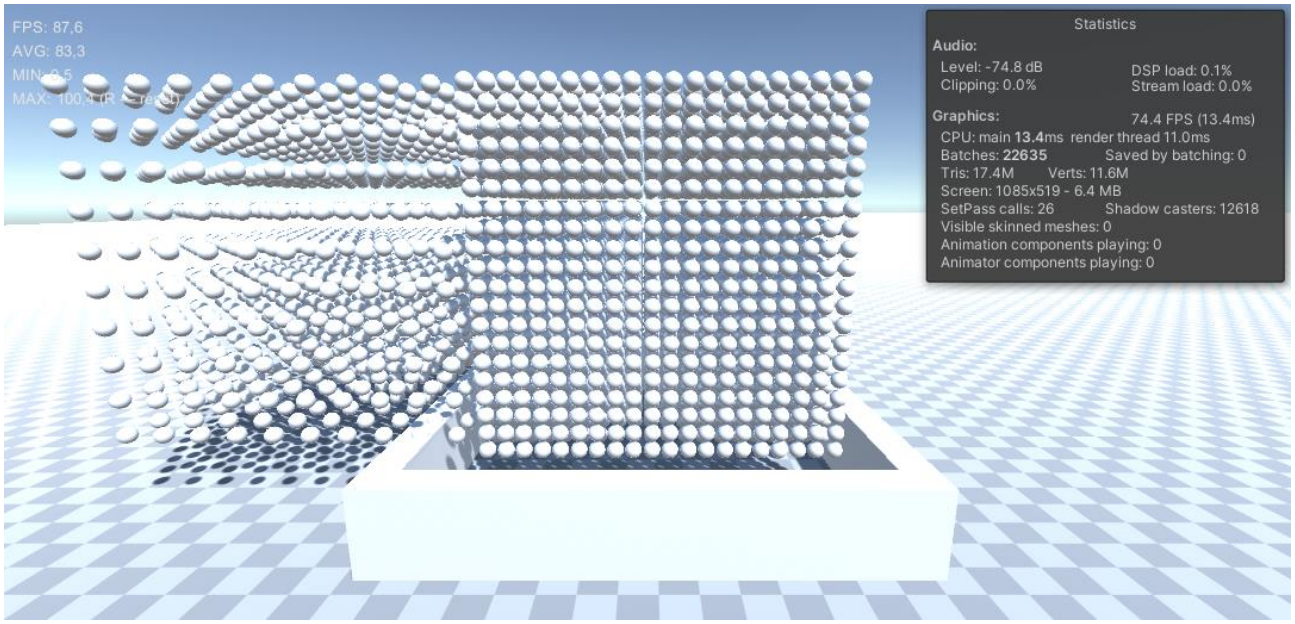

Рисунок 4.7 – 5000 простих об’єктів на Unity

Unity — 5000 об’єктів

На 5000 об’єктах спостерігається найбільший тиск на систему: середній FPS знижується ще більше, а окремі кадри можуть короткочасно збільшувати час рендерингу. Водночас рушій продовжує коректно відображати сцену — проблем з геометрією та матеріалами не виникає. Це підкреслює, що головним обмеженням стає не графіка, а кількість об’єктів, які необхідно обробляти.

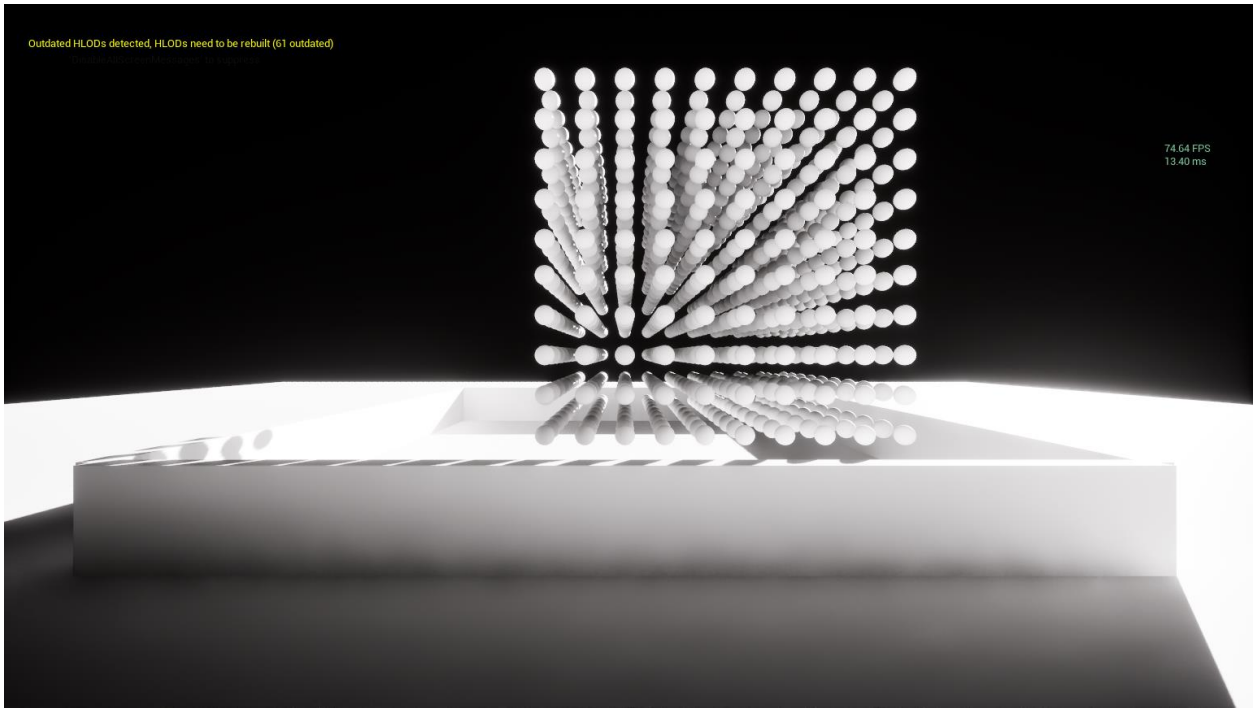

Рисунок 4.8 – 1000 простих об’єктів на Unreal Engine

Unreal Engine 1000 об’єктів

За наявності 1000 об’єктів Unreal Engine показує приблизно 86 FPS (≈11 ms).

Зображення залишається повністю стабільним: тіні, освітлення та матеріали опрацьовуються без артефактів. Навіть при різкому додаванні об’єктів просідань майже не видно — сцена виглядає плавною та керованою.

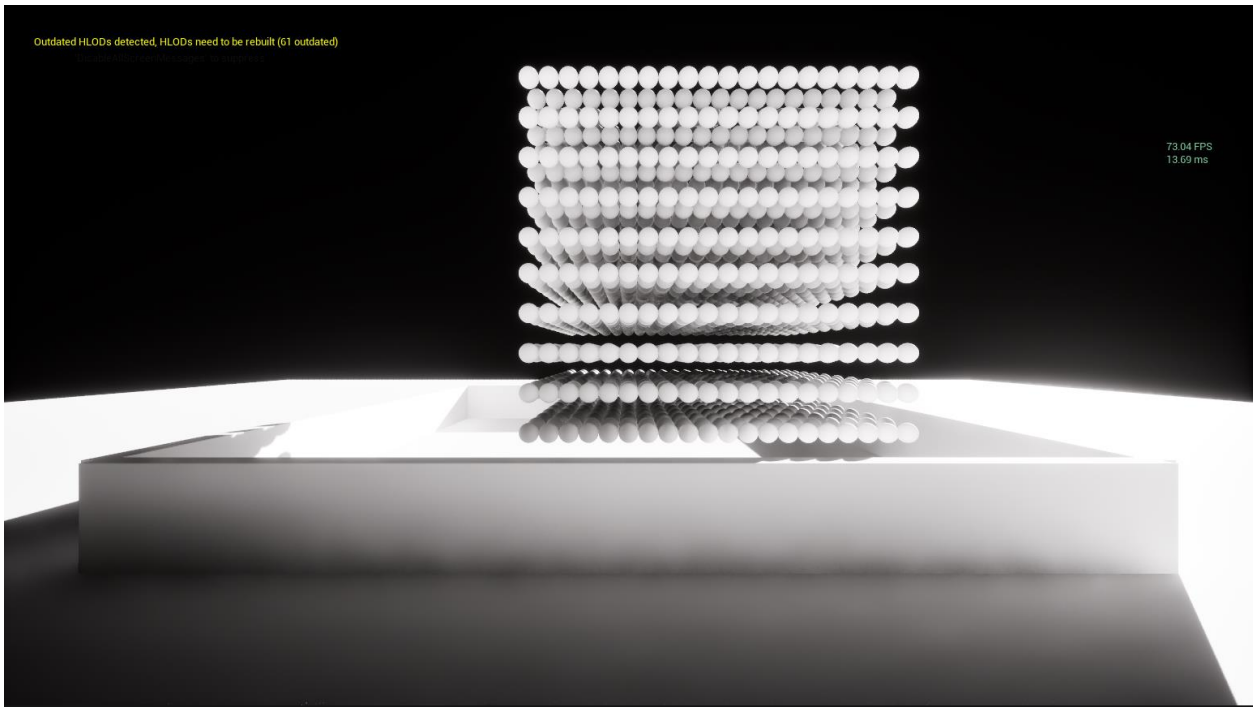

Рисунок 4.9 – 2000 простих об’єктів на Unreal Engine

Unreal Engine 2000 об’єктів

При збільшенні до 2000 об’єктів FPS знижується лише до близько 85 FPS (≈12 ms).

Зміни практично непомітні — рух камери й анімації залишаються плавними. Це показує, що рушій добре масштабує навантаження та рівномірно розподіляє обчислення.

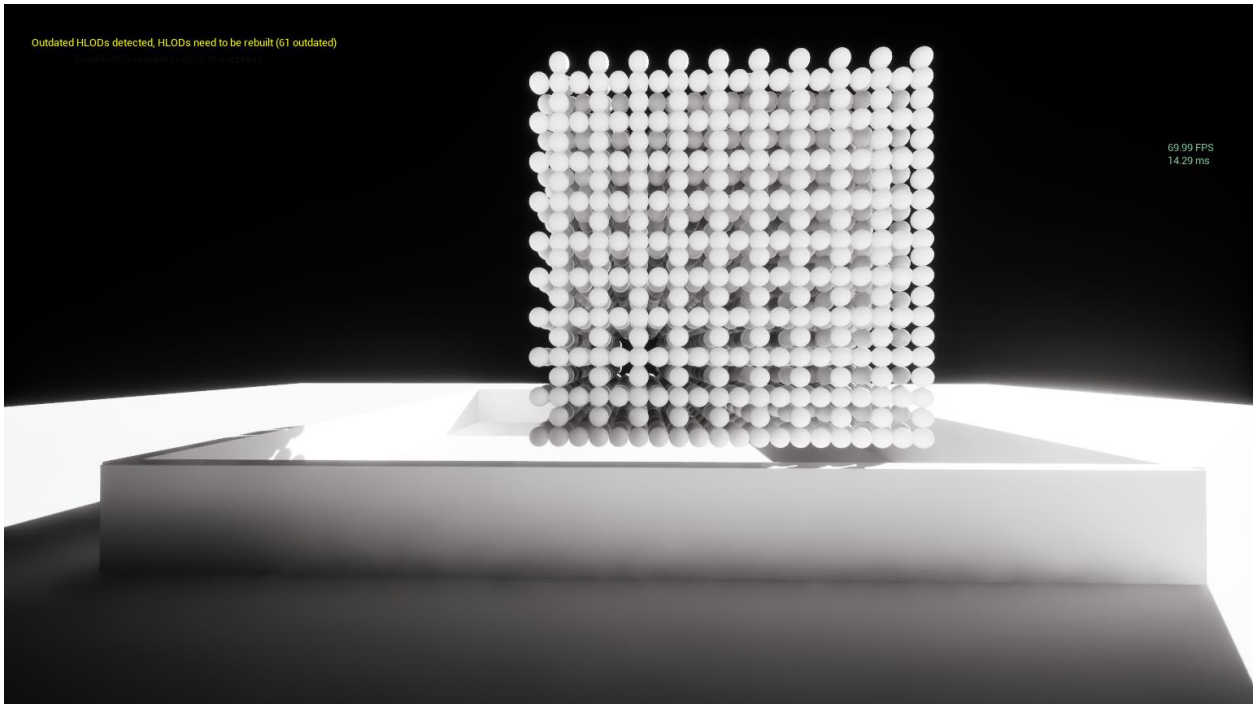

Рисунок 4.10 – 3000 простих об’єктів на Unreal Engine

Unreal Engine 3000 об’єктів

На 3000 об’єктах продуктивність знижується до ≈83 FPS (≈13 ms).

Спад поступовий, без «стрибків». Якість картинки не погіршується, а графік FPS залишається стабільним. Unreal продовжує працювати впевнено навіть при великій кількості елементів у сцені.

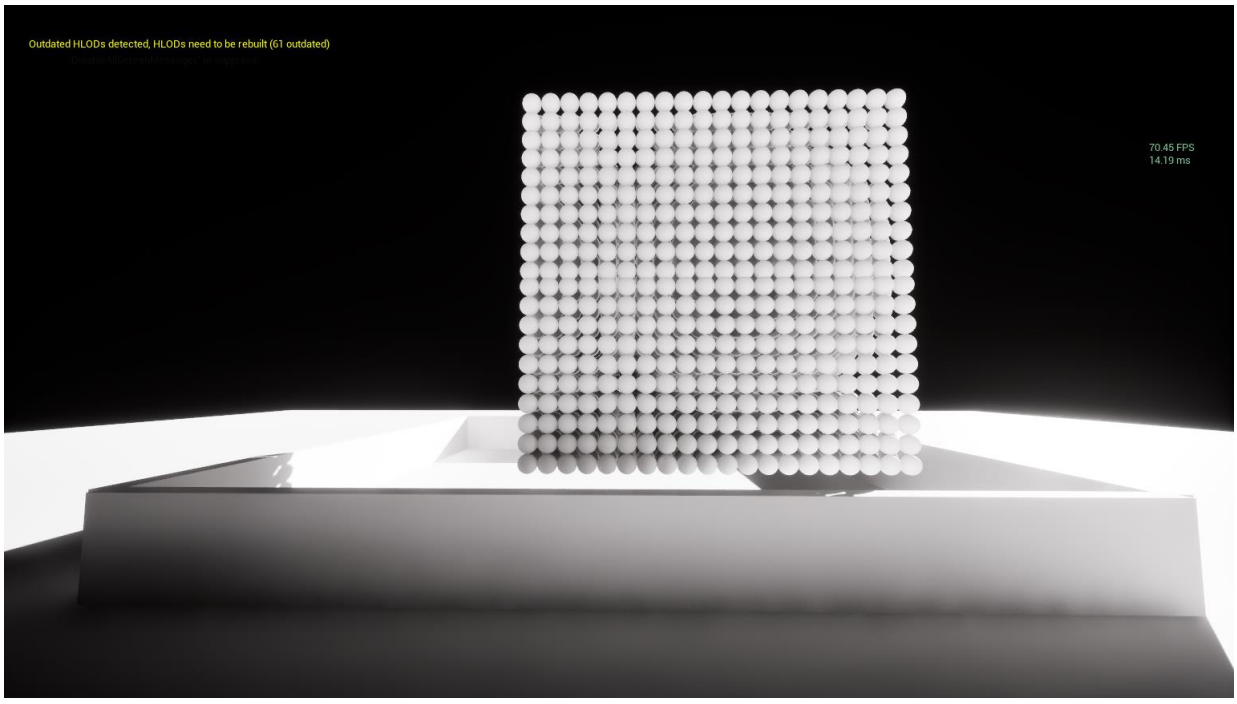

Рисунок 4.11 – 4000 простих об’єктів на Unreal Engine

Unreal Engine 4000 об’єктів

При 4000 об’єктах FPS опускається приблизно до 78 FPS (≈14 ms).

Навантаження стає відчутнішим, але кадри все ще рівномірні. Рушій утримує баланс між якістю та продуктивністю, не створюючи різких фризів або візуальних збоїв.

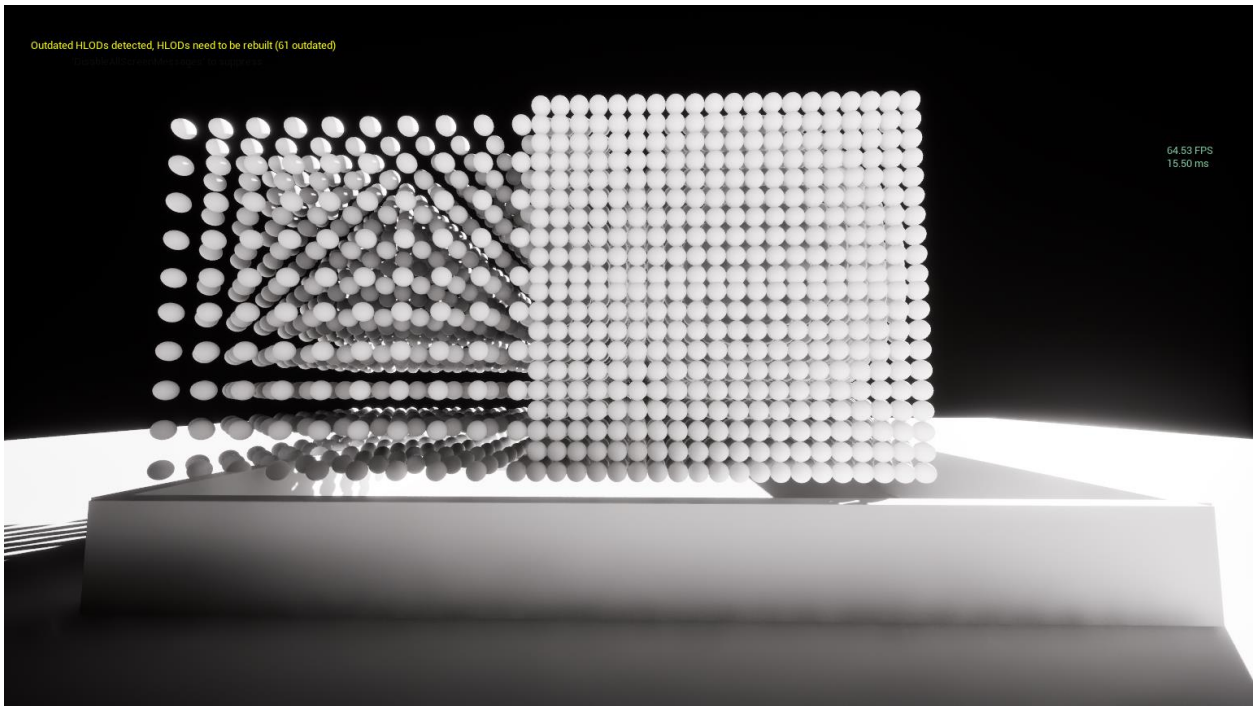

Рисунок 4.12 – 5000 простих об’єктів на Unreal Engine

Unreal Engine 5000 об’єктів

На 5000 об’єктах продуктивність знижується до близько 72 FPS (≈14,9 ms).

Попри значну кількість моделей, сцена залишається плавною. Спад FPS помітний, але він поступовий, без різких провалів — рушій продовжує контролювати навантаження.

Висновки до експерименту №1

Отримані результати експерименту показують, що Unreal Engine краще переносить зростання навантаження та демонструє більш стабільний рівень продуктивності. Навіть при збільшенні кількості об’єктів до 1000, зниження частоти кадрів становило лише близько 3 FPS, що свідчить про ефективні механізми оптимізації рендерингу та управління ресурсами. Водночас було помічено, що на початкових етапах роботи Unreal Engine має дещо нижчий базовий FPS, проте подальше навантаження впливає на нього мінімально.

Результати для Unity виявили протилежну тенденцію. Рушій демонструє дуже високий початковий FPS (приблизно 800 кадрів/с), однак із додаванням об’єктів його продуктивність значно погіршується: у середньому відбувається втрата близько 50 FPS на кожні 1000 об’єктів. Крім того, показники виявилися нестабільними: в межах однієї сцени FPS міг змінюватися від 1044 до 222, що негативно впливає на плавність і передбачуваність роботи проекту.

Таким чином, можна зробити висновок, що Unreal Engine є більш придатним для проектів з великою кількістю об’єктів та високим навантаженням, тоді як Unity доцільніше використовувати у випадках, коли важлива висока початкова швидкодія та проекти не потребують значної кількості складних сценових елементів.

4.2 Експеримент №2 (створення великої кількості однакових об’єктів з симуляцією фізики та зіткнень)

Хронологія проведення експерименту

1. Базовий замір (сцена без об’єктів, але з увімкненою фізичною системою)

1. Створити новий проект у Unity та Unreal Engine.
2. Створити порожню сцену (камера + базове освітлення).
3. Переконатися, що:
   - фізика активна;
   - гравітація ввімкнена;
   - параметри симуляції за замовчуванням.
4. Запустити сцену в режимі відтворення.
5. Зафіксувати:
   - FPS;
   - час рендерингу кадру (ms);
   - використання ресурсів.
6. Повторити замір 3–5 разів і взяти середнє значення.

Мета етапу: визначити продуктивність рушіїв, коли фізична система активна, але ще немає об’єктів для обчислень.

2. Додавання об’єктів з увімкненою фізикою

(об’єкти — примітиви: куб/сфера, з колайдером та Rigidbody/Physics Body)

1. Додати на сцену групу об’єктів (1000 → 2000 → 3000 → 4000 → 5000).
2. Для кожного об’єкта:
   - додати колайдер;
   - увімкнути фізику та гравітацію;
   - переконатися, що вони можуть взаємодіяти та стикатися.

3. Розташувати об’єкти так, щоб вони:
    - падали вниз;
    - взаємодіяли між собою.
4. Запустити сцену.

3. Проведення вимірювань

Для кожної кількості об’єктів зафіксувати:

- середній FPS;
- мінімальний FPS;
- ms за кадр.
- характер сцени:
    - чи з’являються фризи / стоп-кадри;
    - коли починають накопичуватися зіткнення;
    - як швидко рушій стабілізує сцену.

Дані занести в таблицю.

4. Повтор у другому рушії

1. Повторити абсолютно ідентичні кроки в іншому рушії.
2. Слідкувати, щоб:
    - сцена була максимально однаковою;
    - фізичні параметри були наближені;
    - об’єкти мали подібну масу і форму.

Мета етапу: отримати порівняння поведінки фізичних систем.

5. Аналіз результатів

1. Порівняти графіки FPS та ms.
2. Звернути увагу:
    - коли починається різке падіння продуктивності;

- зафіксувати як рушії поводяться при масових зіткненнях;
- чи стабілізується сцена після падіння об’єктів.

3. Зробити висновок, як фізика впливає на роботу рушіїв.

Метою даного експерименту є дослідити, як увімкнення фізики, гравітації та масових зіткнень впливає на продуктивність ігрових рушіїв Unity та Unreal Engine.

Таблицця 4 - Зібрані дані середнього FPS на Unity з створення примітивів з фізикою

	MAX FPS	MID FPS	MIN FPS
0	1035	776	129
1000	622	475	93
2000	476	376	90
3000	349	269	35
4000	286	240	93
5000	248	190	54

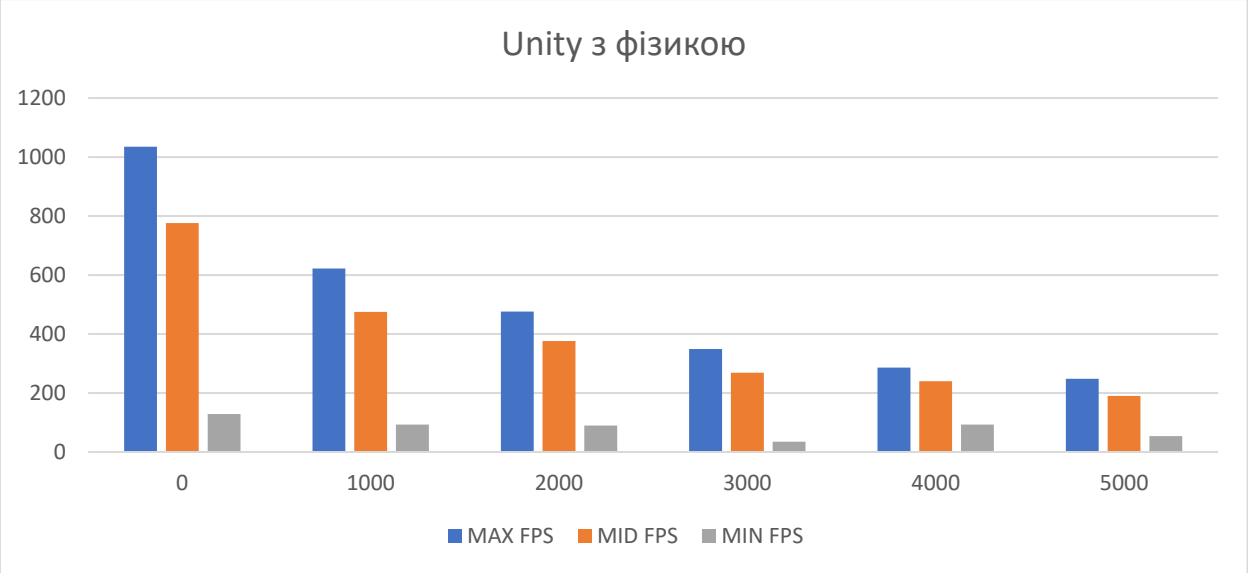

Рисунок 5.1 - Діаграма отриманих данимх до експеримента № 2 на Unity

На цьому графіку, що відображає роботу Unity з увімкненою фізикою, спостерігається плавний і рівномірний спад продуктивності зі збільшенням

кількості об’єктів. Зменшення FPS та зростання часу рендерингу відбуваються поступово, без різких стрибків. Це свідчить про те, що рушій стабільно масштабує фізичні обчислення, хоча підвищення навантаження все ж поступово знижує швидкодію сцени.

Таблиця 5 - Зібрані дані середнього FPS на Unity з створення примітивів з фізикою

	MAX FPS	MID FPS	MIN FPS
0	90	88,5	87
1000	53	51	49
2000	25	24	23
3000	16	15,5	15
4000	11	10,5	10
5000	9,00	8,5	8

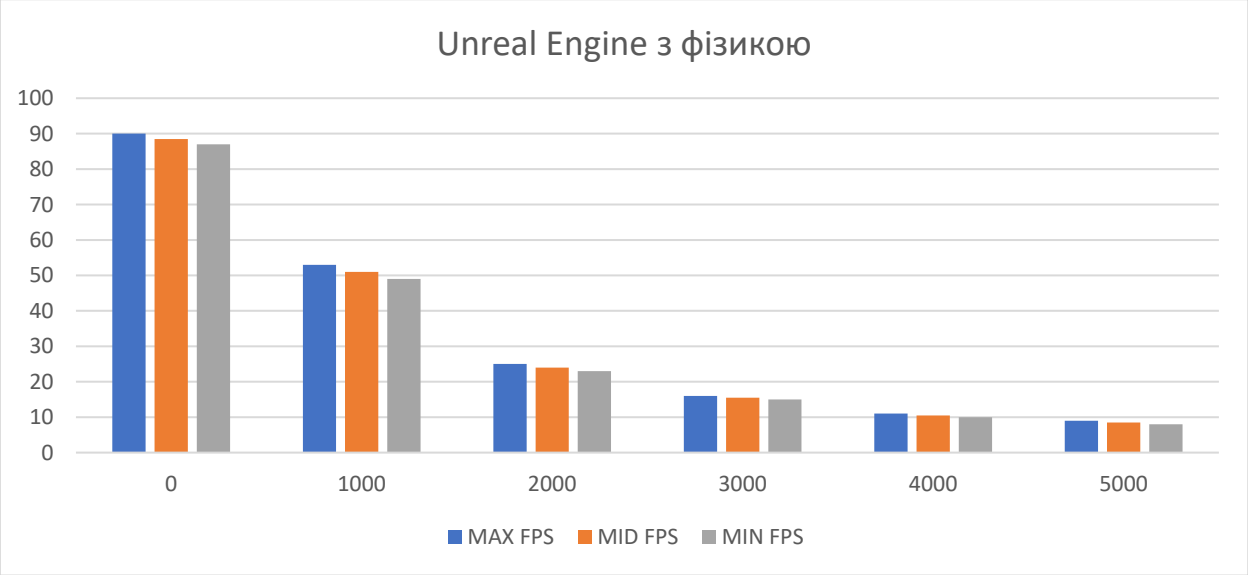

Рисунок 5.2 - Діаграма отриманих даних до експеримента № 2 на Unreal Engine

На цьому графіку, що демонструє роботу Unreal Engine 5 з увімкненою фізикою, спостерігається виражений експоненціальний спад продуктивності. Із збільшенням кількості об’єктів FPS знижується не поступово, а прискорено:

кожне наступне додавання об’єктів створює значно більше навантаження, ніж попереднє. Це свідчить про те, що при великій кількості фізичних взаємодій рушій швидко досягає межі обчислювальних можливостей, що призводить до різких просідань та втрати плавності зображення.

Таблиця 6 – Порівняння часу кадру екперименту №2

	Unity ms	Unreal Engine ms
0	1,2	10,7
1000	2,1	22,3
2000	3,2	43,2
3000	4,3	67,6
4000	9,7	94,9
5000	15,7	124,2

Отримані результати замірів часу кадру (ms) свідчать про різну поведінку Unity та Unreal Engine під навантаженням фізичних об’єктів.

- Unity демонструє плавне збільшення часу кадру до приблизно 3000 об’єктів, після чого спостерігається експоненціальне зростання. Це свідчить, що рушій ефективно обробляє фізику невеликої та середньої кількості об’єктів, але при великому навантаженні продуктивність різко падає.
- Unreal Engine спочатку показує пропорційне та швидке зростання часу кадру вже при додаванні перших 1000–2000 об’єктів. Це вказує на те, що Unreal Engine 5 менш оптимізований для одночасної обробки великої кількості фізичних об’єктів у даному сценарії.

Загалом, Unity краще масштабується при середньому навантаженні, тоді як Unreal Engine демонструє стабільне, але швидке зростання ms навіть на початкових етапах, що слід враховувати при проектуванні сцен з великою кількістю фізики.

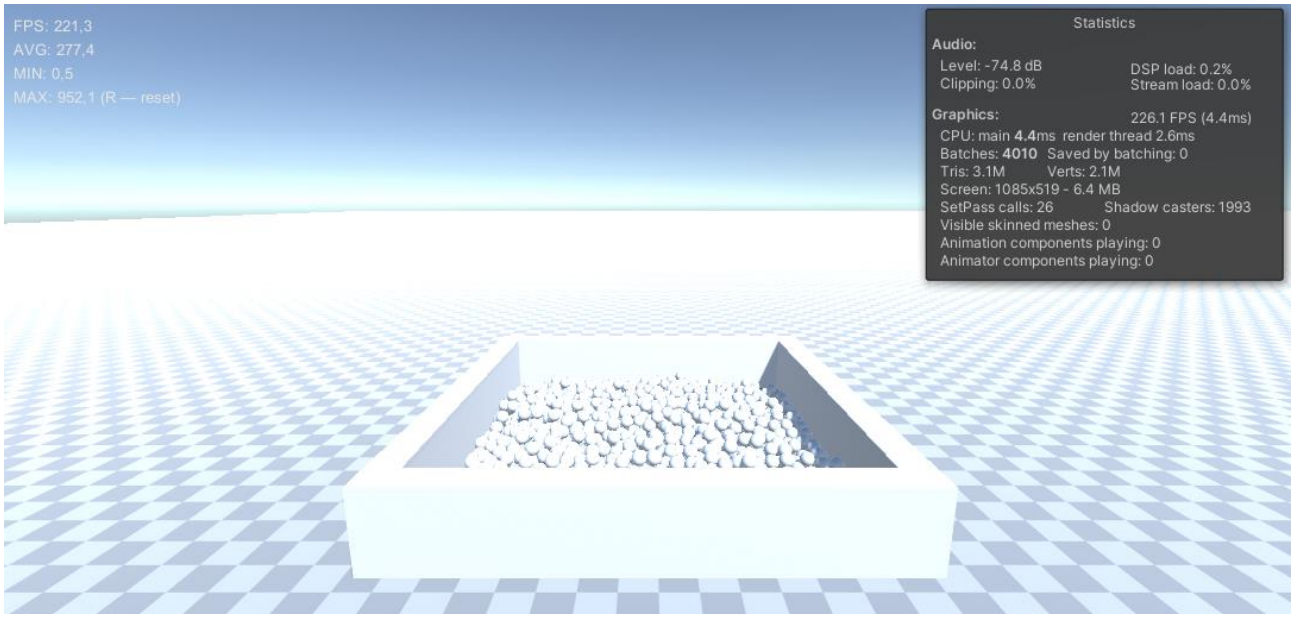

Рисунок 5.3 – 1000 об’єктів з фізикою на Unity

1000 об’єктів з фізикою Unity

При 1000 об’єктах Unity утримує стабільну роботу сцени. FPS зменшується, але залишається комфортним для перегляду. Фізика починає активно опрацьовувати зіткнення, проте просадки короткочасні, без фризів. Час рендерингу кадру зростає, але збільшення відчувається лише на графіках, а не візуально.

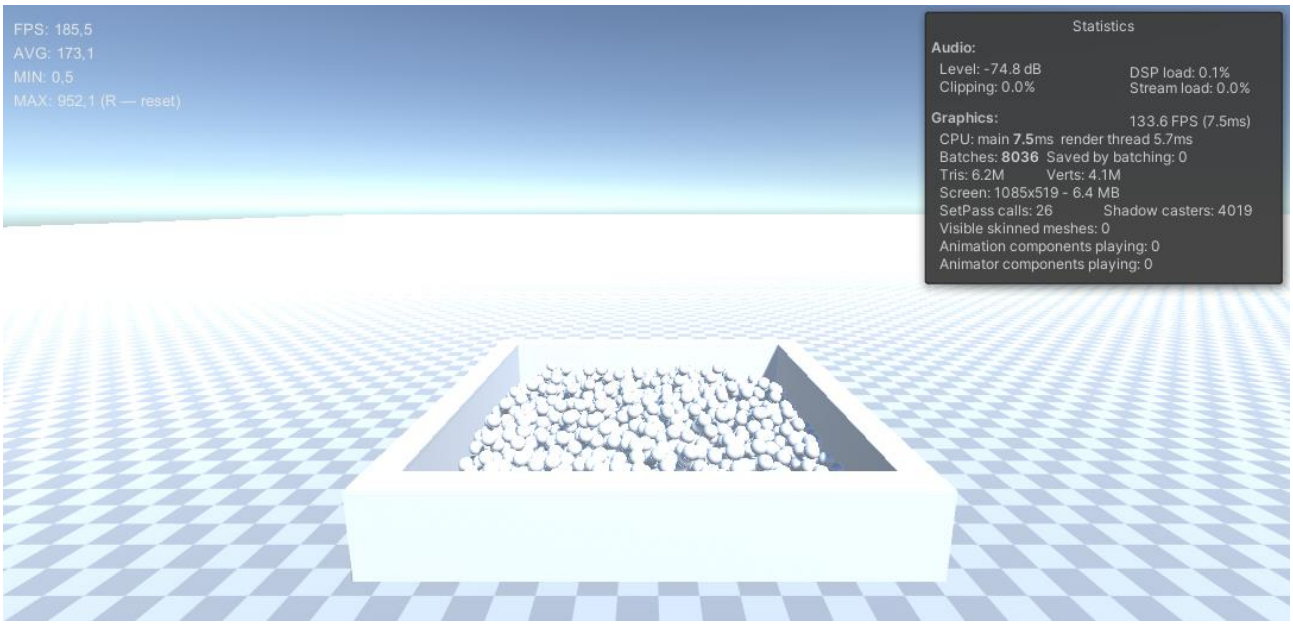

Рисунок 5.4 – 2000 об’єктів з фізикою на Unity

2000 об’єктів з фізикою Unity

На 2000 об’єктах продуктивність помітно знижується, однак спад залишається рівномірним. Рендеринг і фізика працюють синхронно: кадри стають повільнішими, але стабільними. Сцена все ще відтворюється плавно, хоча реакція на дії камери дещо сповільнюється.

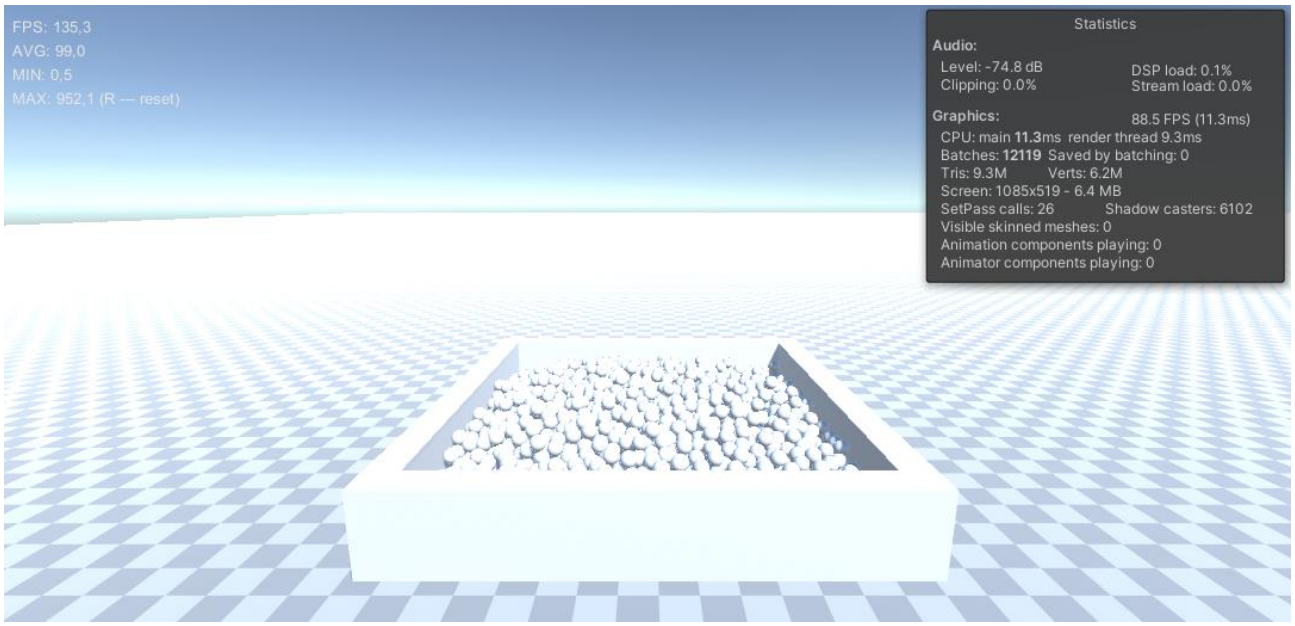

Рисунок 5.5 – 3000 об’єктів з фізикою на Unity

3000 об’єктів з фізикою Unity

При 3000 об’єктах навантаження на систему зростає значно. FPS знижується до середніх значень, а мінімальні піки стають помітнішими. Зіткнення між об’єктами створюють додаткові обчислення, через що час кадру (ms) зростає. Незважаючи на це, сцена залишається керованою й без різких стрибків продуктивності.

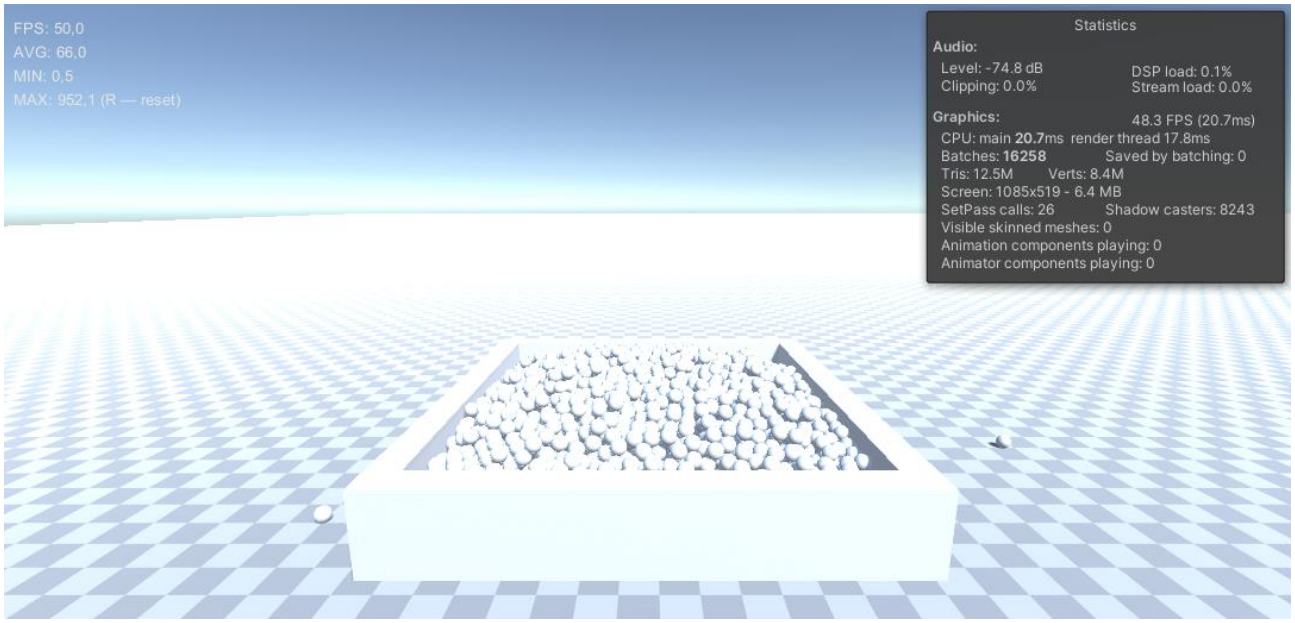

Рисунок 5.6 – 4000 об’єктів з фізикою на Unity

4000 об’єктів з фізикою Unity

На 4000 об’єктах фізика починає відчутно впливати на стабільність. FPS продовжує зменшуватись, а ms збільшується швидше, ніж на попередніх етапах. У моменти масових зіткнень з’являються короткі мікрофризи, але вони не критичні. Загалом спад усе ще плавний, без обвалів продуктивності.

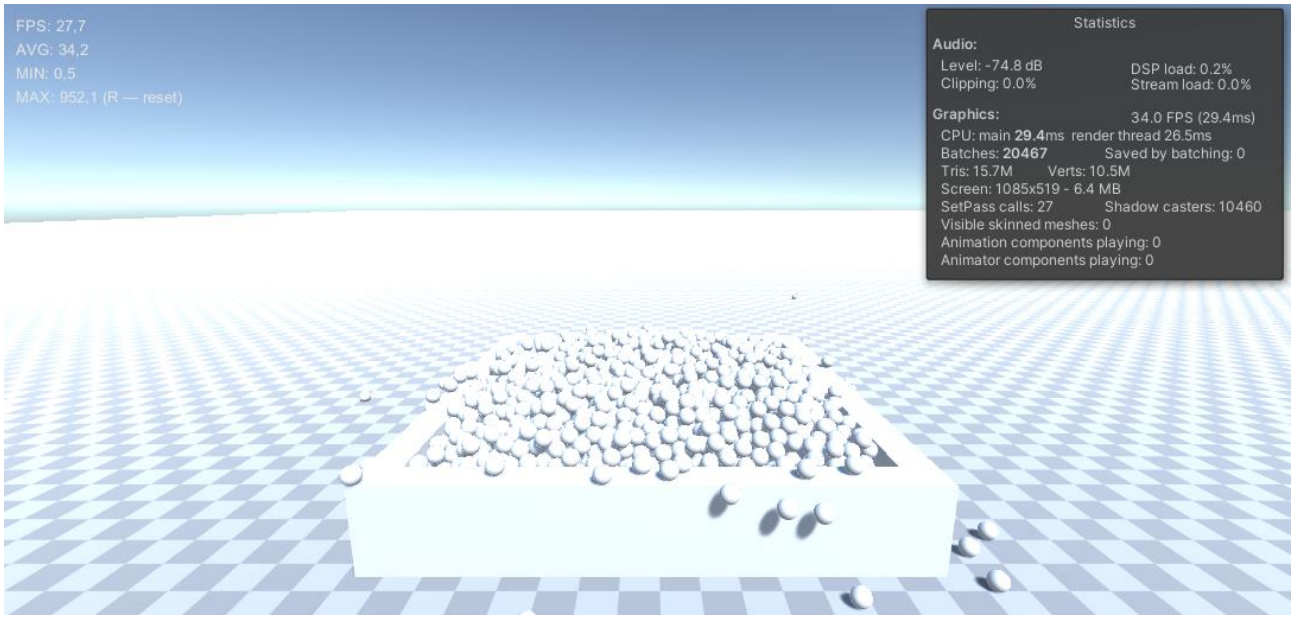

Рисунок 5.7 – 5000 об’єктів з фізикою на Unity

5000 об’єктів з фізикою Unity

При 5000 об’єктах система працює на межі. Середній FPS суттєво нижчий, а мінімальні значення іноді падають . Час кадру стає нестабільнішим через велику кількість одночасних зіткнень. Проте навіть на цьому рівні Unity зберігає відносно передбачуваний характер навантаження — спад поступовий і логічний, а не різкий.

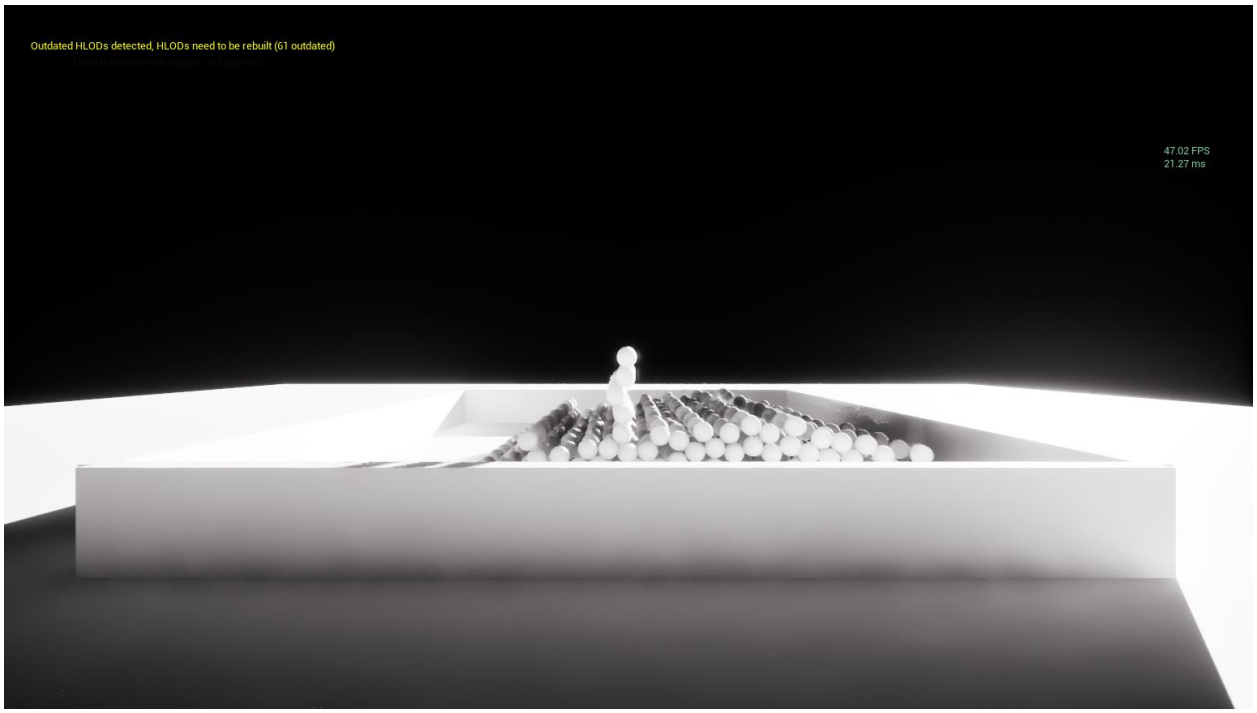

Рисунок 5.8 – 1000 об’єктів з фізикою на Unreal Engine

1000 об’єктів з фізикою Unreal Engine

Сцена виглядає стабільно — рух плавний, помітних затримок немає. Навантаження на GPU помірне, система комфортно справляється з рендером великої кількості примітивів.

Рисунок 5.9 – 2000 об’єктів з фізикою на Unreal Engine

2000 об’єктів з фізикою Unreal Engine

Плавність знижується, але сцена залишається відносно комфортною.

Починають з’являтися перші візуальні артефакти (мерехтіння або дефекти освітлення) — це ознака перевантаження сцени або проблем з LOD/рендерінг-пайплайном.

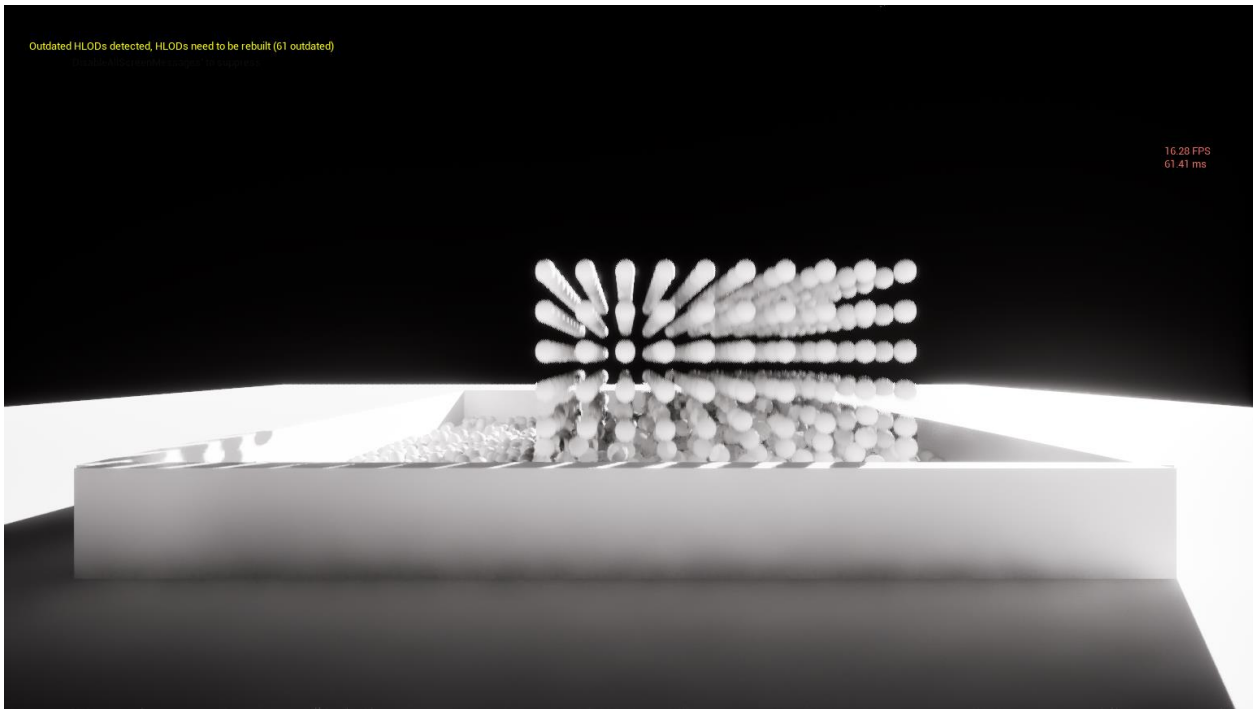

Рисунок 5.10 – 3000 об’єктів з фізикою на Unreal Engine

3000 об’єктів з фізикою Unreal Engine

Рух стає ривками, затримки добре помітні.

Артефакти посилюються, з’являються перебої у відтворенні тіней та віддзеркалень. Двигун починає агресивніше оптимізувати сцену.

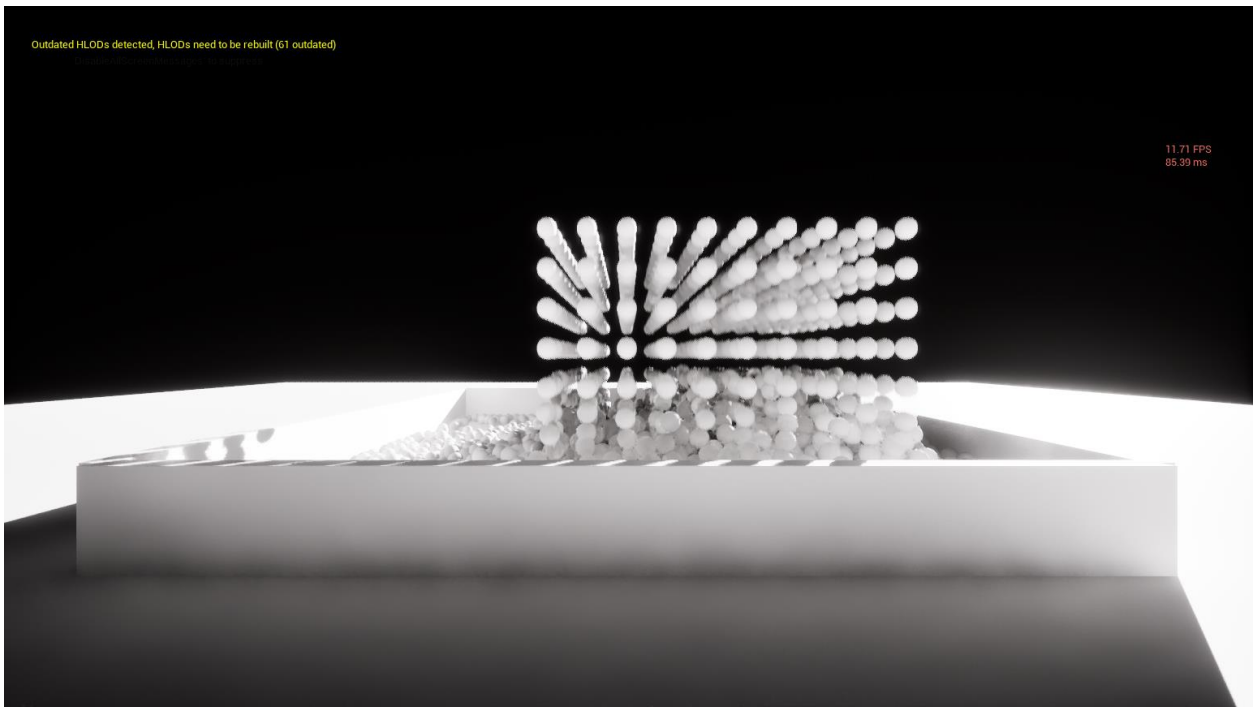

Рисунок 5.11 – 4000 об’єктів з фізикою на Unreal Engine

4000 об’єктів з фізикою Unreal Engine

Сцена працює дуже повільно.

За об’єктами з’являється напівпрозорий шлейф — це результат затримок кадру та накопичення ефектів постобробки.

Можливе «підвисання» камери та ривковий рух.

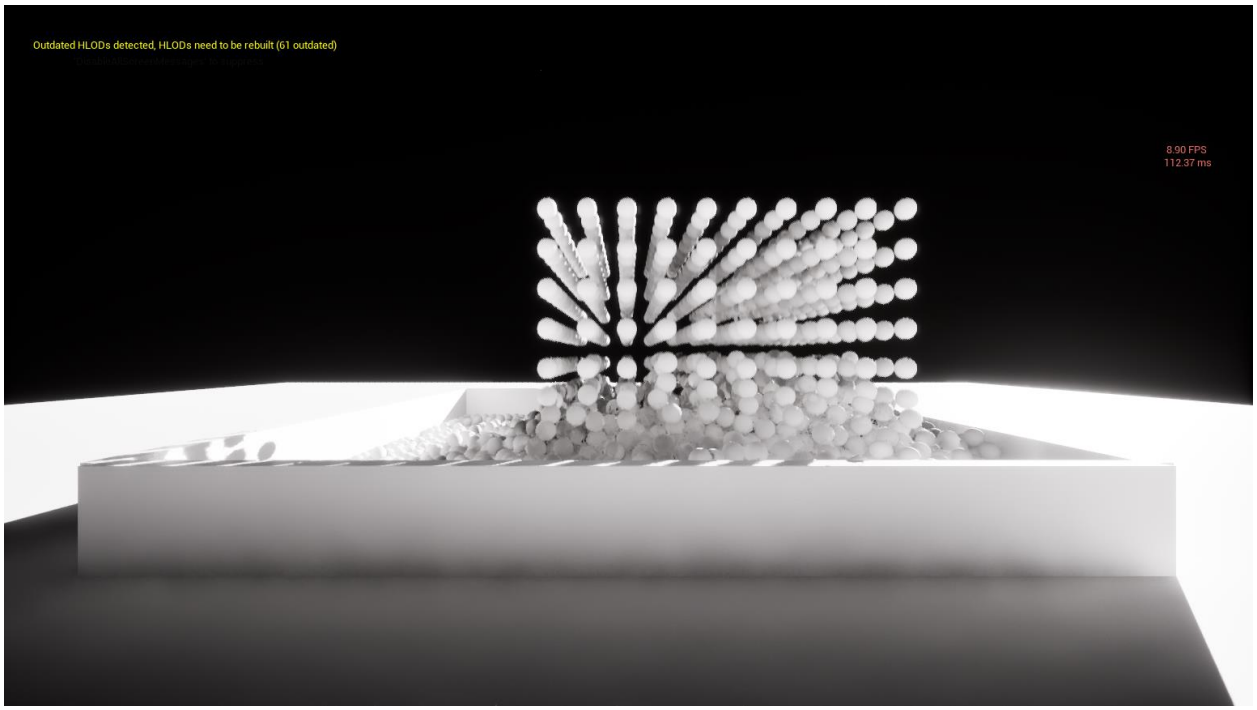

Рисунок 5.12 – 5000 об’єктів з фізикою на Unreal Engine

5000 об’єктів з фізикою Unreal Engine

Робота майже на межі — значні фрізи, шлейфи стають сильнішими, інколи кадри «пропускаються».

Двигун обмежує якість тіней, освітлення і LOD, але цього вже недостатньо, щоб утримати стабільність.

Висновки до експерименту №2

Проведений експеримент показав, що додавання фізики та поява великої кількості зіткнень суттєво впливають на продуктивність обох рушіїв, але характер цього впливу різний.

У Unity спостерігається поступове та відносно рівномірне зниження FPS. При збільшенні кількості об’єктів із фізичними колайдерами та компонентами Rigidbody середній FPS зменшувався послідовно: 475 → 376 → 269 → 240 → 190. Попри помітне зниження мінімального FPS (приблизно на 85 кадрів, із короткочасними просіданнями до 35 FPS), робота сцени залишалася відносно стабільною, без критичних ривків та візуальних спотворень. Це свідчить про достатньо рівномірний розподіл фізичних обчислень у рушії.

Натомість Unreal Engine продемонстрував іншу поведінку. Уже при першому додаванні 1000 фізичних об’єктів спостерігалося різке падіння продуктивності — втрата близько 40 FPS, після чого показники продовжували стрімко погіршуватися: 88 → 51 → 24 → 15 → 10 → 8 FPS. При таких значеннях зображення стає «ривчастим», з’являються візуальні артефакти та затримки відтворення. Для сцени з понад 1000 активних фізичних об’єктів робота рушія стає помітно дискомфортною для користувача.

Таким чином, можна зробити висновок, що Unity краще справляється з поступовим зростанням кількості фізичних об’єктів і забезпечує стабільнішу поведінку, тоді як Unreal Engine значно швидше втрачає продуктивність у фізично насичених сценах, що обмежує його ефективність у сценаріях з великою кількістю одночасних зіткнень.

4.2 Висновки часу рендерінгу кадру до експеримента №2

Висновки (Unity)

Таблиця 7 – Порівняння ms на рушії Unity

	без фізики MID ms	З фізикою MID ms
0	1,2	1,2
1000	1,9	2,1
2000	2,6	2,7
3000	3,5	3,5
4000	4,6	9,7
5000	5,4	15,7

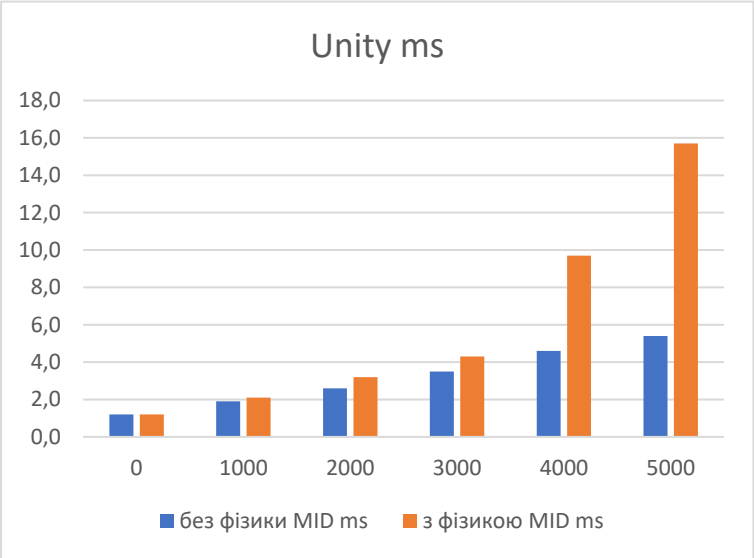

Рисунок 5.13 – Порівняння ms на Unity

За результатами проведених експериментів у Unity було проаналізовано вплив кількості об’єктів на продуктивність, як за показником FPS, так і за часом кадру (ms). Саме показник ms дозволяє точніше оцінити стабільність роботи рушія, оскільки він показує, скільки мілісекунд витрачається на формування одного кадру.

При додаванні нефізичних об’єктів спостерігається плавне та прогнозоване зростання часу рендерингу: від 1,2 ms до приблизно 5,4 ms при 5000 об’єктах. Це означає, що навантаження збільшується поступово, а рушій ефективно опрацьовує рендеринг і трансформації об’єктів без істотних стрибків продуктивності.

Ситуація змінюється після додавання фізики (Collider + Rigidbody). У цьому випадку час кадру зростає значно швидше — з 1,2 ms до 15,7 ms. Причина полягає в тому, що Unity додатково виконує обчислення фізичних взаємодій, перевірку зіткнень, побудову дерев колайдерів і оновлення фізичної симуляції на кожному кадрі. Чим більше об’єктів і зіткнень — тим складнішими стають ці

обчислення.

Особливо помітним є різкий стрибок після 4000 об’єктів. Це свідчить про те, що система переходить у стан перевантаження: CPU витрачає занадто багато часу на симуляцію фізики, через що зростає час кадру і знижується FPS. У цей момент рушій вже не здатний підтримувати стабільну плавність.

Таким чином, експеримент демонструє, що для Unity критичним фактором є не тільки кількість об’єктів на сцені, а й наявність фізики та зіткнень. Для великих сцен необхідно застосовувати оптимізаційні підходи: зменшення кількості активних Rigidbody, спрощення колайдерів, використання тригерів, зон активації та інших інструментів оптимізації.

Висновки (Unreal Engine)

Таблиця 8 – Порівняння ms на рушії Unreal Engine

	без фізики MID ms	з фізикою MID ms
0	10,4	10,7
1000	11,6	22,3
2000	12,4	43,2
3000	13,7	67,6
4000	14,2	94,9
5000	14,9	124,2

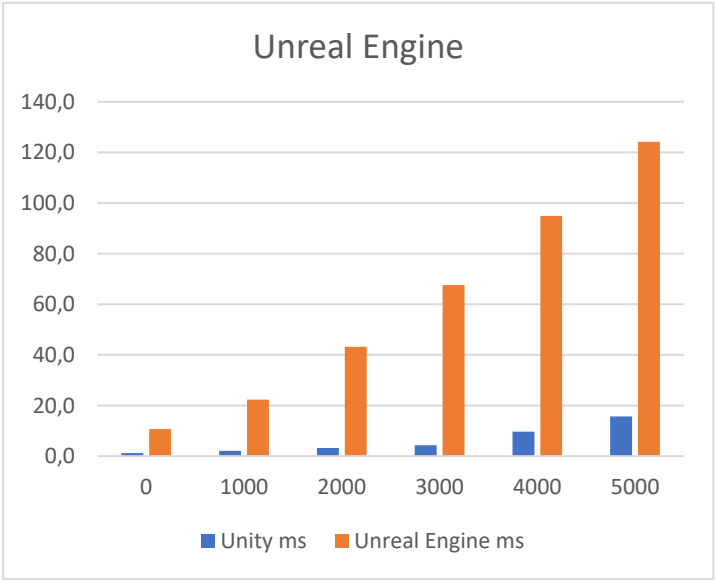

Рисунок 5.14 – Порівняння ms на Unreal Engine

Проведені експерименти показали, що Unreal Engine по-різному реагує на збільшення кількості об’єктів залежно від того, чи використовується фізика.

Без увімкненої фізики об’єкти майже не створюють суттєвого навантаження на систему. Час кадру (ms) зростає повільно та плавно: від приблизно 10,4 ms до 14,9 ms при 5000 об’єктах. Це означає, що рушій стабільно обробляє рендеринг сцени та не перевантажує процесор складними додатковими обчисленнями.

Плавність змін свідчить, що Unreal добре масштабується при роботі з великою кількістю статичних або нефізичних об’єктів.

Однак після додавання фізики ситуація різко змінюється. Час кадру зростає в рази — з 10,7 ms до 124,2 ms. Уже при 1000 об’єктах починають з’являтися короткі «стоп-кадри», а після 3000 картинка стає ривковою і майже непридатною для комфортного сприйняття.

Таке різке зростання пов’язане з особливостями фізичного модуля Unreal Engine. Рушій виконує складні симуляції для кожного об’єкта: перевіряє зіткнення, обчислює взаємодії тіл, оновлює фізичні параметри та синхронізує їх з графічним рендерингом. Коли кількість об’єктів збільшується, навантаження зростає не лінійно, а майже експоненційно — особливо коли об’єкти активно стикаються між собою.

Таким чином, експеримент показує, що Unreal Engine добре працює з великими сценами без фізичної симуляції, але стає сильно залежним від кількості фізичних об’єктів і зіткнень. Для проектів із великою кількістю динамічних тіл необхідні серйозні оптимізації: спрощення фізичних колайдерів, зменшення кількості активних об’єктів, використання фізики та інших інструментів оптимізації.

Порівняльний висновок (Unity vs Unreal Engine)

Проведене дослідження дало змогу порівняти продуктивність Unity та Unreal Engine за двома основними сценаріями: робота зі сценами без фізики та симуляція великої кількості фізичних об’єктів.

У сценах без фізики обидва рушії демонструють стабільну роботу, однак їхня поведінка відрізняється. Unity забезпечує вищий FPS і менший час кадру, що свідчить про ефективне опрацювання великої кількості простих об’єктів. Зміни продуктивності відбуваються плавно, що робить результати передбачуваними. Unreal Engine у цьому режимі показує більший час кадру, проте графік змін залишається рівномірним, а система працює стабільно, без різких стрибків і втрат кадрів.

Ситуація суттєво змінюється після додавання фізики. У Unity спостерігається поступове зниження FPS і зростання ms, однак навантаження збільшується відносно рівномірно. Це означає, що рушій обмежується переважно обчислювальними можливостями CPU, але зберігає керованість процесу симуляції.

У Unreal Engine додавання фізики призводить до різкого і нелінійного збільшення часу кадру. Вже при відносно невеликій кількості динамічних об’єктів з’являються ривки та затримки, а при подальшому зростанні кількості тіл симуляція стає нестабільною. Це свідчить про значну залежність рушія від складності фізичних взаємодій і вартості кожного обчислення.

Загалом результати показують, що:

- Unity краще підходить для сцен із великою кількістю простих об’єктів та помірною фізикою, забезпечуючи стабільнішу роботу;
- Unreal Engine демонструє сильні сторони у графіці та складних сценах, але різко втрачає продуктивність при великій кількості фізичних тіл і зіткнень без оптимізації.

Таким чином, вибір рушія має залежати від специфіки проекту: якщо пріоритет — масові сцени з великою кількістю об’єктів доцільним може бути Unity. Якщо ж проект потребує великої кількості статичних об’єктів, то кращий вибір – Unreal Engine

4.3 Експеримент №3 (створення великої кількості однакових багато полігональних об’єктів)

Базовий замір (сцена без об’єктів)

1. Створити нові проекти в **Unity** та **Unreal Engine**.

2. Створити порожню сцену — залишити тільки камеру та освітлення за замовчуванням.

3. Вирівняти налаштування графіки в обох рушіях:

- однакова роздільна здатність;
- рівень якості;
- тіні;
- відсутність або однакові ефекти постобробки.

4. Запустити сцену в режимі відтворення.

5. За допомогою вбудованих інструментів зафіксувати FPS та час кадру (ms).

6. Повторити замір **3–5 разів** і обчислити середнє значення.

Мета етапу: визначити базовий («нульовий») рівень продуктивності рушіїв без навантаження.

Додавання багатополігональної моделі та замір продуктивності

У цьому етапі використовується **модель мавпи з Blender (~4000 полігонів)**.

1. Імпортувати модель у Unity та Unreal Engine.

2. Створити копії об’єкта партіями по 1000 (1000 → 2000 → 3000 → …).

3. Розмістити моделі рівномірно в просторі так, щоб усі вони потрапляли в камеру.

4. Застосувати однакові матеріали та освітлення.

5. Запустити сцену.

6. Виміряти:

- середній FPS;
- мінімальний FPS;
- час кадру (ms);
- пік використання ресурсів (CPU/GPU, якщо доступно).

7. Записати результати в таблицю.

8. Повторити ідентичні дії в другому рушії.

Мета етапу: дослідити, як складність моделі (велика кількість полігонів) впливає на продуктивність.

Метою експерименту є дослідження того, як ігровий рушій поводиться при значному збільшенні кількості однакових багатополігональних об’єктів у сцені.

Таблиця 9 - Зібрані дані шо до середнього FPS на Unity з створення складної 3D моделі

	MAX FPS	MID FPS	MIN FPS
0	1022	835	122
1000	632	563	120
2000	454	343	156
3000	359	242	116
4000	291	195	90
5000	236	149	74

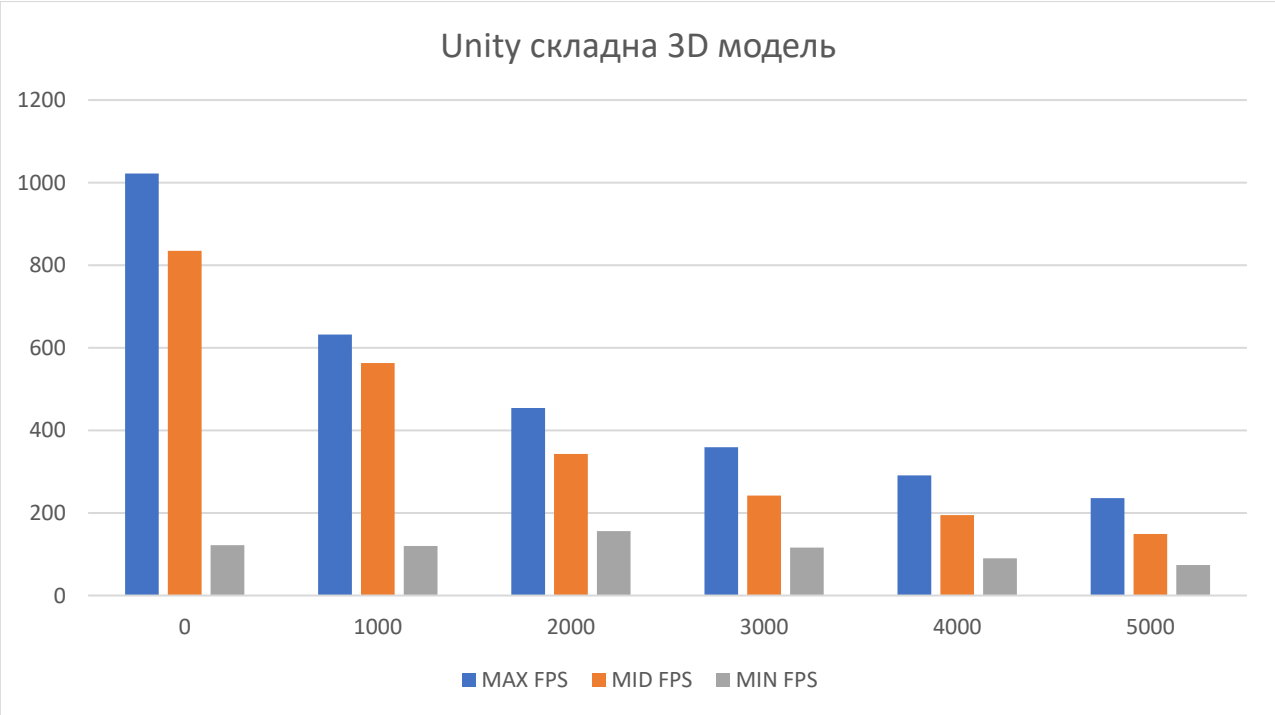

Рисунок 6.1 - Діаграма по отриманим даним до експеримента № 3 на Unity

Після додавання до сцени великої кількості однакових багатополігональних об’єктів було зафіксовано помітне зниження середнього FPS приблизно на 50 кадрів за секунду. При цьому мінімальні значення FPS інколи просідали трохи сильніше, ніж у сцені з простими примітивами, що свідчить про періодичні пікові навантаження на систему рендерингу.

Загалом отримані результати показують, що головний вплив багатополігональних об’єктів полягає саме у підвищенні навантаження на рендеринг, тоді як інші підсистеми рушія залишаються відносно стабільними.

Таблиця 9 - Зібрані дані шо до середнього FPS на Unreal Engine з створення складної 3D моделі

	MAX FPS	MID FPS	MIN FPS
0	83	82	80
1000	80	77	75
2000	80	76	74
3000	78	76	74
4000	78	76	76
5000	77	75	75

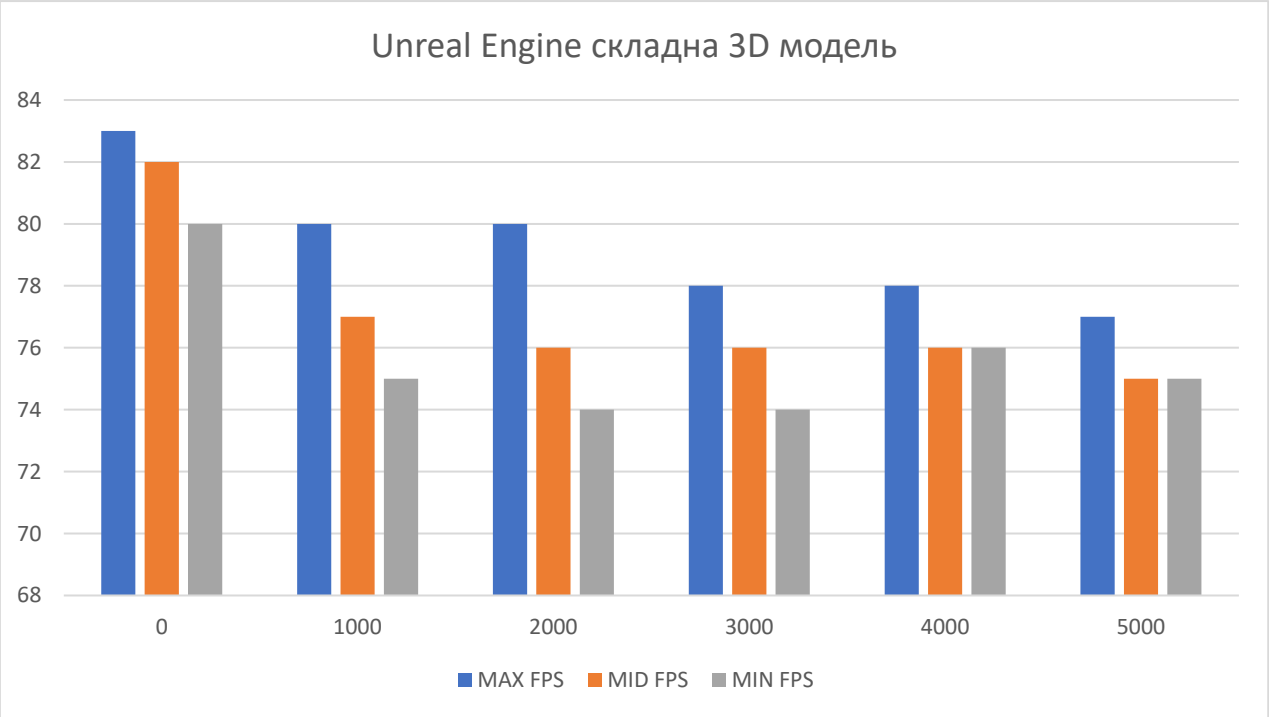

Рисунок 6.2 - Діаграма по отриманим даним до експеримента № 3 на Unreal Engine

Під час експерименту з великою кількістю однакових багатополігональних моделей було зафіксовано нетипову, але показову поведінку рушія Unreal Engine 5. На етапі додавання перших 1000 об’єктів відбувся помітний спад продуктивності — приблизно на 10 FPS у порівнянні з базовим значенням.

Однак подальше збільшення кількості об’єктів дало протилежний ефект:

із кожною наступною тисячею FPS почав поступово стабілізуватися та зростати. До кінця експерименту значення продуктивності фактично не відрізнялося від результатів сцени з простими примітивами, тобто рушій зміг «вирівняти» навантаження.

Це свідчить про те, що Unreal Engine 5 активно застосовує внутрішні механізми оптимізації (кластеризацію, LOD-системи, кешування та роботу рендеринг-потоків). Після початкового «удару» навантаження він перебудовує сцену таким чином, щоб зменшити витрати на відтворення великої кількості однакових моделей.

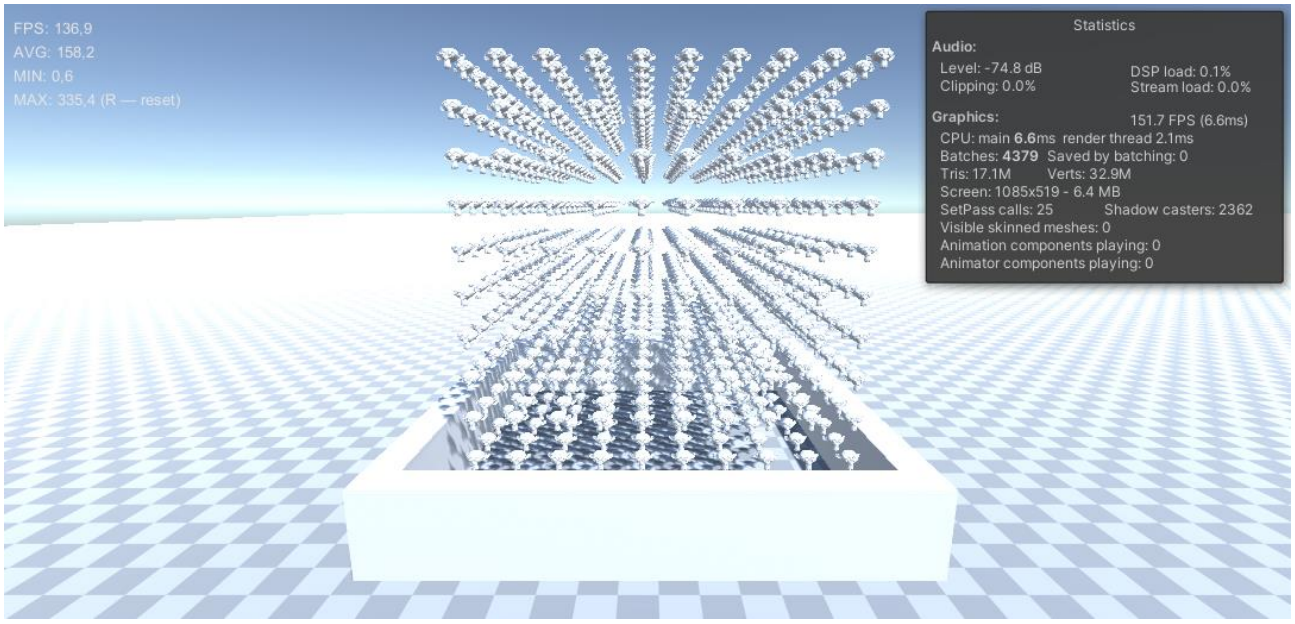

Рисунок 6.3 – 1000 складних об’єктів на Unity

1000 об’єктів Unity

Після додавання першої тисячі складних моделей спостерігається помітний, майже експоненціальний спад FPS. Сцена при цьому залишається стабільною, без візуальних артефактів, але під час появи об’єктів фіксуються короткі просадки мінімальних кадрів.

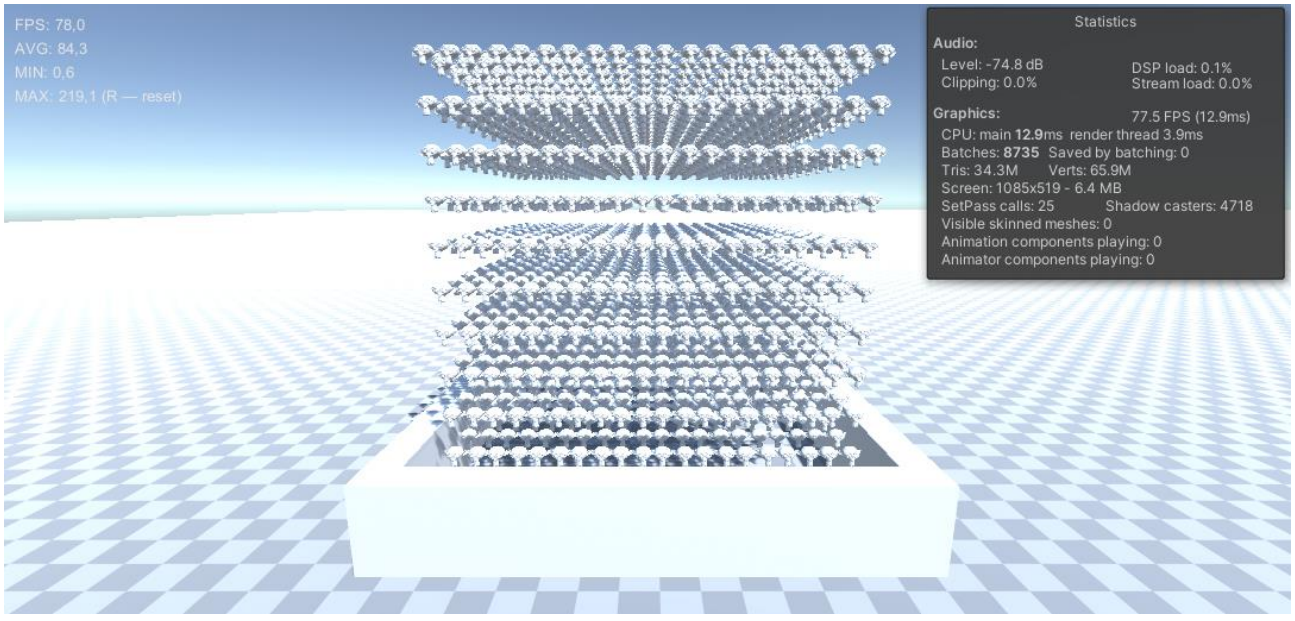

Рисунок 6.4 – 2000 складних об’єктів на Unity

2000 об’єктів Unity

Падіння продуктивності триває, однак воно вже виглядає більш контрольованим. Графіка залишається чіткою, візуальних збоїв не видно. Мінімальні піки FPS все ще проявляються в момент створення нової групи об’єктів.

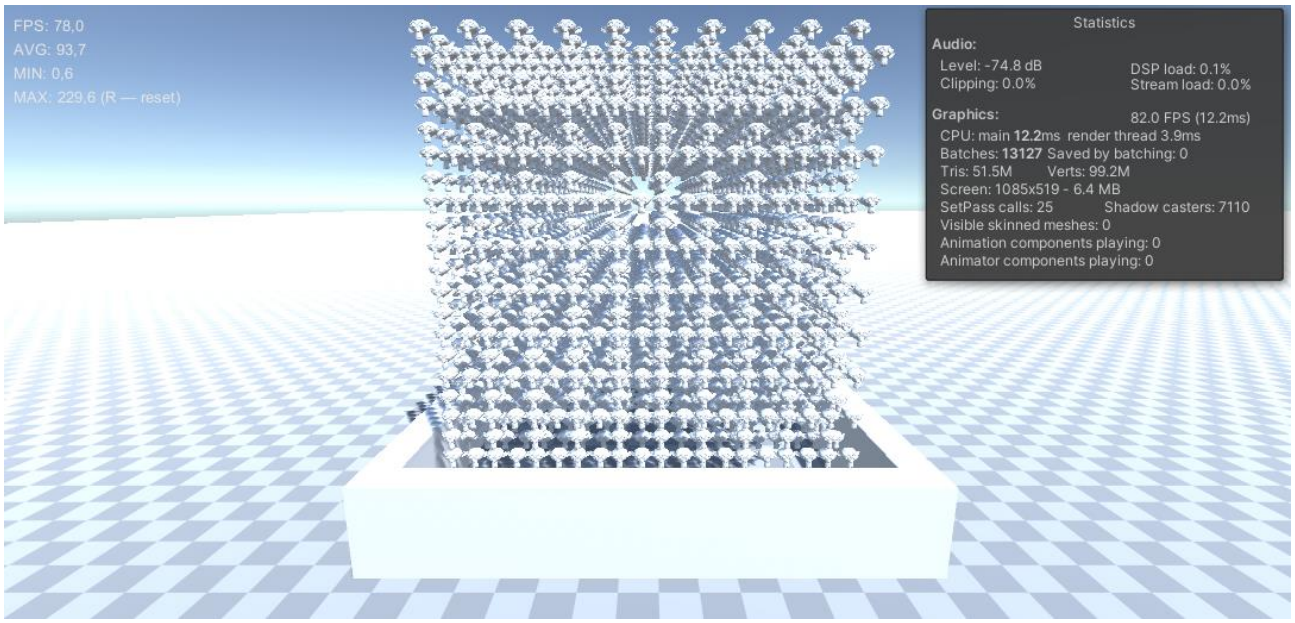

Рисунок 6.5 – 3000 складних об’єктів на Unity

3000 об’єктів Unity

З цього етапу спад FPS стає значно плавнішим. Рушій стабілізується, рендеринг працює рівномірніше. Зображення залишається повністю коректним, без артефактів, а короткі просадки під час створення об’єктів стають менш помітними.

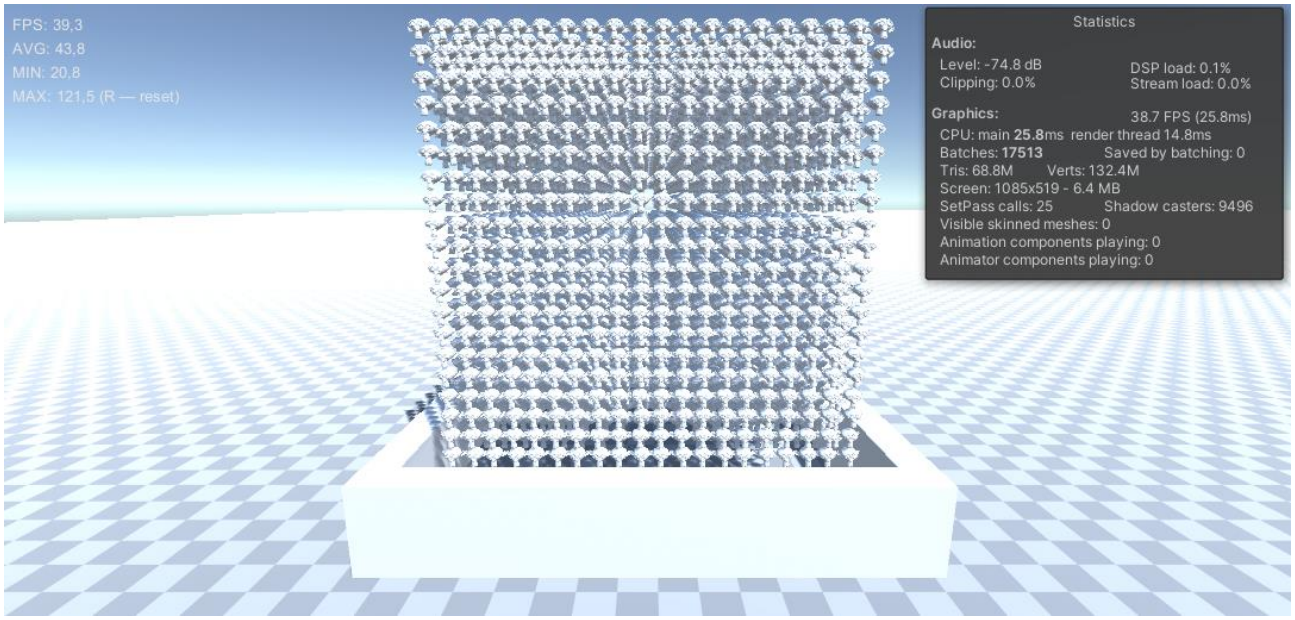

Рисунок 6.6 – 4000 складних об’єктів на Unity

4000 об’єктів Unity

Продуктивність продовжує знижуватися, але вже у лінійнішій формі. Візуальна якість не змінюється — немає “ривків”, спотворень чи мерехтінь. Збільшення навантаження відчувається головним чином по середньому FPS.

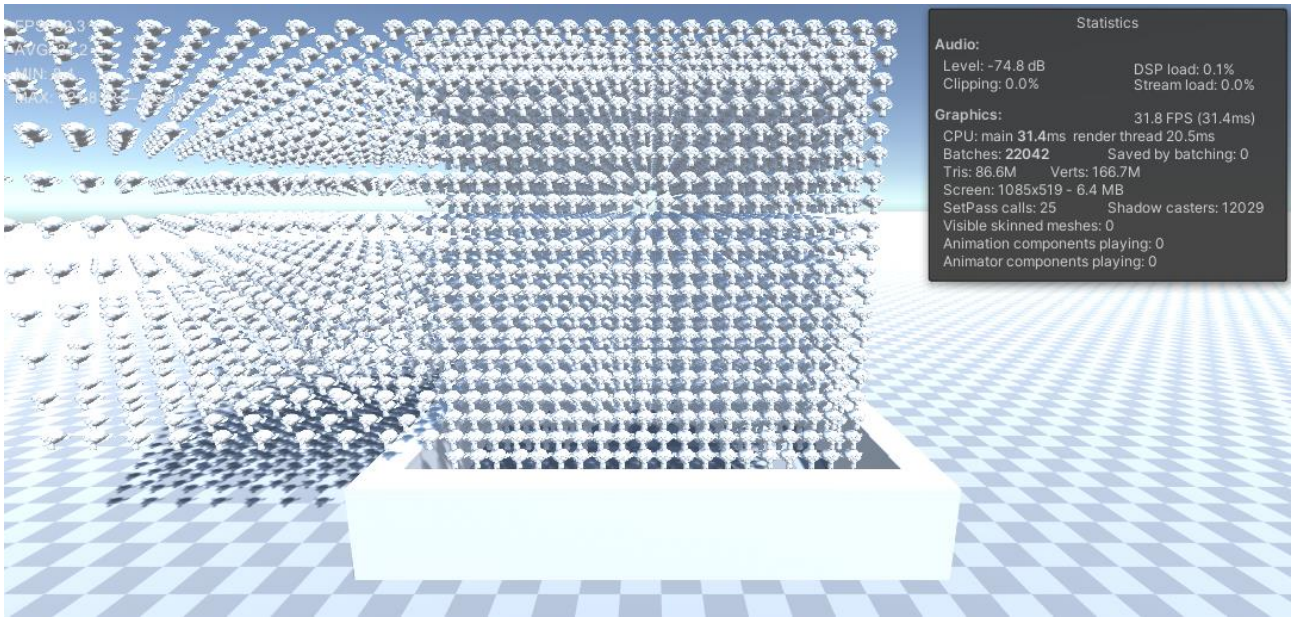

Рисунок 6.7 – 5000 складних об’єктів на Unity

5000 об’єктів Unity

Unity все ще утримує стабільну картинку, хоч середній FPS помітно нижчий.

Різких провалів не виникає, сцена виглядає керованою. Короткі мінімальні піки все ще присутні в момент генерації об’єктів, але вони короткочасні та швидко зникають.

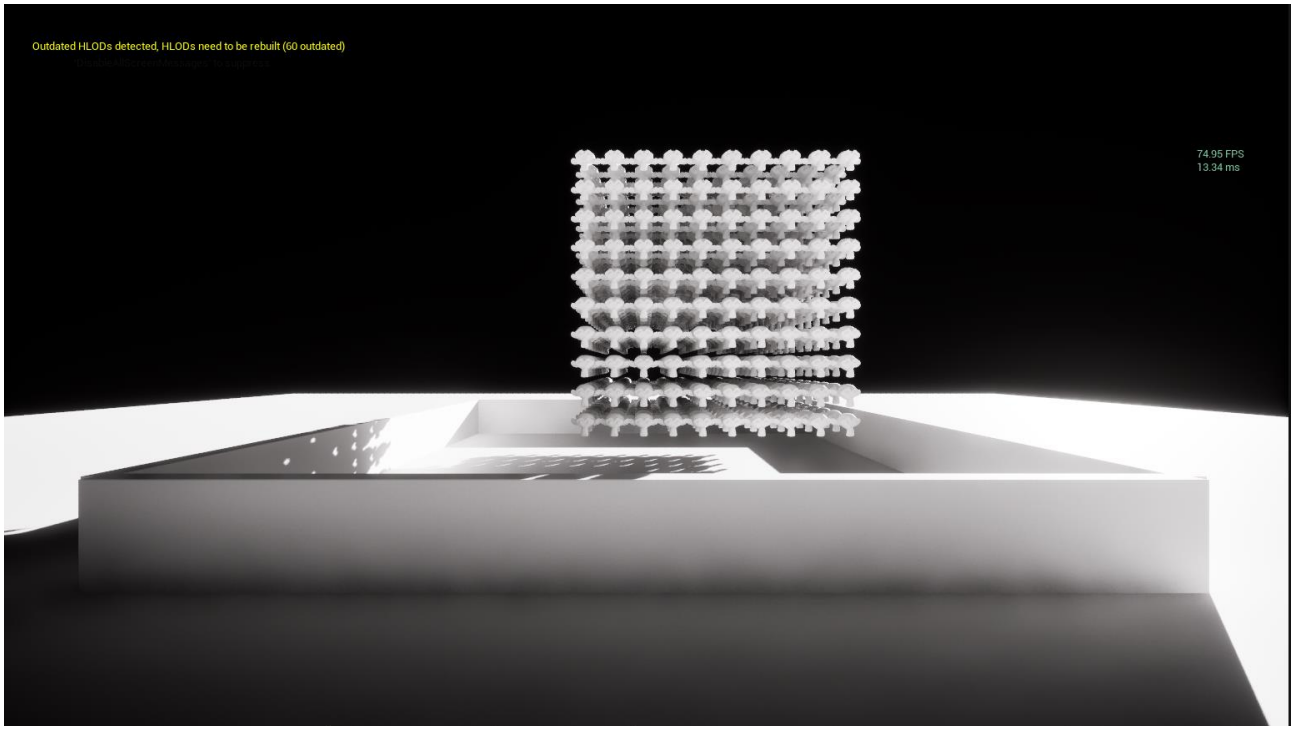

Рисунок 6.8 – 1000 складних об’єктів на Unreal Engine

1000 об’єктів

Після першого додавання тисячі об’єктів FPS помітно зменшується, але залишається стабільним. Зображення виглядає чітко, сцена візуально чиста. Уже на цьому етапі можна інколи побачити легке мерехтіння тіней, але воно короткочасне.

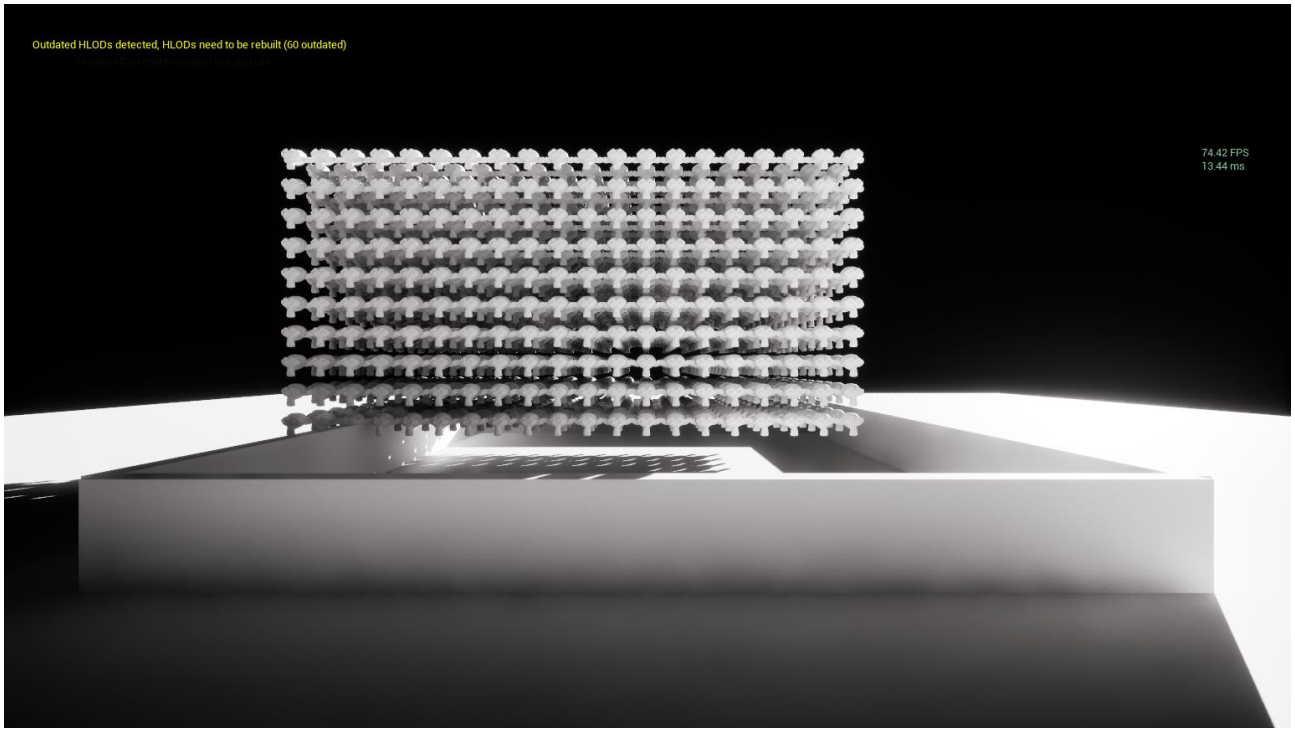

Рисунок 6.9 – 2000 складних об’єктів на Unreal Engine

2000 об’єктів

Подальше збільшення кількості об’єктів майже не впливає на FPS — продуктивність залишається приблизно на тому ж рівні. Проте стають помітніші тіньові артефакти: деякі тіні починають “підмигувати” або зникати на мить під час руху камери.

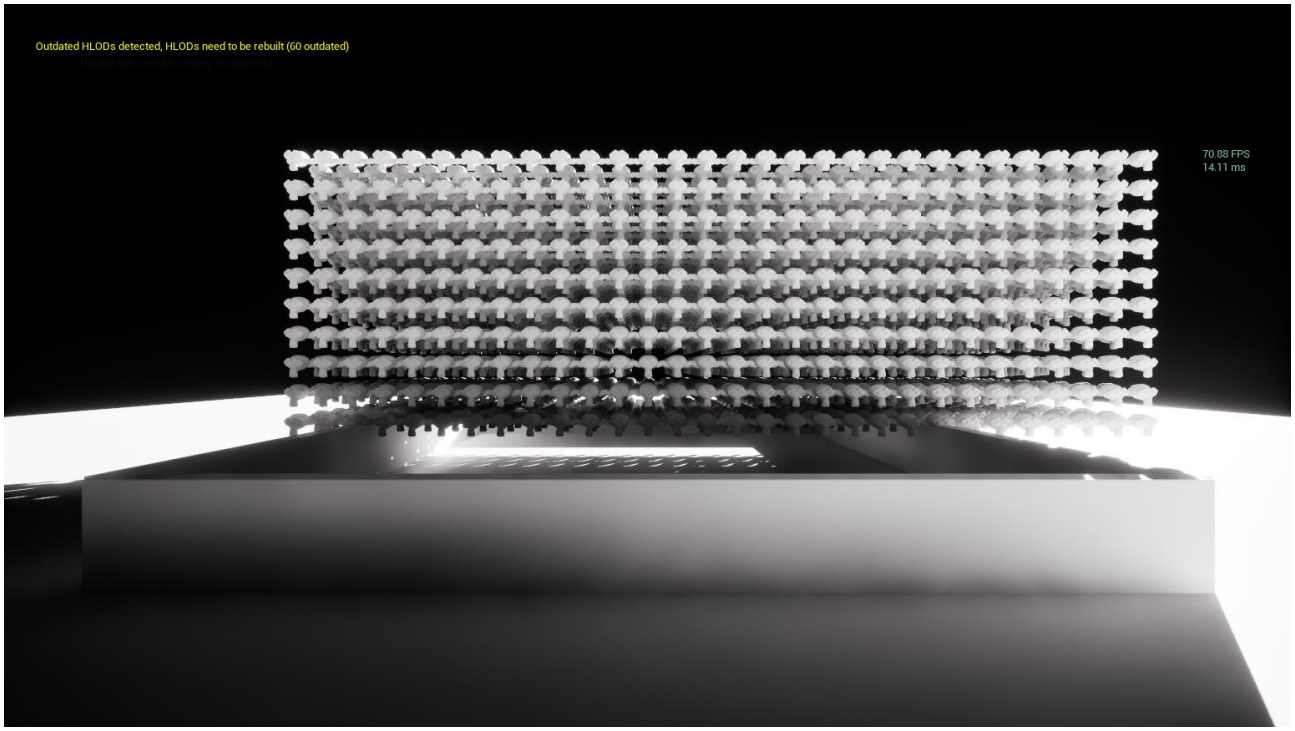

Рисунок 6.10 – 3000 складних об’єктів на Unreal Engine

3000 об’єктів

Чіткість картинки зберігається, кадри залишаються відносно стабільними. Водночас візуальні артефакти проявляються частіше: з’являється мерехтіння окремих елементів сцени, інколи — коротке пропадання тіней від груп об’єктів.

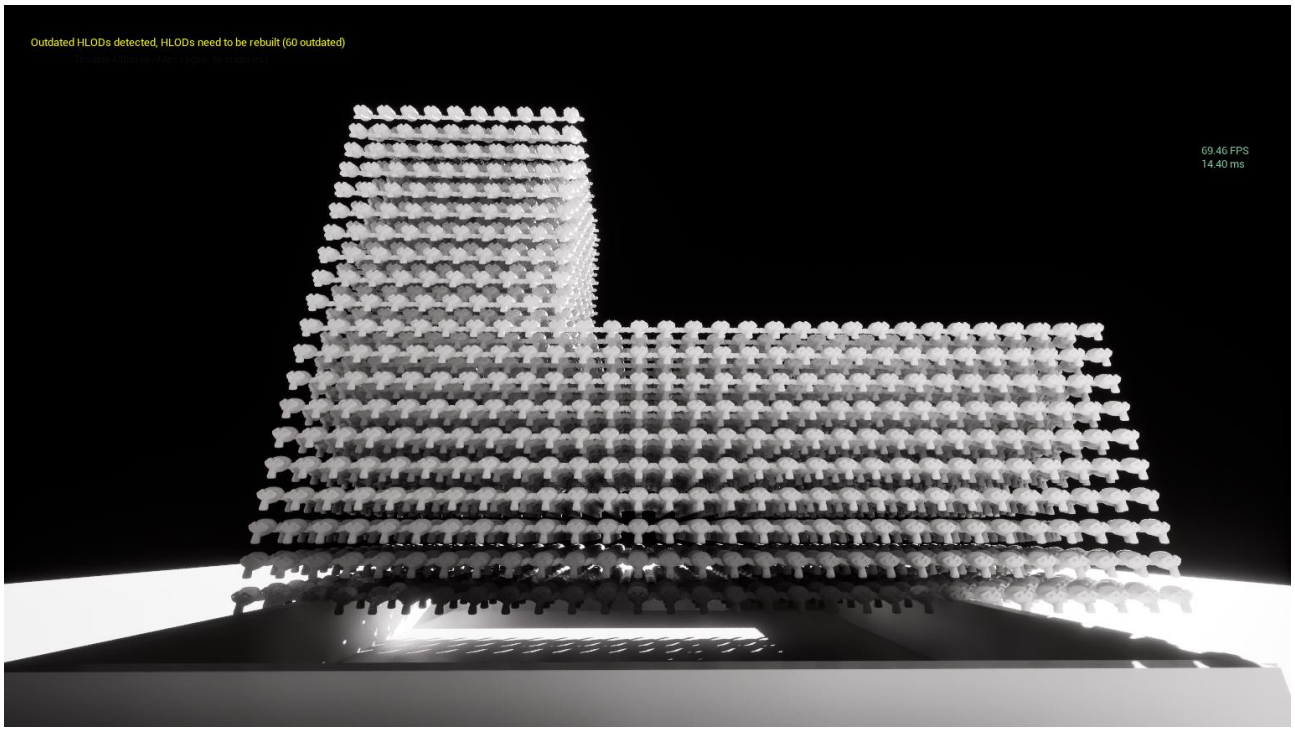

Рисунок 6.11 – 4000 складних об’єктів на Unreal Engine

4000 об’єктів

FPS все ще майже не змінюється, але система дедалі активніше “економить” ресурси за рахунок тіней. На зображенні помітні ділянки, де тіні стають розмитими або повністю зникають, особливо на далеких об’єктах.

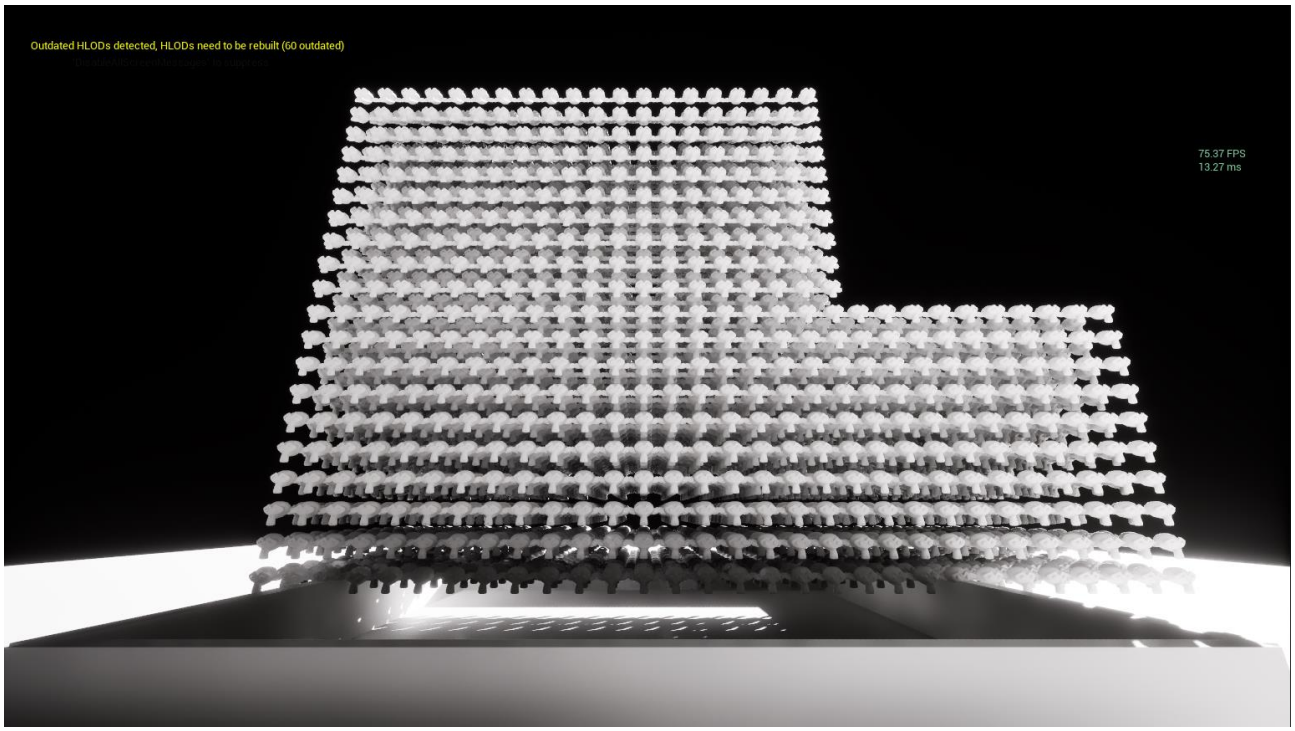

Рисунок 6.12 – 5000 складних об’єктів на Unreal Engine

5000 об’єктів

Продуктивність утримується на близькому рівні до попередніх етапів, однак візуальні артефакти проявляються найвиразніше. Місцями спостерігається сильніше мерехтіння, переривчасті тіні та ефект “пропадання” окремих джерел освітлення.

Висновки до експерименту №3

Отримані результати свідчать, що Unreal Engine орієнтований на проекти з підвищеною візуальною складністю та кінематографічною подачею. Рушій забезпечує високу якість зображення, детальне освітлення та складні шейдери, але вимагає продуманого підходу до фізики: велика кількість динамічних об’єктів без оптимізації призводить до різкого зростання часу кадру та втрати плавності. За умови грамотного контролю фізичних обчислень (спрощення колайдерів, обмеження активних тіл, використання LOD-фізики) Unreal Engine залишається потужним рішенням для проектів, де пріоритетом є картинка та кінематичність.

Порівняння кадрів із простою та складною моделлю показало різну поведінку рушіїв.

Таблиця 10 – Різниця кадрів на Unity

	FPS проста	FPS складна	різниця
0	795	835	-40
1000	535	563	-28
2000	400	343	57
3000	307	242	65
4000	237	195	42
5000	206	149	57

У Unity додавання складної моделі спричиняє відчутне зниження продуктивності — у середньому приблизно на 50 FPS. Попри це, якість візуалізації зберігається: не спостерігається артефактів чи спотворень, однак стають помітнішими мінімальні піки FPS, тобто зростає нестабільність у важких сценах.

Таблиця 11 – Різниця кадрів на Unreal Engine

	FPS проста	FPS складна	різниця
0	88,5	82	6,5
1000	86,5	77	9,5
2000	85,5	76	9,5
3000	83	76	7
4000	78	76	2
5000	72	75	-3

У Unreal Engine спостерігається інша тенденція: початкове додавання складної моделі викликає помітний спад (близько 9,5 FPS), проте зі збільшенням кількості елементів подальше падіння відбувається плавніше. Це означає, що Unreal краще масштабує сцени зі складними матеріалами та деталями, поступово перерозподіляючи навантаження на рендеринг.

Таким чином, хоча Unity демонструє вищий загальний FPS, Unreal Engine забезпечує більш рівномірний, передбачуваний спад продуктивності при збільшенні складності сцени. Це робить його особливо придатним для проектів, де ключову роль відіграють візуальна глибина, кінематографічні ефекти та контрольована фізична симуляція.

4.4 Експеримент №4 (створення великої кількості однакових багато полігональних об’єктів з симуляціею фізики)

Базовий замір (сцена без об’єктів)

1. Створити нові проекти в **Unity** та **Unreal Engine**.

2. Створити порожню сцену — залишити тільки камеру та освітлення та створити підлогу та стінки щоб спровокувати більше зіткнень

3. Вирівняти налаштування графіки в обох рушіях:

- однакова роздільна здатність;
- рівень якості;
- тіні;
- відсутність або однакові ефекти постобробки.

4. Запустити сцену в режимі відтворення.

5. За допомогою вбудованих інструментів зафіксувати FPS та час кадру (ms).

6. Повторити замір **3–5 разів** і обчислити середнє значення.

Мета етапу: визначити базовий («нульовий») рівень продуктивності рушіїв без навантаження.

Додавання багатополігональної моделі та замір продуктивності

У цьому етапі використовується **модель мавпи з Blender (~4000 полігонів)**.

1. Імпортувати модель у Unity та Unreal Engine.

2. Створити копії об’єкта партіями по 1000 (1000 → 2000 → 3000 → …).

3. Розмістити моделі рівномірно в просторі так, щоб усі вони потрапляли в камеру.

4. Застосувати однакові матеріали та освітлення.

5. Запустити сцену.

6. Виміряти:

- середній FPS;
- мінімальний FPS;
- час кадру (ms).

7. Записати результати в таблицю.

8. Повторити ті ж дії в другому рушії.

Мета етапу: дослідити, як складність моделі (велика кількість полігонів) впливає на продуктивність.

Метою даного експерименту є дослідити, як увімкнення фізики, гравітації та масових зіткнень складних багатополігональ Mesh Collider впливає на продуктивність ігрових рушіїв Unity та Unreal Engine.

Таблиця 12 - Зібрані дані середнього FPS на Unity з створення складної 3D моделі з фізикою

	MAX FPS	MID FPS	MIN FPS
0	1041	819	145
1000	619	418	108
2000	452	346	77
3000	344	220	48
4000	280	136	27
5000	47	34	9

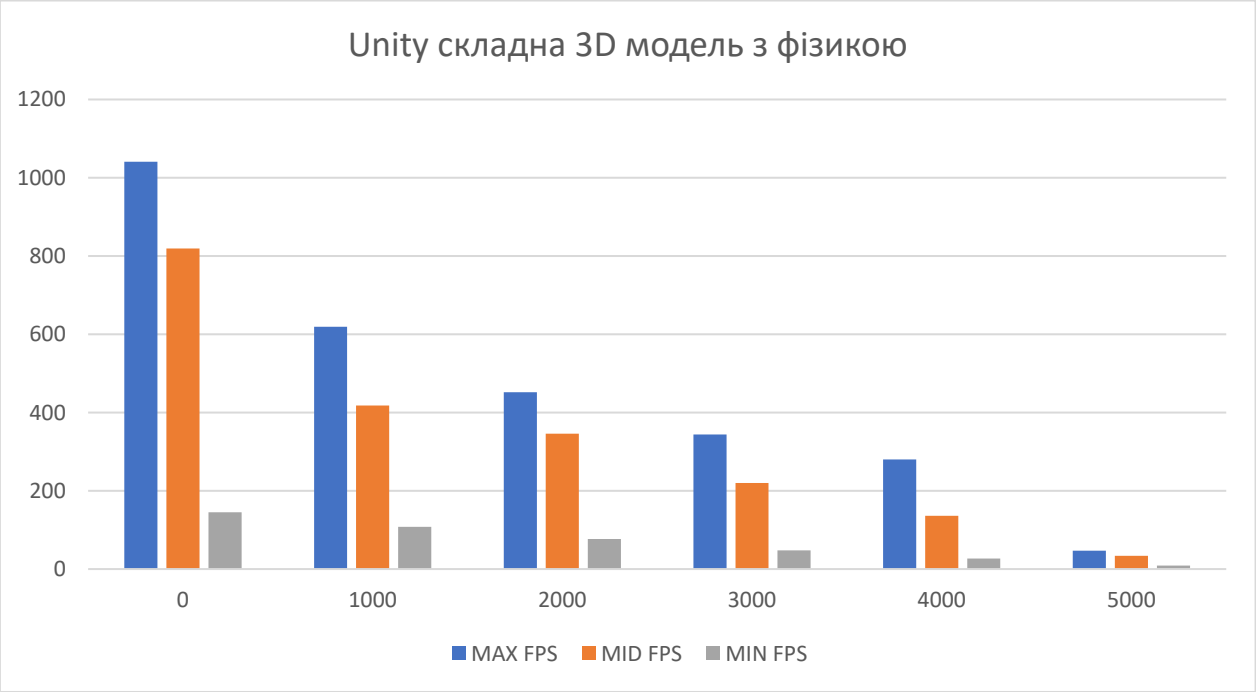

Рисунок 7.1 - Діаграма отриманих даних до експеримента №4 на Unity

Отримані дані показують, що Unity рівномірно справляється з суттєвим навантаженням, помітні просадки середнього FPS, хоча вже на 5000 об’єктах зображення стає рваним через те, що FPS менше 60

Таблиця 13 - Зібрані дані середнього FPS на Unreal Engine з створення складної 3D моделі з фізикою

	MAX FPS	MID FPS	MIN FPS
0	84	83	81
1000	33	30	28
2000	16	14	11
3000	11	9	8
4000	7	7	6
5000	5	5	4

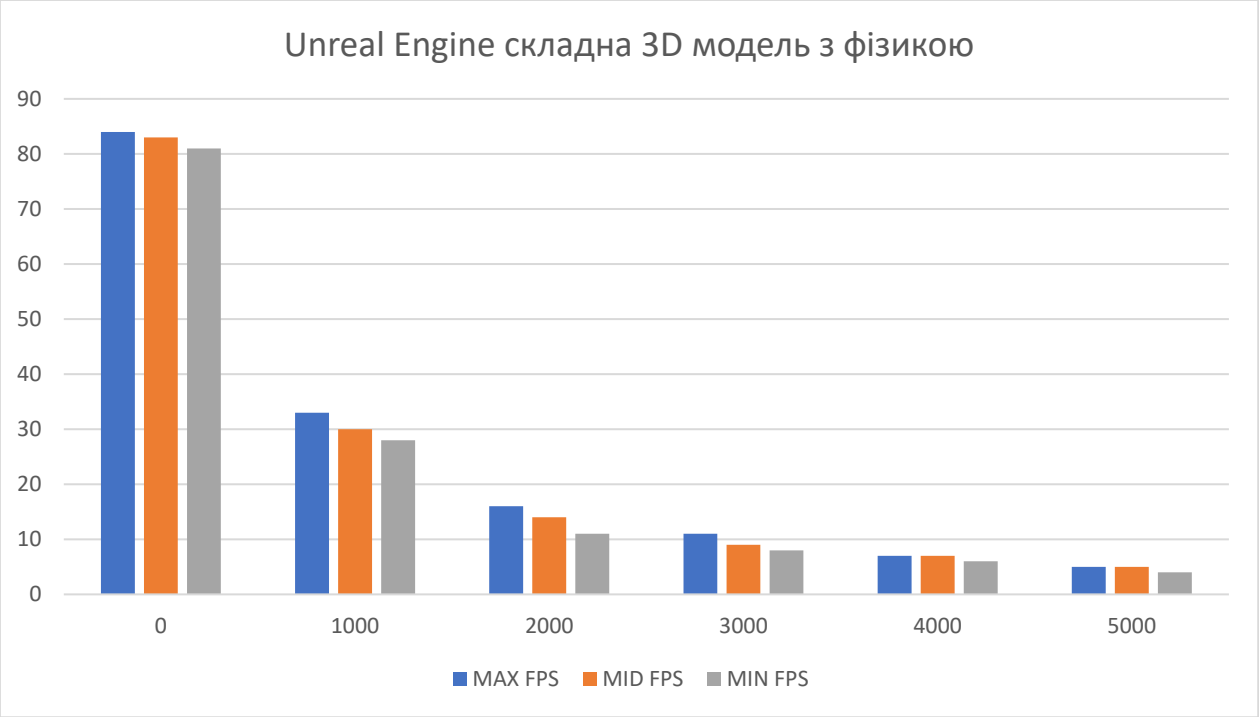

Рисунок 7.2 - Діаграма отриманих даних до експеримента № 3 на Unreal Engine

Unreal Engine показує дуже неприємні результати, рушій одразу втрачає велику кількість кадрів на секунду, спад яких іде по експоненті доходячи до 5

FPS, показує, що рушій складно переносить багатополігональні моделі з фізикою та зіткненнями.

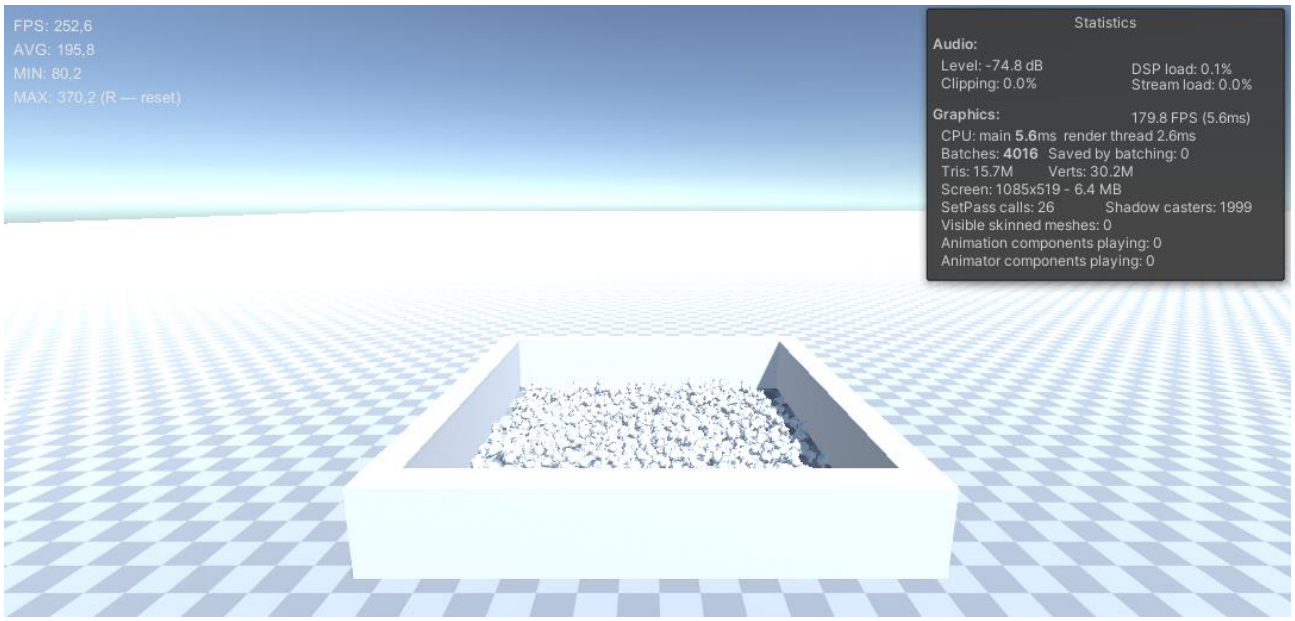

Рисунок 7.3 – 1000 складних об’єктів з фізикою на Unity

1000 об’єктів Unity

Сцена виглядає плавною — середній FPS залишається досить високим. Короткі просадки помітні лише в момент появи нових об’єктів, але вони швидко зникають. Візуальних артефактів немає, рух камери стабільний.

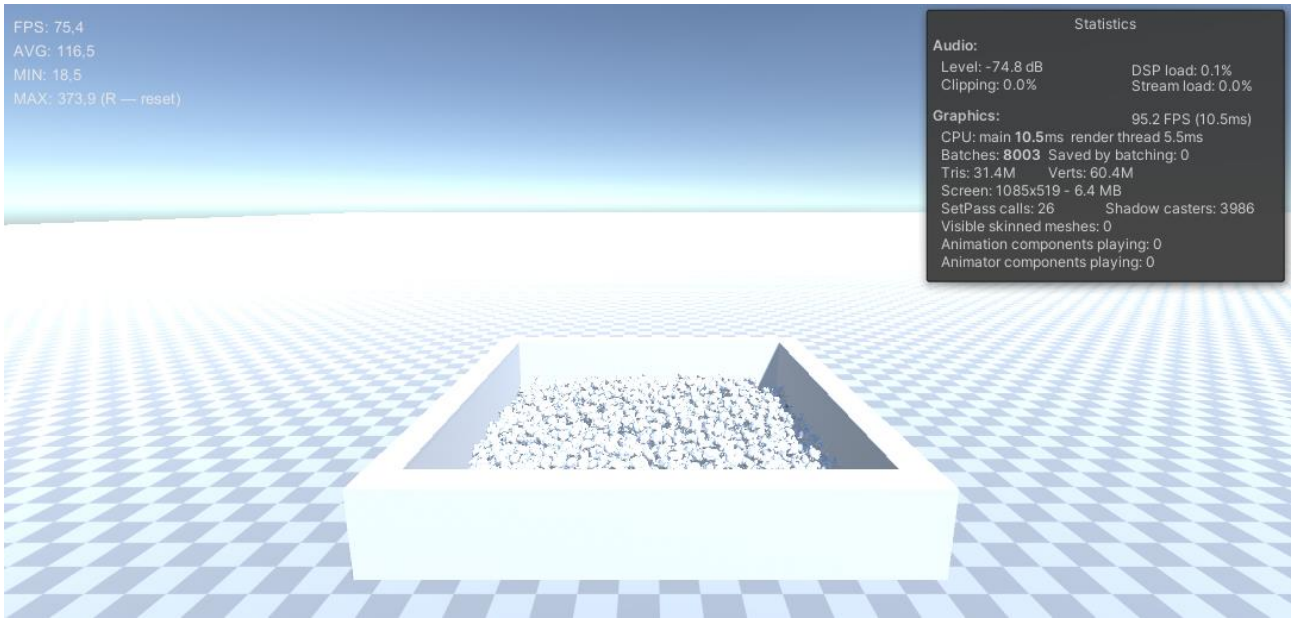

Рисунок 7.4 – 2000 складних об’єктів з фізикою на Unity

2000 об’єктів Unity

Зображення все ще сприймається комфортним. Проте починають з’являтися періодичні мінімальні піки FPS, пов’язані з великою кількістю активних зіткнень між об’єктами. У загальному плані сцена лишається плавною.

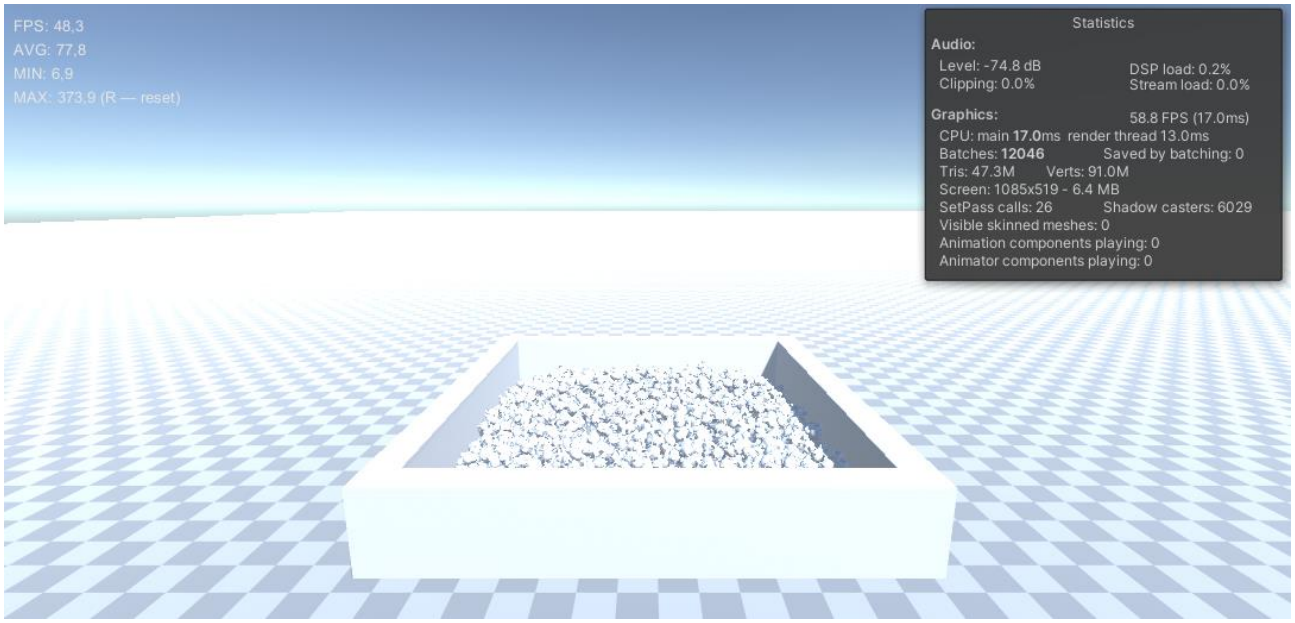

Рисунок 7.5 – 3000 складних об’єктів з фізикою на Unity

3000 об’єктів Unity

Плавність зберігається, але піки продуктивності стають частішими. Фізичний рушій працює помітно інтенсивніше: кожен контакт об’єктів викликає короткочасні “підгальмовування”. Якість зображення при цьому не погіршується.

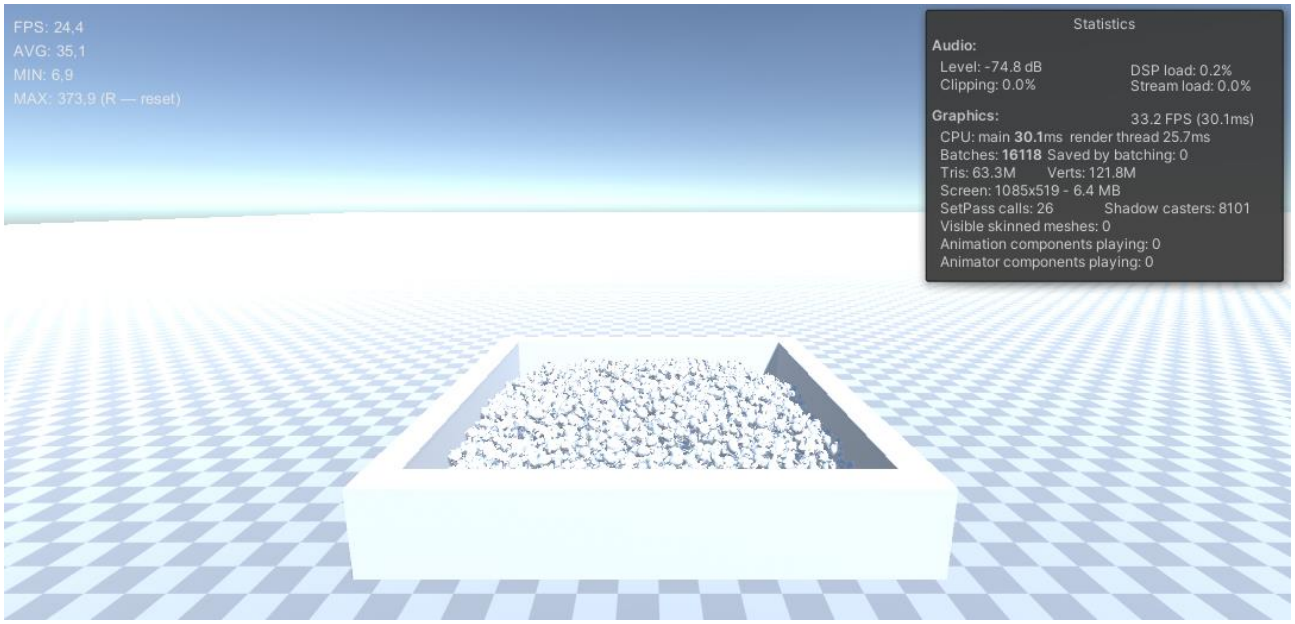

Рисунок 7.6 – 4000 складних об’єктів з фізикою на Unity

4000 об’єктів Unity

Навантаження на фізику значно збільшується. Просідання FPS стають регулярними, а мінімальні значення кадрів падають усе нижче. Візуально сцена вже не така стабільна: рухи виглядають менш рівномірними.

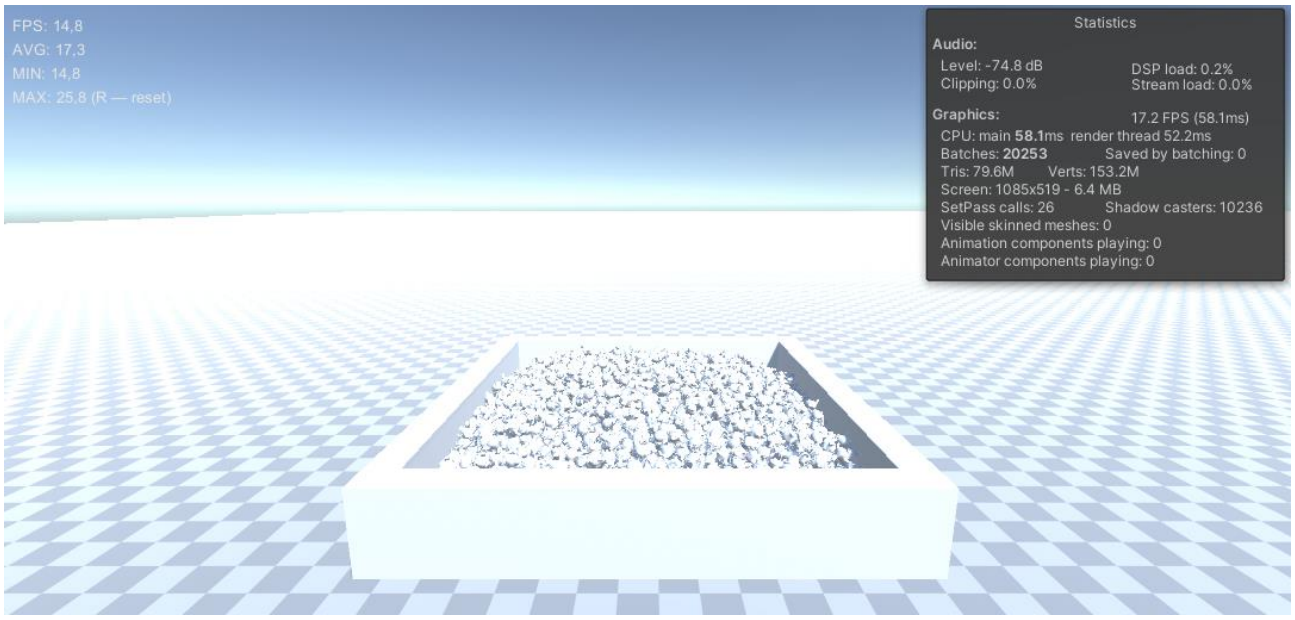

Рисунок 7.7 – 5000 складних об’єктів з фізикою на Unity

5000 об’єктів Unity

Середній FPS знижується приблизно до 34 кадрів, що робить зображення помітно рваним і менш приємним для сприйняття. Через постійні зіткнення велика кількість фізичних об’єктів створює швидкі й часті просадки продуктивності.

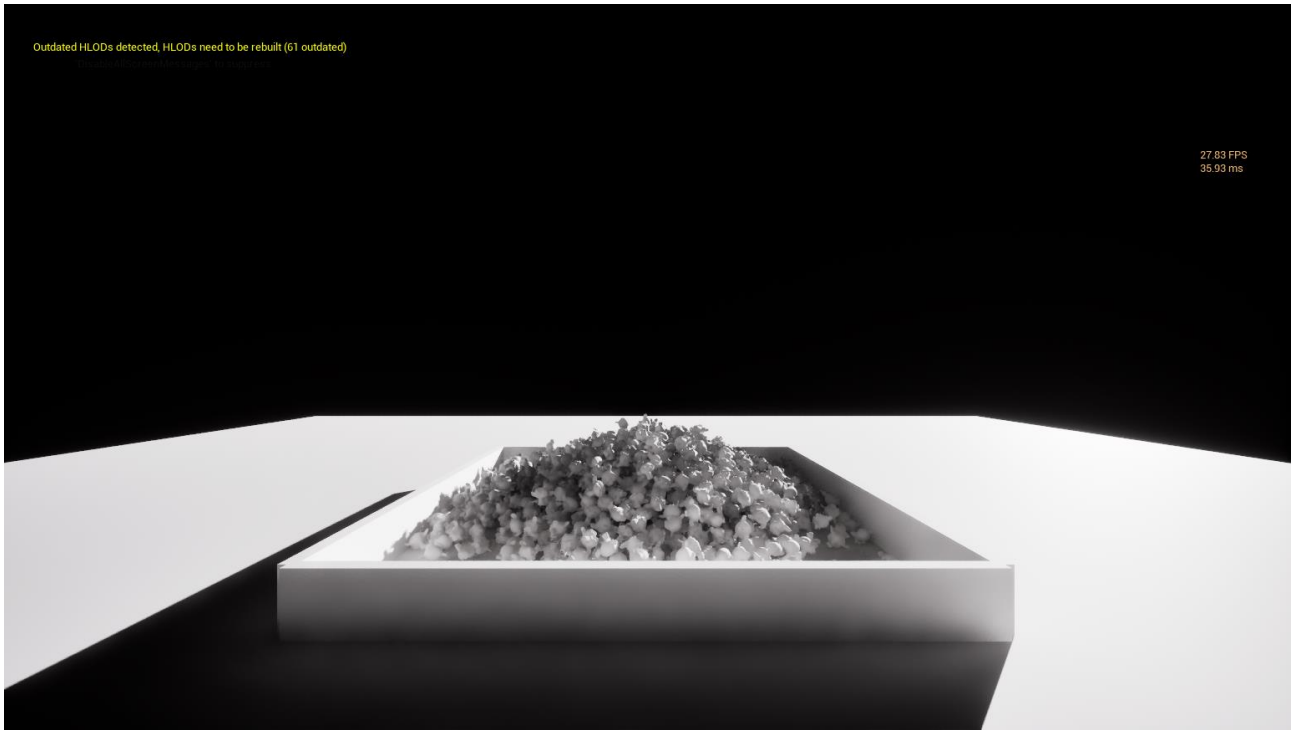

Рисунок 7.8 – 1000 складних об’єктів з фізикою на Unreal Engine

1000 об’єктів Unreal Engine

Попри помітне зниження середнього FPS, зображення залишається досить плавним і приємним для сприйняття. Візуальних артефактів майже немає, сцена виглядає стабільно.

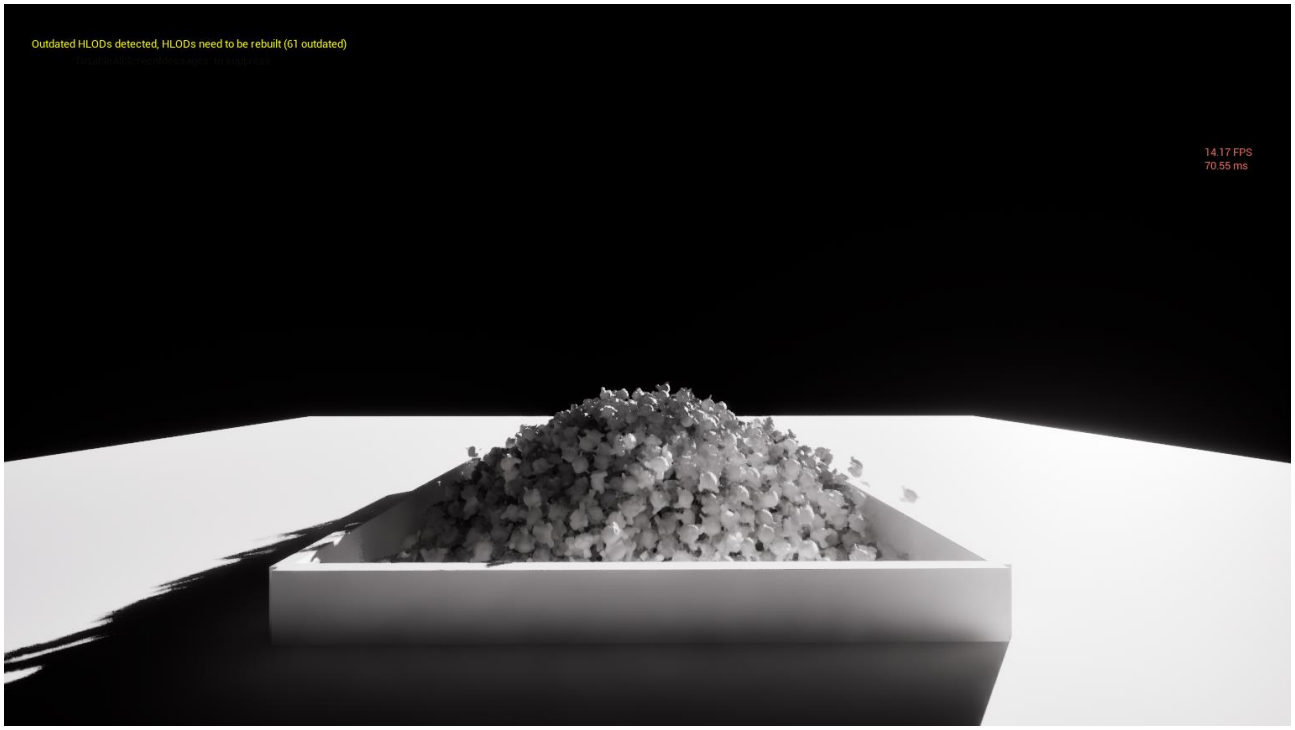

Рисунок 7.9 – 2000 складних об’єктів з фізикою на Unreal Engine

2000 об’єктів Unreal Engine

Починають з’являтися перші візуальні артефакти — переважно мерехтіння та непослідовні тіні. Середній FPS суттєво знижується (приблизно до ~14) і рухи стають менш рівномірними.

Рисунок 7.10 – 3000 складних об’єктів з фізикою на Unreal Engine

3000 об’єктів Unreal Engine

Артефакти проявляються частіше: мерехтіння стає помітним не лише на тінях, а й на об’єктах. FPS падає приблизно до ~9, що робить сцену майже неплавною.

Рисунок 7.11 – 4000 складних об’єктів з фізикою на Unreal Engine

4000 об’єктів Unreal Engine

Продуктивність продовжує різко зменшуватися (близько ~7 FPS). У сцені з’являється відчутне «смикання» зображення, а артефакти — особливо з тінями — стають постійними.

Рисунок 7.12 – 5000 складних об’єктів з фізикою на Unreal Engine

5000 об’єктів Unreal Engine

Середній FPS знижується приблизно до ~5, і сцена стає важко сприйнятною. За об’єктами з’являється помітний напівпрозорий шлейф, що створює ефект «розмазування» руху та робить тест майже непридатним для комфорного перегляду.

Висновки до експерименту №4

Отримані результати показують, що Unity краще справляється з великим числом складних об’єктів, зберігаючи прийнятну продуктивність навіть при значному навантаженні. Проте в реальних проектах бажано уникати великої кількості динамічних об’єктів з фізикою, оскільки саме вони створюють найбільші просідання FPS.

У свою чергу, Unreal Engine 5 демонструє високу ефективність при роботі зі складними, але статичними моделями — якість зображення залишається стабільною, а рендеринг оптимізується. Однак після додавання фізики продуктивність різко падає, тому фізику варто застосовувати лише до ключових і відносно небагатьох об’єктів, щоб зберегти стабільність сцени.

4.5 Висновки що до часу рендерінгу кадру

Висновки (Unity)

Таблиця 14 – Порівняння часу кадру з різними обектами на Unity

	без фізики MID ms	з фізикою MID ms
0	1,2	1,2
1000	2,1	2,2
2000	2,9	3,9
3000	4,2	7,0
4000	5,5	14,2
5000	6,7	30,3

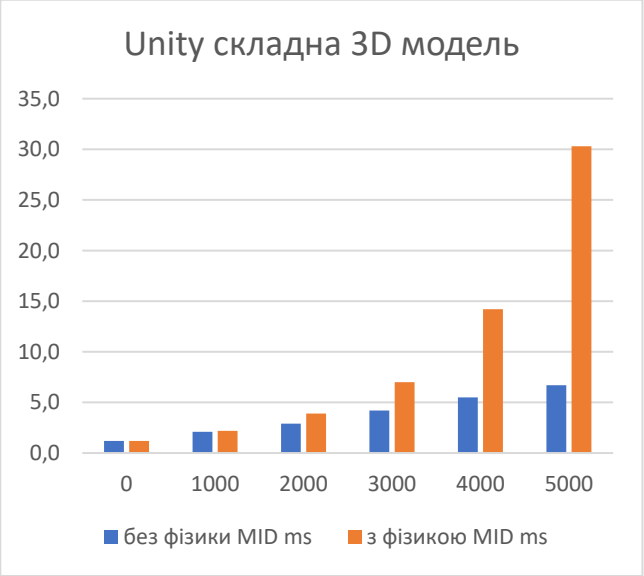

Рисунок 7.13 - Порівняння ms на рушії Unity

Аналіз середнього часу кадру (MID ms) для складної моделі в Unity показує суттєву різницю між сценаріями без використання фізики та з увімкненою фізикою.

За відсутності фізичних обчислень час кадру зростає рівномірно та лінійно — від 1,2 ms до 6,7 ms. Графік не містить різких стрибків, рушій працює стабільно навіть при максимальному навантаженні. Візуальні артефакти відсутні, зображення майже не втрачає якості, що свідчить про коректну роботу рендерингу та ефективне управління ресурсами.

При додаванні фізики спостерігається експоненціальне зростання часу кадру — від 1,2 ms до 30,3 ms. Хоча середній FPS може залишатися на прийнятному рівні, з’являються помітні стопкадри та провисання, що негативно впливає на плавність відображення. Водночас візуальні артефакти не виникають, що вказує на коректну, але ресурсоємну роботу фізичного модуля.

Unity добре справляється зі складними моделями без фізики, забезпечуючи стабільний час кадру та високу якість зображення. Проте використання фізики

суттєво підвищує навантаження на систему, що призводить до різкого зростання часу кадру і появи помітних просідань продуктивності, які необхідно враховувати при розробці сцен із великою кількістю фізичних об’єктів.

Висновки (Unreal Engine)

Таблиця 15 - Порівняння часу кадру з різними обектами на Unreal Engine

	без фізики MID ms	з фізикою MID ms
0	12,3	12,3
1000	12,6	32,4
2000	12,9	66,1
3000	13,1	104,0
4000	13,4	145,5
5000	13,5	195,2

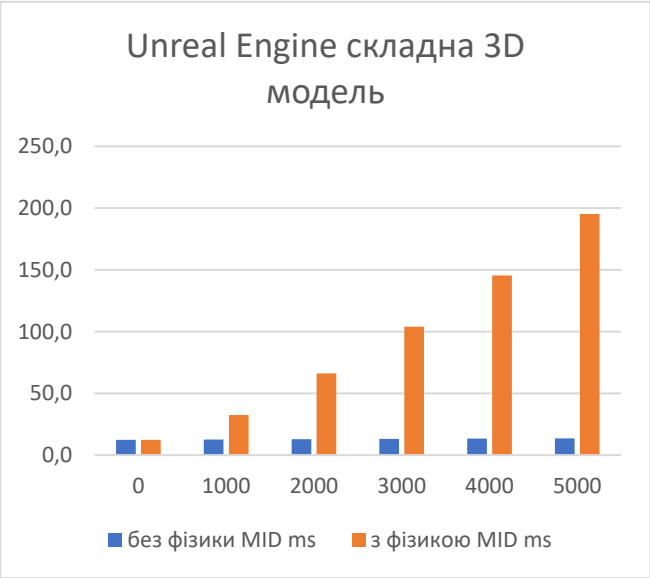

Рисунок 7.14 - Порівняння ms на рушії Unreal Engine

Аналіз середнього часу кадру (MID ms) в Unreal Engine для складної моделі показує різку відмінність у поведінці рушія залежно від використання фізики.

Без використання фізичних обчислень Unreal Engine демонструє високу стабільність часу кадру: значення змінюються незначно — від 12,3 ms до 13,5 ms навіть при значному збільшенні кількості складних об’єктів. Це свідчить про ефективну роботу системи рендерингу та здатність рушія добре масштабуватися при великій кількості геометрично складних моделей. Зображення залишається стабільним, без візуальних артефактів.

При додаванні фізики спостерігається різке, майже експоненціальне зростання часу кадру — від 12,3 ms до 195,2 ms. Таке навантаження призводить до суттєвого падіння плавності відображення. Під час тестування з’являються

мерехтіння, некоректна робота тіней та загальне погіршення якості зображення, що робить сцену візуально неприйнятною для комфортного сприйняття.

Unreal Engine ефективно обробляє велику кількість складних об’єктів за відсутності фізики, зберігаючи стабільний час кадру. Однак використання фізики суттєво збільшує обчислювальне навантаження, що призводить до різкого росту часу кадру та появи візуальних проблем, які необхідно враховувати при розробці сцен з інтенсивною фізичною взаємодією.

Загальні висновки

У межах даної дипломної роботи було проведено комплексне дослідження продуктивності сучасних ігрових рушіїв Unity та Unreal Engine 5 з метою оцінки їх поведінки при різних типах навантаження. Дослідження включало аналіз FPS, часу кадру (ms), стабільності відображення, впливу кількості об’єктів, складності моделей та використання фізики.

У ході експериментів встановлено, що Unity характеризується низьким початковим навантаженням на систему, що дозволяє досягати дуже високого стартового FPS. Однак зі зростанням кількості об’єктів, зіткнень і тіней спостерігається рівномірний, але значний спад продуктивності, а також чутливість до фонових застосунків. Водночас рушій демонструє коректну роботу фізики без появи візуальних артефактів, навіть за умов високого навантаження, що робить його придатним для проектів з інтенсивною фізичною взаємодією за умови оптимізації сцен.

Unreal Engine 5, у свою чергу, одразу резервує значну частину системних ресурсів, що знижує початковий FPS, проте забезпечує дуже стабільний час кадру та мінімальні коливання продуктивності. Рушій добре масштабується при додаванні як простих, так і складних геометричних об’єктів, майже не реагуючи на сторонні процеси. Разом з тим експерименти показали, що UE5 погано переносить велику кількість фізичних об’єктів і зіткнень, що призводить до різкого зростання часу кадру та появи візуальних дефектів, таких як мерехтіння та некоректні тіні.

Додатково встановлено, що середній FPS не завжди є показником реальної плавності, оскільки значний вплив на суб’єктивне сприйняття мають мінімальні піки FPS і стабільність часу кадру. У цьому аспекті Unreal Engine 5 демонструє кращу плавність без фізики, тоді як Unity забезпечує більш передбачувану та візуально коректну роботу при використанні фізичних симуляцій.

Таким чином, результати дослідження підтверджують, що вибір ігрового рушія має базуватися на специфіці проекту. Unity є більш гнучким та придатним

для сцен із великою кількістю фізичних взаємодій і обмеженими ресурсами, тоді як Unreal Engine 5 доцільно використовувати для візуально складних сцен із великою кількістю об’єктів за умови мінімізації фізичних розрахунків.

Отримані результати можуть бути використані як практичні рекомендації для розробників ігор та інтерактивних застосунків, а також як основа для подальших досліджень, зокрема аналізу впливу різних методів оптимізації, типів освітлення та багатопотокової обробки фізики.

Бібліографічний список

1. Wikipedia Unity (game engine) (2026) https://en.wikipedia.org/wiki/Unity_(game_engine)
2. Wikipedia Unreal Engine (2026) https://en.wikipedia.org/wiki/Unreal_Engine
3. Unreal Engine (2026) https://www.unrealengine.com/en-US
4. Unity (2026) https://unity.com/
5. Gamedev DOU (2026) https://gamedev.dou.ua/articles/unity-or-unreal-engine/
6. FoxmindED (2026) https://foxminded.ua/rozrobka-ihor-unity-unreal-engine/
7. Unity in Action: Multiplatform game development in C# (2018)
8. Mastering Game Development with Unreal Engine 4

ДОДАТОК А

Технічне завдання

ЗАТВЕРДЖУЮ
Перший проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

Програма дослідження продуктивності ігрових рушіїв Unity та Unreal Engine
Технічне завдання
44165850.01521-01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Ярослав ГРИВА
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО
44165850.01521-01

Програма дослідження продуктивності ігрових рушіїв Unity та Unreal Engine

Технічне завдання

44165850.01521-01

Листів 11

2026

44165850.01521-01

ЗМІСТ

Вступ 2
1 Підстава для розробки 3
2 Призначення розробки 4
2.1 Функціональне призначення розробки 4
2.2 Експлуатаційне призначення розробки: 5
3 Вимоги до програми 6
3.1 Вимоги до функціональних характеристик 6
3.2 Вимоги до надійності 6
3.3 Вимоги експлуатації 6
3.4 Вимоги до складу та параметрів технічних засобів 7
3.5 Вимоги до інформаційної та програмної сумісності 7
4 Вимоги до програмної документації 8
5 Стадії та етапи розробки 9
6 Порядок прийняття 10
7 Технічно-економічні показники 11

ВСТУП

У сучасному світі дуже активно розвинута галузь комп’ютерних розваг. Комп’ютерні ігри, мобільні ігри набирають великих обертів не тільки серед дітей, а й серед дорослих. Тому в наш час – це дуже затребувана сфера для розвитку творчості, креативного мишлення, а також бізнесу. Цим самим ця галузь останнім часом залучає мільйони людей, з’являються нові робочі місця та розвиваються великі компанії і приносять значні доходи.

Економічний двигун: Створює робочі місця та приваблює інвестиції.

Культурний вплив: Формує тренди, впливає на цінності та спосіб життя.

Соціальна функція: Надає можливість для відпочинку, релаксації та соціалізації.

Та в останні роки ігрова індустрія вступає в нову еру. Вона не лише вийшла за межі мобільних пристроїв, але й стала нормою кросплатформні ігрові ігри, такі як PUBG або Fortnite, ігрові середовища для спілкування та хмарні ігрові сервіси на основі підписки від таких компаній, як PlayStation, Google, Amazon та інших. У багатьох відношеннях ця галузь сьогодні перебуває в центрі тенденцій соціальних мереж, електронної комерції та розваг.

Історично в ігровій індустрії домінували великі гравці та інвестори, орієнтовані на ігри, але вона розвивалася в останні роки, задовго до появи карантинних обмежень у зв’язку з COVID-19. У цей час люди були ізольовані від один від одного, та щоб проводити свій час з користю, почали активно вивчати IT напрямок у сфері розробки ігор. Швидке зростання в цьому секторі привабило безліч нових компаній, що створюються в цій сфері, а також геймерів, глядачів, фанатів і, як наслідок, ширшу базу інвесторів.

Практична робота полягає в створенні умов для проведення однакових експериментів та порівняння навантаження одинакових умов на різних ігрових рушіях . З отриманими результатами ми зможемо зробити висновки щодо оптимізації рушіїв в різних умовах та взяти до уваги результати для забезпечення оптимізації проектів .

44165850.01521-01

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ ректора Українського державного університету науки і технології Радкевич А.В. «Про затвердження тем та призначення керівників дипломних проектів» № 1401ст від 02.10.2025 року

Тема проекту: «Дослідження продуктивності ігрових рушіїв Unity та Unreal Engine».

Керівник дипломного проекту: Іванов О. П.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою експерименту є порівняння ігрових рушіїв Unity і Unreal Engine за однакових тестових умов при виконанні ідентичних навантаженнях. Аналіз полягає у зміні параметрів об’єкту:

- кількість відображуваних об’єктів;
- складність геометрії об’єктів;
- наявність фізичних симуляцій;
- вплив цих факторів на продуктивність;
- збір та фіксація результатів при різних навантаженнях;
- порівняльний аналіз отриманих даних;

2.1 Функціональне призначення розробки

Мета порівняння полягає в оцінці того, як Unity і Unreal Engine із зростанням навантаження змінюються ключові показники продуктивності та які внутрішні механізми рушіїв визначають ці зміни.

До об’єкту порівняння входять такі складові:

- ігрові рушії Unity та Unreal Engine — програмні платформи, що виконують рендеринг, фізичні обчислення та управління сценами;
- тестові сцени — однакові за структурою середовища, що містять 3D-об’єкти, джерела світла;
- інструментальні засоби збору даних — скрипти, розроблені для автоматичного вимірювання та логування продуктивності;
- обчислювальні ресурси — апаратна платформа (комп’ютер), на якій проводяться тести: процесор, відеокарта, оперативна пам’ять, система зберігання даних;
- параметри експерименту — кількість об’єктів у сцені, рівень деталізації, тип освітлення, частота оновлення кадру, тощо.

2.2 Експлуатаційне призначення розробки

Експлуатаційне призначення даної розробки полягає у практичному використанні результатів проведених експериментів для підвищення ефективності процесу створення ігрових та інтерактивних застосунків. Отримані дані дають змогу здійснити коректний і обґрунтований вибір ігрового рушія відповідно до поставлених технічних і продуктивних вимог.

Результати дослідження можуть бути використані для:

- **оцінки та вибору рушія** розробки залежно від типу навантаження, кількості об’єктів і необхідного рівня фізичної симуляції;
- **планування процесу розробки**, зокрема визначення обмежень щодо кількості динамічних та фізичних об’єктів у сцені;
- **оптимізації використання апаратних ресурсів** з метою досягнення максимальної продуктивності та стабільності застосунку.

Таким чином, розробка сприяє підвищенню загальної ефективності створюваного продукту та зменшенню ризиків, пов’язаних із некоректним вибором технологій на ранніх етапах розробки.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Програмний продукт повинен забезпечувати:

– створення та відтворення тестових сцен в ігрових рушіях Unity та Unreal Engine;

– можливість додавання великої кількості об’єктів (від 0 до 5000) з фізикою та без неї;

– фіксацію показників продуктивності, зокрема:

– середнього FPS;

– мінімального FPS;

– часу рендерингу кадру (ms);

– проведення багаторазових замірів з подальшим усередненням результатів;

– відображення та збереження отриманих даних у вигляді таблиць і графіків;

– можливість порівняльного аналізу результатів між рушіями.

3.2 Вимоги до надійності

Програма повинна:

- стабільно працювати протягом усього часу проведення експериментів;
- коректно обробляти великі навантаження без аварійного завершення;
- забезпечувати відтворюваність експериментів за однакових умов;

3.3 Вимоги до умов експлуатації

Програмний продукт призначений для експлуатації:

- у навчальних, дослідницьких та тестових умовах;
- на персональних комп’ютерах із настільною операційною системою;
- при наявності встановленого відповідного ігрового рушія та драйверів відеокарти.

3.4 Вимоги до складу та параметрів технічних засобів

Апаратне забезпечення, необхідне для роботи:

- процесор: не нижче 4-ядерного з частотою від 3.0 ГГц;
- оперативна пам’ять: не менше 16 ГБ ;
- відеокарта з підтримкою обсяг відеопам’яті — від 6 ГБ;
- накопичувач з вільним простором не менше 20 ГБ;
- пристрої введення: клавіатура та миша.

3.5 Вимоги до інформаційної та програмної сумісності

Програмні продукти розробляється для всіх видів операційних систем сімейства “Windows” починаючи від версії 10 та наступні версії.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

– технічне завдання

– пояснювальна записка;

– текст програми;

– керівництво користувача для проведення досліджень.

Вся документація програмних додатків повинна задовольняти вимоги до програмної документації.

44165850.01521-01

5 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця 16 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	04.09.25 –15.10.25
Робочий проект	Програмування та відлагодження програми.	18.10.25 – 22.10.25
	Тестування програми	23.10.25 – 27.11.25
	Розробка, узгодження і затвердження програмної документації.	28.11.25 – 29.11.25

6 ПОРЯДОК ПРИЙНЯТТЯ

Контроль за виконанням роботи здійснює керівник розробки доц. Іванов О. П.

7 ТЕХНІЧНО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Показники та їх розрахунок не були описані у документах бо розробка програмного продукту несе в собі навчальний характер, а не комерційний.

ДОДАТОК Б

Текст програми

ЗАТВЕРДЖУЮ
Перший проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

Програма дослідження продуктивності ігрових рушіїв Unity та Unreal Engine
Текст програми
44165850.01521–01 12–01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Грива ЯРОСЛАВ
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО
44165850.01521–01 12–01

Програма дослідження продуктивності ігрових рушіїв Unity та Unreal Engine

Текст програми
44165850.01521–01 12–01
Листів 4

2026

АНОТАЦІЯ

Документ 44165850.01521–01 12 01 «Дослідження продуктивності ігрових рушіїв Unity та Unreal Engine» входить до складу програмної документації на програму, що описує результати аналізу продуктивності та ефективності ігрових рушіїв Unity та Unreal Engine.

У даному документі представлений текст додатку. Додаток написаний на мові C# з використанням Visual Studio Code та Blue Print вбудований в Unreal Engine.

44165850.01521–01 12–01

ЗМІСТ

1 Текст програми 2
1.1 Текст файлу PhysicsSpawner.cs 2
1.2 Текст файлу FPSCounter.cs 3

1 ТЕКСТ ПРОГРАМИ

1.1 Текст файлу PhysicsSpawner.cs

```
using UnityEngine;

public class PhysicsSpawner : MonoBehaviour
{
    [Header("Object Settings")]
    public GameObject prefab;
    public int countX = 20;
    public int countY = 20;
    public int countZ = 20;
    public float spacing = 1.1f;

    [Header("Spawn Settings")]
    public Vector3 startPos = new Vector3(0, 100, 0);

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.S))
        {
            for (int x = 0; x < countX; x++)
            {
                for (int y = 0; y < countY; y++)
                {
                    for (int z = 0; z < countZ; z++)
                    {
                        Vector3 pos = startPos + new Vector3(x * spacing, y * spacing, z * spacing);
                        Instantiate(prefab, pos, Quaternion.identity);
                    }
                }
            }
        }
    }
}
```

1.2 Текст файлу FPSCounter.cs

```
using UnityEngine;

public class FPSCounter : MonoBehaviour
{
   public float updateInterval = 0.5f;

   public float current;
   public float average;
   public float min = float.MaxValue;
   public float max = 0f;

   float accum;
   int frames;
   float timeLeft;

   void Start()
   {
      timeLeft = updateInterval;
   }

   void Update()
   {
      float fps = 1f / Time.unscaledDeltaTime;
      current = fps;

      // обновляем min / max
      if (fps < min) min = fps;
      if (fps > max) max = fps;

      // считаем средний за интервал
      accum += fps;
      frames++;
      timeLeft -= Time.unscaledDeltaTime;

      if (timeLeft <= 0f)
      {
         average = accum / Mathf.Max(frames, 1);

         timeLeft = updateInterval;
         accum = 0f;
         frames = 0;
      }

      if (Input.GetKeyDown(KeyCode.R))
      {
         min = float.MaxValue;
         max = 0f;
      }
   }

   void OnGUI()
   {
      GUI.Label(new Rect(10, 10, 260, 22), $"FPS: {current:0.0}");
      GUI.Label(new Rect(10, 30, 260, 22), $"AVG: {average:0.0}");
      GUI.Label(new Rect(10, 50, 260, 22), $"MIN: {min:0.0}");
      GUI.Label(new Rect(10, 70, 260, 22), $"MAX: {max:0.0} (R — reset)");
```

```
  }
}
```

ДОДАТОК В

Керівництво користувача

ЗАТВЕРДЖУЮ
Проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

Програма дослідження продуктивності ігрових рушіїв Unity та Unreal Engine
Керівництво користувача
44165850.01521 – 01 І3 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олександр ІВАНОВ
Виконавець
_____Ярослав ГРИВА
Нормоконтролер
_____Світлана ВОЛКОВА

2026

ЗАТВЕРДЖЕНО
44165850.01521-01 І3 01

Програма дослідження продуктивності ігрових рушіїв Unity та Unreal Engine
Керівництво користувача
44165850.01521 – 01 І3 01
Листів 9

2025

АНОТАЦІЯ

Документ 1116130.01521 – 01 І3 01 «Дослідження продуктивності ігрових рушіїв Unity та Unreal Engine».

Програми написані на мові C#, з використанням програмноого середовищі Visual Studio Code та вбудовану функцію Blueprint вбудовану в Unreal Engine.

44165850.01521-01 І3 01

ЗМІСТ

1 Введення 2
2 Призначення та умови застосування 3
3 Підготовка до роботи 4
4 Керівництво користувача 5
5 Аварійні ситуації 7
6 Рекомендації щодо застосування 8

1 ВВЕДЕННЯ

Дані, отримані під час проведення дослідження «Дослідження продуктивності ігрових рушіїв Unity та Unreal Engine», слугують практичним підтвердженням наведених у роботі результатів та висновків. Подані матеріали дозволяють більш комплексно й обґрунтовано здійснити вибір ігрового рушія залежно від поставлених завдань, типу навантаження та вимог до продуктивності.

Аналіз цих даних сприяє прийняттю технічно коректних рішень під час розробки ігрових та інтерактивних проектів, що у свою чергу, дає змогу підвищити ефективність оптимізації та загальну продуктивність кінцевого продукту.

2 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Результати даного дослідження призначені для використання під час проектування, розробки та оптимізації ігрових і інтерактивних застосунків, створених із використанням ігрових рушіїв Unity та Unreal Engine. Отримані дані можуть бути корисними для розробників ігор, студентів технічних спеціальностей, а також для фахівців, які здійснюють вибір рушія залежно від вимог до продуктивності та візуальної складності проекту.

Матеріали дослідження доцільно застосовувати за таких умов:

– під час вибору ігрового рушія для проектів із великою кількістю об’єктів на сцені;

– при розробці застосунків із інтенсивним використанням фізичної симуляції;

– у процесі оптимізації продуктивності для систем із обмеженими апаратними ресурсами;

– для порівняльного аналізу продуктивності рушіїв за однакових умов тестування.

Застосування наведених результатів є найбільш коректним за умови подібної структури сцен, налаштувань графіки та характеру навантаження, оскільки зміна цих параметрів може впливати на кінцеві показники продуктивності.

3 ПІДГОТОВКА ДО РОБОТИ

– **Встановлення програмного забезпечення**

Завантажити ігрові рушії **Unity** та **Unreal Engine** з офіційних вебсайтів розробників. Після встановлення необхідно пройти процедуру реєстрації та авторизації в обох середовищах для отримання доступу до повного функціоналу.

– **Створення проєктів**

Запустити Unity та Unreal Engine і створити **нові проєкти** з використанням стандартних шаблонів (3D-проєкт). Усі початкові налаштування мають залишатися типовими, без додаткової оптимізації чи постобробки.

– **Підготовка тестових сцен**

У кожному рушії створити **ідентичні тестові сцени** з мінімальним використанням об’єктів. Сцени повинні містити лише базові елементи (камера, освітлення та примітиви), необхідні для проведення фізичної симуляції та візуалізації.

– **Забезпечення чистоти експерименту**

Під час проведення експерименту дозволяється змінювати **лише одну змінну за раз** (наприклад, кількість об’єктів або наявність фізики). Усі інші параметри сцени, налаштування графіки та апаратні умови мають залишатися незмінними, що забезпечує об’єктивність і коректність отриманих результатів.

4 КЕРІВНИЦТВО КОРИСТУВАЧА

1. Запуск проєкту

1. Запустіть ігровий рушій **Unity** або **Unreal Engine**.

2. Відкрийте підготовлений тестовий проєкт.

3. Переконайтесь, що активна **тестова сцена** з мінімальним набором об’єктів.

2. Початковий замір продуктивності

1. Запустіть сцену в режимі відтворення (Play / Play In Editor).

2. Зафіксуйте початкові показники:

- середній FPS;
- мінімальний FPS;
- час рендерингу кадру (ms).

3. Повторіть замір 3–5 разів.

4. Обчисліть середнє значення показників.

3. Додавання об’єктів дослідження

1. Додайте на сцену **1000 однакових об’єктів**.

2. Рівномірно розмістіть об’єкти в межах зони огляду камери.

3. Запустіть сцену та зафіксуйте показники FPS і ms.

4. Збережіть результати.

5. Повторіть дії для **2000, 3000, 4000 та 5000 об’єктів**.

4. Проведення повторних вимірів

1. Для кожного тесту виконайте не менше **3 повторів**.

2. Записуйте всі результати в таблицю.

3. Обчисліть середні та мінімальні значення FPS і ms.

6. Фіксація результатів

1. Збережіть отримані дані у вигляді таблиць.

2. Зафіксуйте візуальний стан сцени (скріншоти).

3. Позначте появу візуальних артефактів або просідань продуктивності.

7. Завершення експерименту

1. Закрийте сцену без збереження змін (за потреби).

2. Повторіть усі етапи в іншому рушії за **ідентичних умов**.

3. Порівняйте отримані результати.

5 АВАРІЙНІ СИТУАЦІЇ

При швидкому додаванні на сцену великої кількости обектів рушій може дати помилку через навантаження .

При проведенні експериментів з увімкненими фоновими програмами результати можуть сильно змінюватися .

Якщо програмний засіб під час роботи буде поводити некоректно чи із помилкою, необхідно перезапустити додаток для продовження роботи.

6 РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

Отримані результати доцільно застосовувати на етапах вибору ігрового рушія, планування архітектури сцени та оптимізації продуктивності.

Для проєктів із великою кількістю однотипних об’єктів та активною фізичною взаємодією рекомендується використовувати Unity, оскільки він забезпечує більш стабільну роботу та рівномірний спад продуктивності.

Unreal Engine доцільно застосовувати у проєктах, орієнтованих на високу візуальну якість і кінематографічну графіку, за умови обмеженого та продуманого використання фізики.

Дотримання цих рекомендацій дозволить зменшити навантаження на систему, підвищити стабільність роботи застосунку та досягти оптимального балансу між якістю зображення і продуктивністю.