Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту
Завідувач кафедри комп’ютерних наук,
доктор технічних наук, професор
_____ Андрій БОНДАРЧУК
«____» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»
спеціальність 122 Комп’ютерні науки
освітня програма 122.00.01 Інформатика

Тема: 3D-МОДЕЛЮВАННЯ ІГРОВОЇ ЛОКАЦІЇ З ІНТЕРАКТИВНИМИ ЕЛЕМЕНТАМИ У СЕРЕДОВИЩІ GODOT

Виконав
студент групи ІНб-2-22-4.0д
Марченко Даніїл
Максимович
_____
(підпис)

Науковий керівник
кандидат педагогічних наук,
доцент
Бойко М.А.
_____
(підпис)

Київ − 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри комп’ютерних наук,
доктор технічних наук, професор
_____ Андрій БОНДАРЧУК
«31» жовтня 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
«3D-моделювання ігрової локації з інтерактивними елементами у середовищі Godot»

Виконавець - студент групи ІНб-2-22-4.0д,
спеціальності 122 Комп'ютерні науки,
освітньої програми 122.00.01 Інформатика
Марченко Данііл Максимович

1. Вихідні дані: Офіційна технічна документація рушія Godot та програмного забезпечення для моделювання Blender. Публікації та наукові статті за напрямом просторового моделювання та алгоритмічної оптимізації.

2. Основні завдання: Здійснити огляд публікацій за темою роботи. Обґрунтувати мету, об'єкт, предмет, методи дослідження та засоби дослідження. Підготувати матеріали до розділів роботи. Створити оптимізовані 3D-моделі. Розробити діючий прототип локації у середовищі Godot та алгоритми взаємодії. Провести аналіз результатів дослідження та скласти висновки. Оформити роботу відповідно до затверджених вимог. Підготувати тези доповіді за темою роботи.

3. Пояснювальна записка: Обсяг приблизно 49 сторінок формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.

4. Графічні матеріали: Комп'ютерна презентація.

5. Додатки: Лістинг програмного коду (скриптів взаємодії).

6. Строк подання роботи на кафедру: «1» червня 2026 р.

Науковий керівник:
к.п.н., доцент
_____ Бойко М.А.
«31» жовтня 2025 р.

Виконавець:
_____ Марченко Д.М.
«31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Етапи виконання роботи	Термін виконання	Примітка
1	Затвердження теми кваліфікаційної роботи бакалавра	31.10.25	Виконано
2	Аналіз проблеми, вивчення аналогів, формування цілей і завдань	01.11.25 - 11.01.26	Виконано
3	Аналіз предметної області, існуючих ігрових рушіїв та їх вимог	26.01.26 - 15.03.26	Виконано
4	Дослідження методів модульного проєктування та фізично коректного рендерингу	16.03.26 - 31.03.26	Виконано
5	Розробка модульних тривимірних об'єктів у Blender та програмна реалізація інтерактивних механік у середовищі Godot	01.04.26 - 15.04.26	Виконано
6	Тестування продуктивності віртуального простору	16.04.26 - 30.04.26	Виконано
7	Впровадження алгоритмів просторової оптимізації	01.05.26 - 15.05.26	Виконано
8	Підготовка пояснювальної записки, графічних матеріалів, додатків.	25.05.25 - 12.06.26	Виконано
9	Захист кваліфікаційної роботи	16.06.26	

«Затверджую»

Науковий керівник

к.п.н., доцент, Бойко М.А.

Індивідуальний план склав

Марченко Д.М.

РЕФЕРАТ

Кваліфікаційна робота: 49с., 13 рис., 7 табл., 3 формули, 5 лістингів, 30 джерел, 3 додатки.

Актуальність теми: розвиток індустрії інтерактивних розваг вимагає створення переконливих просторів, де локація виступає повноцінним інструментом просторової оповіді. Попри наявність потужних комерційних платформ, існує значний попит на відкриті рушії з прозорою архітектурою. Відповідно, дослідження методів розробки комплексних просторів у відкритих середовищах є своєчасним науковим завданням.

Об'єкт дослідження: процес інженерного проєктування, алгоритмічної оптимізації та розробки віртуальних ігрових просторів.

Предмет дослідження: інструментарій, патерни та методи модульного проєктування інтерактивних 3D-локацій у середовищі Godot

Мета роботи: аналіз інструменті розробки та програмної реалізація тривимірної ігрової локації з системою інтерактивних елементів у середовищі Godot.

Методи дослідження: системний аналіз для порівняння програмного забезпечення; просторове полігональне моделювання та процедурне текстурування; об'єктно-орієнтоване програмування для реалізації алгоритмів поведінки; профілювання продуктивності рендерингу в реальному часі. .

Основні результати: проаналізовано повний конвеєр розробки та обґрунтовано практичну доцільність застосування модульного дизайну. Створено оптимізовані 3D-моделі з фізично коректними матеріалами. Розроблено інтерактивні механіки на базі подієво-орієнтованої архітектури, що забезпечують стабільну взаємодію з середовищем без критичного падіння продуктивності.

Ключові слова: тривимірне моделювання, ігровий рушій, ігрова локація, інтерактивність, алгоритмічна оптимізація, фізично коректний рендеринг, модульний дизайн.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

Скорочення/Термін	Розшифровка та визначення
BRDF/Bidirectional Reflectance Distribution Function	Двопроменева функція розподілу відбивної здатності - базова математична функція у тривимірній графіці, яка визначає та описує, як саме світлові промені відбиваються від непрозорої поверхні матеріалу.
GDScript	Динамічно типізована, об'єктно-орієнтована мова програмування з синтаксисом, подібним до Python, створена спеціально для рушія Godot.
GGX	Математична функція розподілу нормалей мікрорельєфу в конвеєрі PBR, яка відповідає за розрахунок форми, інтенсивності та розмитості світлового відблиску на поверхні об'єкта
glTF	Відкритий стандарт формату файлів для ефективного зберігання та обміну даними тривимірних сцен і моделей.
PBR/Physically Based Rendering	Фізично коректний рендеринг підхід у комп'ютерній графіці, що дозволяє відображати об'єкти з високою достовірністю шляхом математичного моделювання фізичної поведінки потоку світла у реальному світі.
Vulkan	Сучасний низькорівневий кросплатформний графічний інтерфейс, що забезпечує високоефективний прямий доступ до ресурсів відеокарти.

ЗМІСТ

ВСТУП

Сучасна індустрія інтерактивних розваг, освітніх симуляцій та розробки віртуальних просторів є однією з найбільш динамічних і технологічно містких сфер світового ринку. Згідно з актуальними науковими дослідженнями в галузі комп'ютерних наук, попит на якісний візуальний контент та глибоке занурення користувача у віртуальні світи безперервно зростає [1, с. 3549 - 3550]. Сучасна ігрова локація еволюціонувала з простого набору геометричних перешкод у складну систему, що органічно інтегрує архітектуру, освітлення та логіку. Ключовим елементом успіху будь-якого проєкту є грамотно спроєктоване середовище, яке не лише виконує роль статичної декорації, але й виступає як самостійний засіб невербальної комунікації. Локація активно взаємодіє з гравцем, розповідаючи історію та задаючи контекст через візуальне оформлення простору.

На сьогоднішній день індустрія пропонує розробникам значну кількість потужних платформ для розробки просторових рішень. Проте провідні комерційні рушії часто характеризуються високим порогом входження, вимогливістю до апаратних ресурсів робочої станції та жорсткими умовами ліцензування, що може створювати бар'єри для незалежних розробників та академічних дослідників [2, с. 32 - 38]. У цьому контексті особливої уваги набуває інструментарій, що поширюється за моделлю відкритого вихідного коду. Рушій Godot Engine інтегрується у професійний робочий процес завдяки своїй вузловій архітектурі, оптимізованому розміру та впровадженню передових графічних технологій рендерингу [3, с. 15 - 18]. Водночас, деталізовані академічні та практичні методики створення комплексних просторів із системою обробки подій саме у цьому середовищі потребують глибокого вивчення та систематизації, що підтверджує високу актуальність обраної теми.

Метою кваліфікаційної роботи є розробка та програмна реалізація алгоритмічно оптимізованої тривимірної ігрової локації з системою

інтерактивних механік у середовищі Godot для демонстрації та наукового аналізу можливостей рушія у створенні сучасних віртуальних рівнів.

Для досягнення поставленої мети було сформульовано наступні завдання:

- проаналізувати існуючі технологічні підходи до генерації просторової графіки та обробки інтерактивних подій у сучасному програмному забезпеченні;
- здійснити об'єктивний порівняльний аналіз провідних графічних платформ та науково обґрунтувати вибір інструментів розробки для реалізації проєкту;
- дослідити принципи модульної архітектури, математичні основи фізично коректної візуалізації та методи просторової оптимізації;
- створити необхідні тривимірні моделі, оптимізувати їх полігональну сітку та розробити процедурні матеріали у спеціалізованому програмному забезпеченні;
- виконати експорт активів, налаштувати сцену, фізичні реакції, параметри середовища та глобального освітлення.
- запрограмувати поведінку інтерактивних об'єктів та розробити контролер гравця за допомогою вбудованої скриптової мови, інтегрувавши патерни проєктування для забезпечення модульності коду.

Об'єктом дослідження є процес інженерного проєктування, алгоритмічної оптимізації та програмної розробки віртуальних інтерактивних просторів.

Предметом дослідження виступають спеціалізований інструментарій, математичні методи тривимірного моделювання та архітектурні програмні рішення для створення інтерактивності у середовищі Godot Engine.

У процесі виконання кваліфікаційної роботи застосовувалися методи аналізу фахових наукових джерел та офіційної документації для формулювання системних вимог; методи оптимізації та маніпуляції полігональними сітками для формування візуальної компоненти; методи

системного та об'єктно-орієнтованого програмування для реалізації алгоритмів просторової поведінки; емпіричні методи тестування та профілювання продуктивності алгоритмів рендерингу.

Практичне значення одержаних результатів полягає у тому, що розроблена програмна система локації має високий ступінь готовності і може слугувати повноцінним базовим шаблоном для подальшої комерційної розробки проєктів жанру пригодницьких ігор. Крім того, зібрана та систематизована у роботі інформація може застосовуватися як структурований посібник для студентів під час вивчення дисциплін, пов'язаних із просторовою розробкою та комп'ютерною графікою.

РОЗДІЛ 1
ЗАГАЛЬНИЙ АНАЛІЗ ЗАСОБІВ РОЗРОБКИ ІГРОВИХ ЛОКАЦІЙ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд сучасних тенденцій у проєктуванні ігрових світів

Розробка тривимірних ігрових рівнів являє собою багаторівневий процес, який поєднує творче бачення, архітектурну структуру та алгоритмічну оптимізацію ресурсів. У сучасних комп'ютерних науках ігрові середовища давно переросли роль звичайних статичних декорацій. Сьогодні локація виступає динамічним інструментом взаємодії, що комунікує з гравцем завдяки продуманому дизайну та візуальним акцентам. Замість прямого тексту чи аудіопідказок, контекст світу передається опосередковано: через освітлення, специфічне розміщення предметів чи сліди минулих подій. Дослідники просторового наративу підтверджують, що саме така невербальна подача інформації найкраще сприяє глибокому зануренню та створенню переконливої ілюзії присутності у віртуальному світі [4, с. 15 - 22].

Для забезпечення ефективного функціонування таких систем, до сучасних локацій висувається низка жорстких технічних та естетичних вимог:

1. Візуальна достовірність та фізична коректність. Сучасна графіка вимагає імплементації матеріалів на основі алгоритмів PBR. В їх основі лежить математична модель Кука-Торранса та BRDF. Ці моделі математично точно симулюють поведінку світла при взаємодії з поверхнею: наприклад, функція GGX розраховує розсіювання променів на мікрорельєфі, а рівняння Френеля визначає здатність відбивати світло залежно від кута зору [5, с. 531 - 535; 6, с. 195 - 197; 7, с. 82 - 84]. Завдяки математичному обмеженню, за яким сума відбитого та розсіяного світла не може перевищувати енергію початкового променя, цей підхід унеможливлює неприродне пересвічення матеріалів та гарантує реалістичний вигляд металів, пластику чи дерева за будь-яких умов динамічного освітлення.

2. Алгоритмічна оптимізація та продуктивність. Підтримання стабільної та високої частоти кадрів є абсолютним пріоритетом для забезпечення комфортної взаємодії з програмою. Це вимагає від розробника ретельного планування топології кожної моделі, контролю за кількістю викликів малювання з боку центрального процесора та використання адаптивних методів рендерингу [8, с. 815 - 820]. Цей аспект набуває критичної ваги в умовах незалежної розробки або при таргетуванні проєкту на платформи з обмеженими апаратними ресурсами.

3. Глибока інтерактивність простору. Віртуальне середовище повинно адекватно і передбачувано реагувати на дії гравця. Наявність функціональних об'єктів створює стійке відчуття живого світу та значно розширює простір можливих алгоритмів проходження рівня.

4. Ергономічна просторова навігація. Архітектура рівня має бути спроєктована таким чином, щоб рух гравця був інтуїтивно зрозумілим. Використовуючи контрастне освітлення, колірні акценти та геометричні направляючі лінії, дизайнер рівня скеровує увагу користувача у правильному напрямку, уникаючи необхідності розміщення штучних графічних покажчиків, які руйнують атмосферу [9, с. 38 - 39].

Окремої уваги серед сучасних тенденцій заслуговує повноцінний перехід індустрії на підхід модульного проєктування. Замість створення унікальної монолітної геометрії для кожної сцени чи приміщення, середовище конструюється з набору заздалегідь підготовлених, стандартизованих і взаємозамінних блоків, що базується на принципах компонентної збірки та повторного використання активів [10, с. 112 - 115] . Такий інженерний підхід мінімізує обсяг оперативної пам'яті, необхідний для завантаження текстур та вершин, значно прискорює процес ітеративного тестування прототипів та дозволяє гнучко вносити масштабні зміни в архітектуру локації на будь-якому етапі розробки без необхідності повного перероблення графічних активів. Саме модульний підхід є базовим стандартом для створення масштабних інфраструктурних об'єктів у віртуальних світах.

Традиційний підхід до моделювання локацій передбачав створення унікальної монолітної геометрії для кожної сцени. Головним недоліком такого методу є експоненційне зростання споживання оперативної пам'яті та значні часові витрати на внесення ітеративних змін. На противагу йому, сучасна індустрія перейшла до модульного проєктування. Проте цей метод створює новий виклик: ризик візуальної монотонності та розпізнаваності патернів (ефект «шахової дошки») [11].

Отже, у даній роботі досліджується баланс між продуктивністю конвеєра рендерингу та якістю віртуального середовища шляхом комбінування стандартизованих модулів із методами просторового маскування.

1.2 Аналіз ігрових рушіїв для створення тривимірної графіки

Щоб ефективно реалізувати розробку та забезпечити належну якість кінцевого проєкту, критично важливо підібрати найбільш відповідний ігровий рушій. Сучасна індустрія пропонує розробникам три провідні системи: Unreal Engine, Unity та Godot Engine. Зазначені платформи відрізняються власними архітектурними підходами, вбудованими бібліотеками та інструментарієм, що суттєво визначає темпи створення прототипів, підходи до написання коду та ефективність роботи готової програми.

Платформа Unreal Engine від корпорації Epic Games історично є індустріальним стандартом для розробки високобюджетних ігор. Вона пропонує безпрецедентну якість графіки прямо з базових налаштувань, інтегровану потужну систему візуального скриптування та передові інструменти для роботи з освітленням і геометрією кінематографічної якості. Проте для індивідуальних розробників, студентських проєктів та невеликих незалежних команд цей інструмент часто виявляється надлишковим. Це зумовлено високим споживанням апаратних ресурсів під час роботи редактора, гігантським обсягом базових файлів проєкту та високим порогом входження при необхідності глибокого програмування логіки на мові C++.

Крім того, ліцензійна угода передбачає відрахування після перевищення певного порогу доходів.

Інша популярна платформа, Unity, вирізняється гнучкістю, підтримкою величезної кількості цільових платформ та наявністю колосальної бібліотеки готових ресурсів. Рушій використовує поширену мову C#, що робить його привабливим для багатьох програмістів. Однак в останні роки фахівці індустрії відзначають суттєві архітектурні ускладнення всередині самої платформи. Зокрема, фрагментація графічних конвеєрів призводить до постійних проблем сумісності компонентів та матеріалів.

Крім того, часті та суперечливі зміни в політиці ліцензування компанії, пов'язані з введенням різноманітних моделей підписки та плати за встановлення, змушують розробників шукати більш стабільні та прозорі альтернативи.

У цьому конкурентному середовищі стрімко набирає популярність Godot Engine - сучасна система розробки з відкритим вихідним кодом, що розповсюджується за дозвільною вільною ліцензією. Перехід платформи на четверте покоління ознаменувався фундаментальною перебудовою графічного ядра. Рушій отримав повноцінну підтримку сучасного низькорівневого графічного інтерфейсу Vulkan та впровадження системи динамічного глобального освітлення, що дозволило йому конкурувати з комерційними гігантами у якості відображення тривимірної графіки [12, с. 42 - 45].

Для об'єктивного оцінювання платформ доцільно звернутися до порівняльної характеристики ключових показників, що наведена у додатку А.

Системний аналіз наведених даних виявляє суттєві недоліки провідних комерційних платформ у контексті індивідуального прототипування та академічних досліджень. Unreal Engine 5, попри свій статус індустріального стандарту, характеризується надмірним споживанням апаратних ресурсів та великим обсягом базових файлів проєкту, що робить його архітектуру надлишковою для створення оптимізованих локальних просторів. Unity, зі

свого боку, має критичні проблеми з фрагментацією графічних конвеєрів, що призводить до постійних конфліктів сумісності матеріалів. Саме тому в даній роботі обґрунтовується вибір Godot Engine 4. Його архітектура дозволяє уникнути програмної перевантаженості проєкту, а інтеграція інтерфейсу Vulkan дає змогу реалізувати алгоритми PBR без втрати контролю над системними ресурсами.

1.3 Особливості архітектури середовища Godot та вбудованої мови програмування

Більшість сучасних ігрових рушіїв спираються на жорстку компонентну модель або складне об'єктно-орієнтоване програмування, що часто призводить до проблеми сильної зв'язності коду між об'єктами сцени та ускладнює їх масштабування [13, с. 211 - 215].

У цій роботі як вирішення даної алгоритмічної проблеми розглядається та застосовується альтернативний програмний патерн - ієрархічна вузлова архітектура Godot.

Фундаментальною відмінністю обраної платформи від найближчих конкурентів є повна відмова від традиційного для ігрової індустрії патерну на користь унікальної ієрархічної структури вузлів та сцен [14]. У цій архітектурній парадигмі будь-який об'єкт від найпростішої базової геометричної фігури, джерела світла чи звуку до комплексного повноцінного ігрового рівня з власною логікою є вузлом або структурованим деревом вузлів, які інкапсулюються у єдине глобальне поняття «сцена».

Ця парадигма об'єктно-орієнтованого дизайну та композиції забезпечує розробнику потужний механізм створення екземплярів. Складний об'єкт, наприклад, двері з налаштованою анімацією, звуковими ефектами та скриптом взаємодії, конструюється та тестується лише один раз в ізольованому середовищі. Після цього він може бути багаторазово відтворений у глобальному просторі будь-якої іншої сцени зі збереженням стійкого зв'язку з оригінальним шаблоном. Будь-які структурні модифікації чи виправлення

помилок у базовому шаблоні миттєво та безпомилково реплікуються на всі його екземпляри у проєкті. Ця функціональність є важливою вимогою при імплементації модульного дизайну, де кількість однотипних об'єктів може вимірюватися тисячами.

Програмування поведінкової логіки у середовищі переважно здійснюється за допомогою високорівневої мови програмування GDScript, яка була спеціально спроєктована з урахуванням архітектури рушія. Мова має чіткий синтаксис, орієнтований на обов'язкові відступи, що робить код читабельним та лаконічним. Її найсильнішою технологічною перевагою є безшовна інтеграція з внутрішніми структурами рушія на рівні пам'яті. Програміст має можливість безпосередньо та безпечно звертатися до будь-яких властивостей просторових вузлів, динамічно створювати нові зв'язки під час виконання програми та реалізовувати надійний патерн через систему вбудованих подій. Ця спеціалізована мова надає можливість швидко прописувати складну логіку взаємодії без необхідності постійної перекомпіляції ядра платформи, що прискорює ітеративний процес розробки та тестування гіпотез.

Для забезпечення безпеки типів даних мова підтримує поступову типізацію, що дозволяє виявляти помилки на етапі написання коду.

Окремим вагомим аргументом на користь обраної платформи є оновлений конвеєр інтеграції зовнішніх цифрових ресурсів у четвертій версії. Він забезпечує глибоку та пряму підтримку сучасних відкритих форматів передачі даних. Зокрема, автоматизований процес парсингу формату glTF дозволяє безперешкодно переносити з графічних редакторів налаштування ієрархії об'єктів, складні параметри матеріалів та попередньо згенеровані базові форми колізій. Це створює єдиний, надійний та нерозривний робочий процес між програмою тривимірного моделювання Blender та цільовим ігровим середовищем, зводячи до мінімуму рутинні операції з повторного налаштування імпортованих ресурсів [15].

1.4 Постановка задачі на кваліфікаційну роботу

З огляду на результати детального аналізу предметної області, сучасних стандартів графіки та особливостей обраного інструментарію, висувається комплексне завдання: спроєктувати, алгоритмічно оптимізувати та програмно реалізувати прототип інтерактивної локації в закритому просторі.

Розширена технічна специфікація проєкту включає наступні обов'язкові етапи та компоненти:

1. Проєктування архітектури та геометрії простору: створення повноцінного модульного набору архітектурних конструктивних елементів (сегментів стін, перекриттів, опорних колон, підлогових панелей) та декоративних об'єктів для формування переконливого та деталізованого середовища з дотриманням єдиної метричної сітки [16, с. 115 - 118].

2. Реалізація системи керування та навігації: програмування кінематики головного персонажа для забезпечення плавного переміщення у тривимірному просторі та реалізації огляду навколишнього середовища за допомогою віртуальної камери, закріпленої на рівні очей.

3. Імплементація систем об'єктної взаємодії: розробка алгоритмів на основі методу RayCast3D та патерну проєктування «Спостерігач», що дозволяють системі розпізнавати фокус погляду користувача на цільовий об'єкт та активувати його закладені функції за допомогою децентралізованої системи сигналів.

4. Фізична симуляція середовища: налаштування математично коректних розрахунків маси, сили тяжіння, імпульсу та сили тертя для рухомих інтерактивних об'єктів, щоб забезпечити їх природну реакцію на зіткнення з гравцем або іншими елементами сцени.

5. Інформаційний супровід користувача: проєктування адаптивного та мінімалістичного графічного інтерфейсу, що своєчасно інформує гравця про доступні дії у зоні його видимості.

6. Системна оптимізація: застосування алгоритмів відсікання невидимої геометрії та динамічної генерації рівнів деталізації для забезпечення стабільного показника частоти кадрів.

7. Для виконання художньої та геометричної частини роботи обрано відкритий програмний комплекс тривимірного моделювання Blender. Для інтеграції ресурсів, налаштування глобального рендерингу, фізики та безпосереднього програмування логіки застосовуватиметься платформа Godot Engine 4.

Висновки до розділу 1

Перший розділ присвячено аналізу предметної області, сучасних тенденцій проєктування ігрових світів та обґрунтуванню вибору інструментарію розробки. За результатами огляду сформульовано такі висновки:

1. Аналіз сучасних тенденцій у проєктуванні ігрових світів (підрозділ 1.1) показав, що ключовими вимогами до віртуальних середовищ є фізична коректність візуалізації (на базі PBR-матеріалів), алгоритмічна оптимізація та інтерактивність простору. Встановлено, що перехід до модульного проєктування мінімізує споживання оперативної пам'яті, проте вимагає застосування методів просторового маскування для запобігання ефекту візуальної монотонності.

2. Порівняльний аналіз ігрових рушіїв Unreal Engine, Unity та Godot Engine (підрозділ 1.2, додаток А) засвідчив, що провідні комерційні платформи характеризуються надмірним споживанням апаратних ресурсів або архітектурними ускладненнями, що робить їх надлишковими для академічних досліджень та індивідуального прототипування. Це підтвердило доцільність вибору Godot Engine 4, який завдяки відкритій ліцензії, оптимізованому розміру та підтримці графічного інтерфейсу Vulkan дозволяє реалізувати передові графічні технології без перевантаження системи.

3. Дослідження архітектури середовища Godot (підрозділ 1.3) довело, що використання унікальної ієрархічної вузлової структури та механізму створення екземплярів ідеально підходить для реалізації модульного дизайну об'єктів. Інтеграція відкритого формату обміну даними glTF та використання спеціалізованої високорівневої мови GDScript забезпечують нерозривний, надійний робочий конвеєр від етапу тривимірного моделювання в Blender до програмування просторової логіки безпосередньо в рушії.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ І ВИБІР ТЕОРЕТИЧНИХ ТА ЕКСПЕРИМЕНТАЛЬНИХ МЕТОДІВ ДОСЛІДЖЕННЯ

2.1 Принципи модульного проєктування віртуальних середовищ

Проєктування масштабних ігрових просторів потребує оптимізованих методів створення моделей. Спроби розробляти кожну кімнату чи локацію як один цілісний, унікальний об'єкт призводять до невиправданих витрат часу та надмірного споживання оперативної пам'яті комп'ютерної системи. Вирішенням цієї проблеми стає модульний підхід, який передбачає розбиття великих архітектурних форм на уніфіковані базові блоки. Ці компоненти можна багаторазово використовувати та складати в різноманітні конфігурації, зберігаючи при цьому загальну стилістику та високу якість графіки [10, с. 142 - 145; 11].

У межах даної кваліфікаційної роботи для практичної реалізації локації було розроблено власний стандартизований набір базових архітектурних модулів. До нього увійшли: прямі сегменти стін, кутові переходи, підлогові плити різного формату, стельові перекриття та секції з дверними отворами (рис. 2.1).

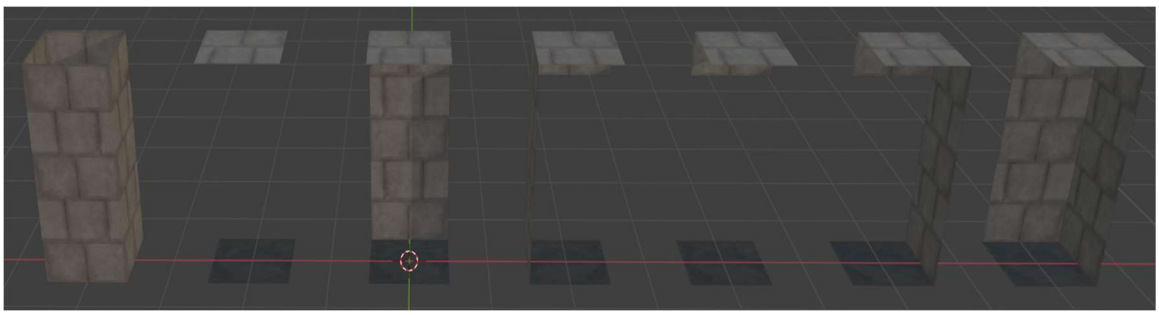


Рисунок 2.1. Базовий набір розроблених архітектурних модулів локації

Комбінування цих елементів у єдину структуру простору реалізовано за допомогою спеціалізованого вузла Gridmap в ігровому рушії Godot. Цей інструмент працює за принципом тривимірної тайлової карти, дозволяючи

розташовувати екземпляри модулів із жорсткою прив'язкою до комірок визначеної просторової сітки. Застосування цього методу забезпечило швидке формування чорнового прототипу рівня для перевірки масштабу та навігації без ризику виникнення геометричних розривів (рис. 2.2).

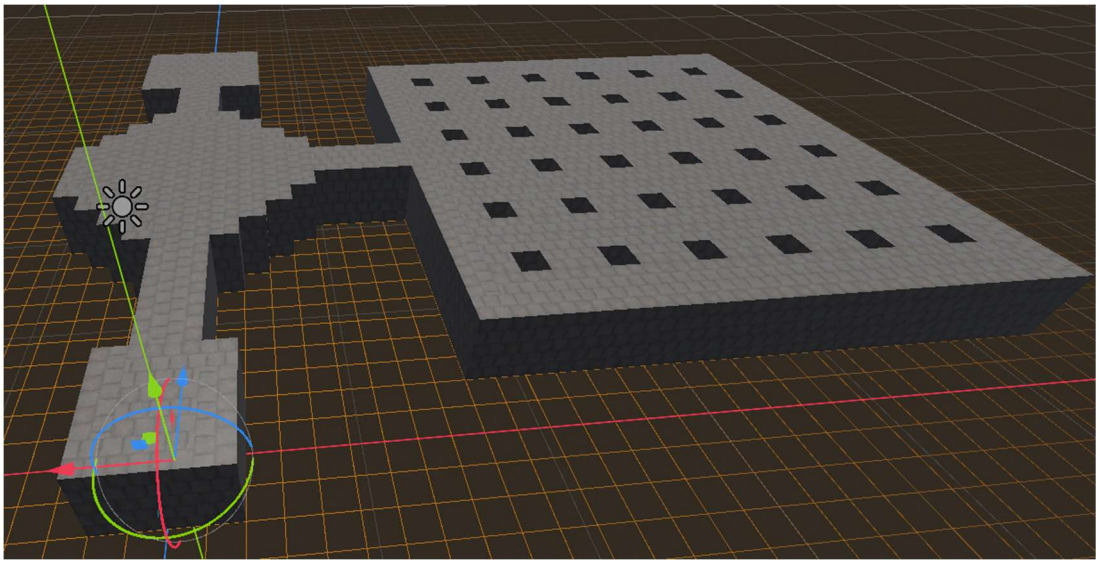


Рисунок 2.2. Прототип ігрової локації, зібраний із модулів за допомогою системи GridMap

Фундаментальним правилом успішного застосування модульного підходу є суворе дотримання єдиної просторової прив'язки. Кожен структурний елемент повинен мати габарити, які є строго кратними заздалегідь визначеному базовому кроку у всіх трьох вимірах. Це математично гарантує абсолютно безшовне стикування полігональних країв без виникнення видимих щілин, накладання текстур або небажаних перетинів геометрії, що руйнують ілюзію цілісності об'єкта [17, с. 145 - 147].

Другим критично важливим технічним аспектом є розташування локального центру координат кожного об'єкта. Для вертикальних архітектурних модулів опорну точку найбільш доцільно розміщувати у центрі на рівні підлоги. Це дозволяє розробнику виконувати точну автоматичну математичну прив'язку до глобальної тривимірної сітки ігрового рушія під час

позиціонування, виключаючи людський фактор та помилки ручного розміщення.

Схематичне представлення прив’язки опорної точки для розробленого набору модулів наведено на рис. 2.3. Опорна точка розташована у нульовій координаті локального простору моделі, що відповідає геометричному центру нижньої площини об'єкта. Таке позиціонування є технічно виправданим для коректної роботи алгоритмів дискретної прив’язки, оскільки дозволяє автоматично вирівнювати об'єкти за заданим кроком тривимірної сітки. Це зменшує появу випадкових просторових зсувів через людський фактор під час ітеративного заповнення сцени та забезпечує щільне стикування геометрії.

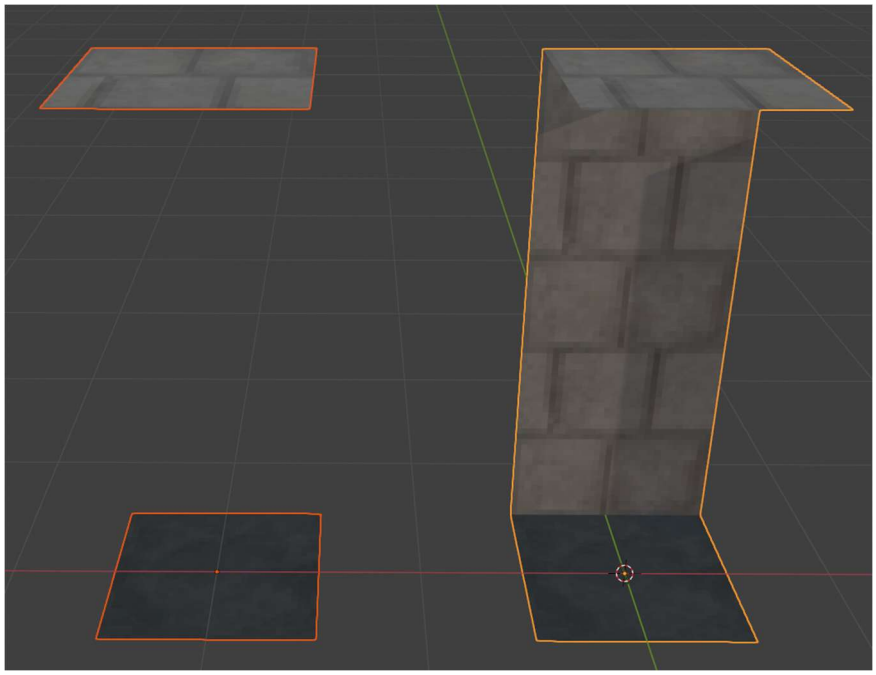
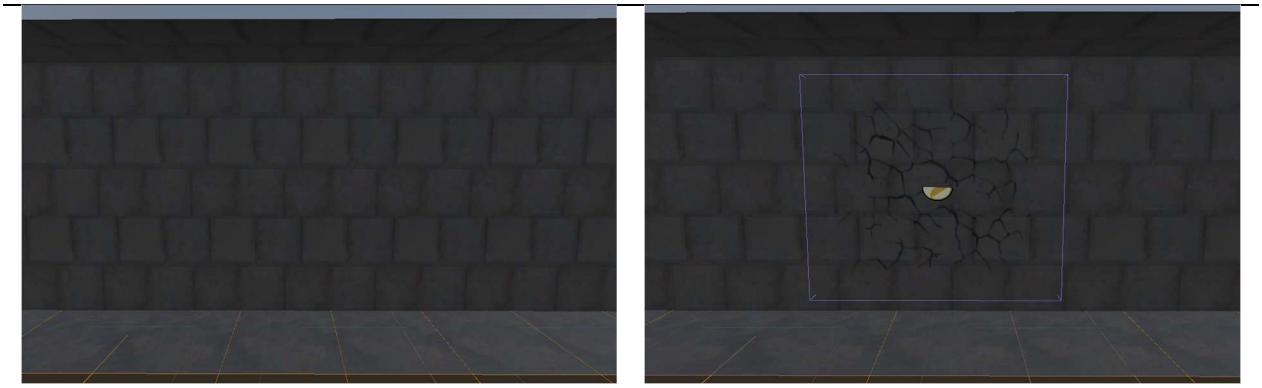


Рисунок 2.3. Розташування опорної точки архітектурного модуля в середовищі Blender

Для запобігання візуальній монотонності та ефекту «шахової дошки», які є неминучим побічним ефектом частого повторення однакових модулів, застосовуються додаткові методи просторового маскування поверхонь [18, с. 88 - 92]. Ефективність застосування цього підходу у розробленій локації продемонстрована на рис. 2.4. Як видно з порівняння базової стіни (рис. 2.4, а)

та стіни з накладеною декаллю (рис. 2.4, б), використання системи вузлів Decal дозволяє накладати напівпрозорі текстури сколів, ерозії та тріщин безпосередньо поверх швів полігональних сіток. Такий підхід дозволяє повністю приховати регулярні стики модулів і створити ілюзію унікальності кожної ділянки замкового середовища, зберігаючи високу продуктивність рендерингу завдяки багаторазовому використанню тієї самої базової геометрії.



а) б)

Рисунок 2.4. Демонстрація маскування візуальної монотонності за допомогою декалей у середовищі Godot: а) базова модульна стіна; б) стіна із застосованим вузлом Decal

У програмному середовищі Godot ця парадигма проєктування ідеально поєднується з глибинною концепцією екземплярів. Вона дозволяє завантажити геометрію та супутні текстурні карти базового модуля в пам'ять відеокарти лише один раз, а потім відтворювати сотні ідентичних модулів на екрані з мінімальним додатковим навантаженням на центральний процесор комп'ютера.

Для емпіричного підтвердження ефективності цього підходу було проведено профілювання споживання системних ресурсів за допомогою вбудованого інструменту моніторингу рушія. Тестовий сценарій передбачав рендеринг ділянки локації, що складається з 39 архітектурних модулів із накладеними картами текстур та картою нормалей. Результати порівняння фактичного використання відеопам'яті (модульний підхід з екземплярами) та

споживання при традиційному (монолітному) підході, де кожен з 39 модулів завантажувався б як унікальна полігональна сітка з власним незалежним набором текстур, наведено в табл. 2.1.

Таблиця 2.1.

Порівняльний аналіз споживання ресурсів при рендерингу архітектурних модулів

Показник	**Традиційний (монолітний) підхід**	**Модульний підхід**	**Результат порівняння**
Споживання відеопам'яті	1930,1 МБ	49,49 МБ	Економія ~ 97,4%
Споживання пам'яті текстур	1450,0 МБ	37,18 МБ	Економія ~ 97,4%
Виклики малювання	117	39	Скорочення у 3 рази

2.2 Математичні та фізичні основи сучасних матеріалів

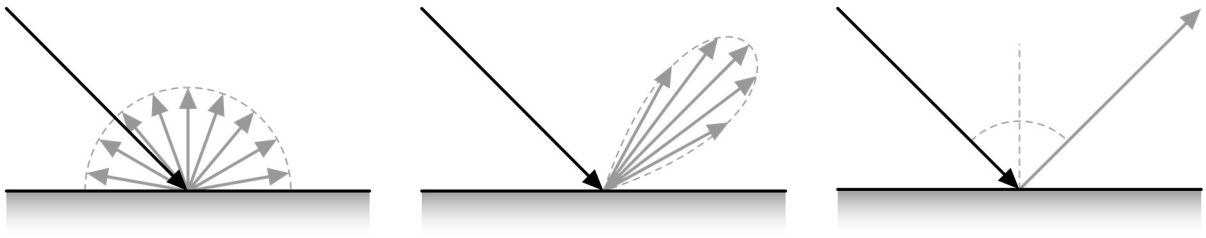
Формування достовірного та переконливого візуального ряду вимагає використання математичних моделей рендерингу, що імітують справжні оптичні явища. Сучасним індустріальним стандартом є алгоритми, які ґрунтуються на фундаментальному законі збереження енергії та мікрофацетній моделі взаємодії світлових променів з матеріалами [5, с. 531 - 534; 19, с. 7 - 10]. Математично це описується двопроменевою функцією розподілу відбивної здатності (формула 2.1). Стандартною є аналітична модель Кука-Торренса, яка розділяє відбите світло на дифузну та дзеркальну компоненти:

$$f(l,v) = f_{diffuse} + f_{specular} \quad (2.1)$$

де l - вектор напрямку на джерело світла;

v - вектор напрямку на спостерігача.

Згідно з мікрофацетною теорією, будь-яка, навіть найбільш гладка на вигляд поверхня, на мікроскопічному рівні розглядається як сукупність безлічі дрібних плоских граней (мікрофацет). Характер взаємодії світлового променя з матеріалом повністю визначається просторовою орієнтацією цих мікрограней, як це показано на рис. 2.5 [20].



а) б) в)

Рисунок 2.5. Схема взаємодії світлових променів із поверхнею матеріалу: а) дифузне розсіювання; б) глянцеве відбиття; в) ідеальне дзеркальне відбиття

Дзеркальна компонента, яка безпосередньо відповідає за формування чітких або розмитих відблисків на поверхні кам'яних та металевих елементів розробленої локації, розраховується за формулою 2.2:

$$f_{specular} = \frac{D(h)\times F(l,h)\times G(l,v,h)}{4\times(n\times l)\times(n\times v)} \quad (2.2)$$

де h - вектор середньої нормалі;

n - макронормаль поверхні;

$D(h)$ - функція розподілу мікрофацет (GGX);

$F(l,h)$ - функція Френеля;

$G(l,v,h)$ - функція геометричного затінення.

Закон збереження енергії у контексті комп'ютерної графіки жорстко постулює, що загальна кількість відбитого поверхнею світла за жодних умов не може перевищувати кількість світла, що впало на неї від джерела. Математично це фундаментальне оптичне обмеження для будь-якої моделі відбиття виражається через інтегральну нерівність по півсфері (формула 2.3):

$$\int_{\Omega} f(l,v) \times (n \times l) d\varpi_i \tag{2.3}$$

де Ω - одиночна півсфера над точкою поверхні;

$f(l,v)$ - значення функції над точкою поверхні;

n - макронормаль поверхні;

l - вектор напрямку на джерело світла;

$d\varpi_i$ - диференціал тілесного кута.

Впровадження цього закону в алгоритми шейдерів назавжди виключає використання штучних, намальованих вручну відблисків чи тіней у текстурах, і робить поведінку матеріалу абсолютно передбачуваною та природною за будь-яких, навіть найекстремальніших, умов динамічного освітлення сцени.

Сучасний стандартизований робочий процес створення таких матеріалів для ігрових рушіїв оперує кількома основними текстурними картами, що передають різні фізичні властивості моделі. Для реалізації об'єктів розробленої локації (модулів кам'яної кладки стін, підлоги, а також декоративних металевих елементів) було застосовано повноцінний конвеєр PBR-візуалізації. Дані про ці карти, їхнє фізичне значення та особливості практичного впровадження у проєкті зведено у табл. 2.2.

Таблиця 2.2

Основні текстурні карти PBR та їх характеристики

Тип текстурної карти	**Фізичне значення та функція у конвеєрі рендерингу**	**Практичне застосування у розробленому віртуальному середовищі**
Карта базового кольору (Albedo)	Визначає чистий колір дифузного розсіювання для діелектриків або колір відбиття для провідників. Категорично не повинна містити намальованої інформації про об'єм, тіні чи відблиски.	Використано для передачі автентичних колірних відтінків кам'яних блоків, плит підлоги та первинного кольору залізного ящика.
Карта металевості (Metallic)	Бінарна або градієнтна чорно-біла маска, що вказує графічному процесору, які ділянки моделі є металом, а які - діелектриком.	Застосовано для чіткого розмежування матеріалів: ізоляції кам'яної кладки (значення 0.0) від повністю металевої поверхні корпусу ящика (значення 1.0).
Карта жорсткості (Roughness)	Керує мікрорельєфом поверхні на основі мікрофацетної теорії, безпосередньо впливаючи на розмір, інтенсивність та чіткість світлового відблиску.	Налаштовано для симуляції матової шорсткості старої кам'яної кладки та формування розмитих відблисків на потертому металі ящика.
Карта нормалей (Normal)	Векторна текстура, яка математично змінює напрямок вектора нормалі поверхні для кожного пікселя. Це створює високодеталізовану ілюзію складної геометричної форми абсолютно без збільшення кількості реальних полігонів у моделі.	Забезпечує реалістичну імітацію глибоких тріщин та відколів на архітектурних модулях стін, а також візуальних вм’ятен на металевій поверхні ящика.

Результат практичного застосування текстурних карт PBR-конвеєра та їхній безпосередній вплив на взаємодію поверхні з джерелами динамічного освітлення у робочому просторі ігрового рушія Godot Engine 4 продемонстровано на рис. 2.6. Для наочності наведено порівняння відображення створеного металевого технічного ящика при різних значеннях карти жорсткості.

а) б)

Рисунок 2.6. Візуальне порівняння реакції PBR-матеріалу на динамічне освітлення: а) низька жорсткість, висока металевість; б) висока жорсткість, низька металевість

Застосування цього індустріального стандарту в інтегрованих шейдерах Godot дозволяє отримати фотореалістичний результат. Як видно з результатів тестування, рушій фізично коректно розраховує поведінку променя: при низькій жорсткості метал діє як дзеркало, а при високій − мікрорельєф розсіює світло, роблячи поверхню матовою. Це гарантує, що кам'яний блок завжди виглядатиме як природний камінь, а залізний ящик − як щільний метал, незалежно від того, освітлюються вони яскравим спрямованим світлом чи тьмяним фоновим туманом.

2.3 Аналіз методів просторової оптимізації та рівня деталізації

Збільшення кількості геометричних об'єктів, джерел світла та складних матеріалів у сцені експоненційно підвищує навантаження як на центральний, так і на графічний прискорювачі системи. Тому невід'ємною частиною

розробки інтерактивного продукту є глибока алгоритмічна оптимізація процесу рендерингу.

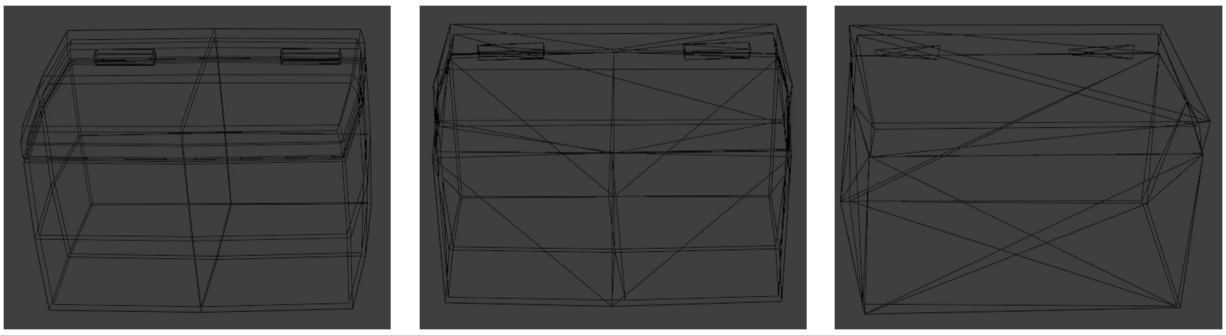
Першим базовим етапом збереження ресурсів є алгоритм відсікання невидимої геометрії, який за замовчуванням автоматично виключає з конвеєра рендерингу всі просторові об'єкти, що геометрично не потрапляють у піраміду видимості віртуальної камери гравця на поточному кадрі [21 с. 512 - 510]. Проте у складних закритих просторах тисячі об'єктів можуть математично знаходитися в межах кута огляду піраміди видимості, але при цьому повністю перекриватися найближчими не прозорими стінами. Відеокарта витрачатиме ресурси на їх малювання, хоча гравець їх ніколи не побачить.

Для вирішення цієї серйозної проблеми застосовується складна технологія відсікання перекритих об'єктів [22, с. 412 - 415]. У четвертій версії Godot імплементовано високоефективний динамічний алгоритм, який генерує сильно спрощений буфер глибини сцени на центральному процесорі та превентивно відкидає виклики малювання для тих полігонів, які гарантовано знаходяться позаду перешкод. Це дозволяє зменшити навантаження в архітектурно насичених локаціях, звільняючи ресурс для розрахунку фізики чи освітлення.

Ще одним ефективним інструментом збереження продуктивності виступає динамічна зміна рівнів деталізації моделі [23, с. 55 - 60]. Суть алгоритму полягає у врахуванні особливостей людського ока: чим далі знаходиться об'єкт, тим менше деталей здатен розрізнити гравець. Тому промальовувати складну сітку на великих дистанціях недоцільно. Варто зазначити, що у межах даної роботи було свідомо обрано не надто високополігональні моделі для створення ретро/low-poly візуального стилю, що само по собі значно знижує базове навантаження на систему, однак використання алгоритмів спрощення геометрії усе одно дозволяє додатково оптимізувати графічний конвеєр. Сучасний конвеєр імпорту Godot здатний автоматично аналізувати сітку і генерувати кілька спрощених версій тривимірної моделі під час її завантаження. Під час гри рушій динамічно і

непомітно для ока підміняє високополігональну модель на спрощені версії залежно від математичної відстані до віртуальної камери, таким чином радикально зменшуючи загальну кількість вершин у кадрі без візуальних втрат якості зображення.

Приклад автоматичної генерації трирівневої системи LOD для розробленого архітектурного елемента (технічного ящика) та метрики послідовного спрощення топології полігональної сітки залежно від відстані до камери спостерігача продемонстровано на рис. 2.7.



а) б) в)

Рисунок 2.7. Рівні деталізації полігональної сітки тривимірної моделі ящика: а) LOD 0 (дистанція < 5 м, 100% полігонів); б) LOD 1 (дистанція 5 - 12 м, ~50% полігонів); в) LOD 2 (дистанція > 12 м, ~15% полігонів)

Для емпіричного підтвердження цього критерію та оцінювання ефективності впроваджених методів деклюдингу було проведено комплексне профілювання розробленої локації за допомогою вбудованого інструменту моніторингу продуктивності. Під час експерименту порівнювалися показники геометричного навантаження графічного процесора для двох сценаріїв: за умов повної відсутності просторової оптимізації та при активованому комплексі алгоритмів просторового відсікання та автоматичних рівнів деталізації. Отримані інженерні дані профілювання зведено в табл. 2.3.

Таблиця 2.3

Вплив методів просторової оптимізації на геометричне навантаження графічного конвеєра

Режим рендерингу локації	**Кількість оброблюваних вершин у кадрі**	**Кількість геометричних примітивів**	**Середня частота кадрів**
Базовий режим (без оптимізації)	190 800	95 400	74 FPS
Оптимізований режим (Culling + LOD)	13 520	6 760	144 FPS (стабільно)
Коефіцієнт ефективності	Скорочення у 14 разів	Скорочення у 14 разів	Приріст продуктивності на ~94,5%

Аналіз результатів експериментального профілювання доводить, що без використання просторових фільтрів графічний процесор витрачає потужності на обчислення геометричних параметрів 95 400 полігонів, значна частина яких прихована від погляду користувача іншими стінами або через велику відстань не потребує базової деталізації. Активація оптимізаційного конвеєра дозволяє знизити щільність вершин на екрані до 13 520 (6 760 примітивів), що розвантажує блоки растеризації GPU і забезпечує стабільну частоту кадрів на рівні 144 FPS без втрати загальної візуальної якості сцени.

Додатковим, але не менш важливим фактором оптимізації є архітектурне групування викликів малювання. При використанні десятків однакових модульних елементів система оптимізації Godot здатна об'єднувати їх в єдині масиви даних для одночасної передачі на відеокарту за один цикл, що суттєво розвантажує вузьке місце системи.

У розробленому проєкту цей інженерний механізм було практично реалізовано шляхом інтеграції модульних конструкцій у систему вузла Gridmap, який на низькому рівні автоматично оперує технологією апаратного

інстансингу. Замість того, щоб надсилати окремий запит на малювання для кожного індивідуального кам'яного блока чи плити підлоги, графічний конвеєр групує всі геометричні копії, які поділяють спільний PBR-матеріал, та передає їх GPU за один крок рендерингу. Для аналізу ефективності батчингу в межах створеної локації було проведено заміри щільності викликів малювання для основних масивів об'єктів. Результати аналізу зведено в табл. 2.4.

Таблиця 2.4

Аналіз викликів малювання при оптимізації модульних елементів сцени

Категорія об'єктів локації	**Фактична кількість об'єктів у просторі сцени**	**Виклики малювання без групування**	**Виклики малювання після батчингу**
Архітектурні модулі стін (камінь)	120	120	1
Модулі плитки підлоги та стелі	85	85	1
Декоративні технічні ящики (метал)	14	14	1
Усього	219	219	3

Чисельні результати проведеного аналізу наочно демонструють, що без застосування групування центральний процесор змушений був виконати 219 послідовних викликів малювання, що призвело б до простою конвеєра через постійне перемикання контексту графічного драйвера. Об'єднання ресурсів у межах архітектури Godot Engine дозволило скоротити загальну кількість Draw Calls для базової геометрії сцени всього до 3 викликів (по одному на кожен унікальний тип об'єкта та його PBR-матеріал), що повністю нівелює ризик виникнення апаратної надлишковості обчислень.

2.4 Патерни проєктування інтерактивних систем

Для створення стійкої, гнучкої та масштабованої програмної архітектури взаємодії між суб'єктом управління та інтерактивними об'єктами локації необхідно повністю уникати жорсткої зв'язаності коду. Найбільш ефективним та сучасним інженерним підходом у цьому контексті є комплексне застосування архітектурного патерну «Компонент» спільно з подієво-орієнтованою моделлю комунікації [13, с. 211 - 215]. Організаційну структуру розробленої децентралізованої архітектури, ієрархію успадкування класів та логіку міжкомпонентної взаємодії на основі сигнальних зв'язків представлено у вигляді UML-діаграми на рис. 2.8.

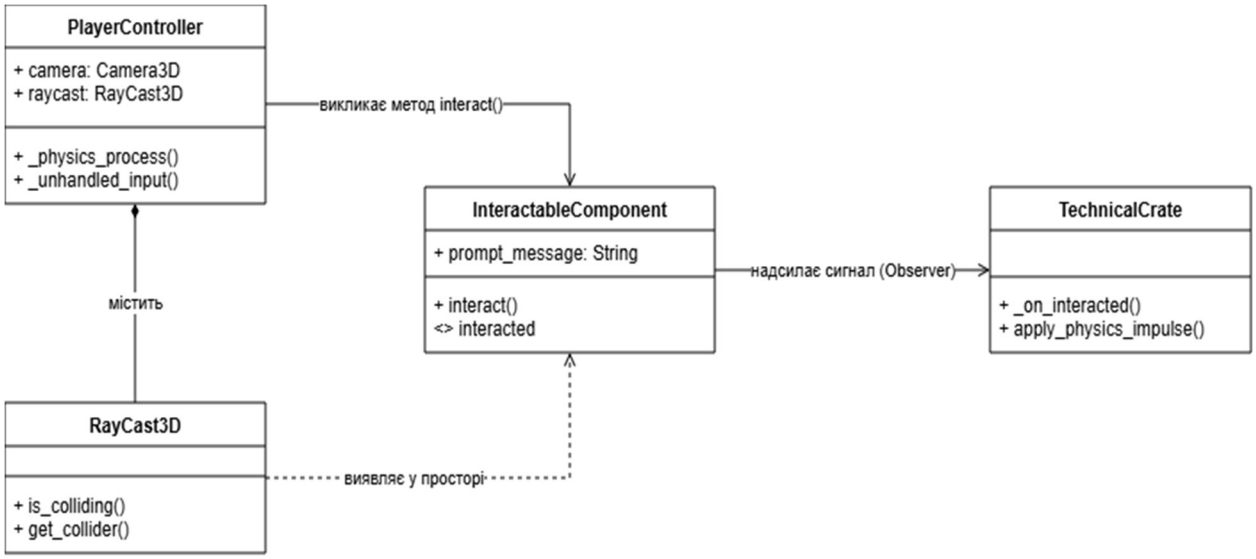


Рисунок 2.8. UML-діаграма архітектурних патернів «Компонент» та «Спостерігач» у програмній системі взаємодії об'єктів локації

Фундаментальна суть компонентного підходу у проєктуванні логіки полягає в тому, що абстрактна властивість бути інтерактивним об'єктом виноситься розробником в окремий, ізольований програмний вузол. Замість того, щоб програмувати кожні унікальні двері чи інформаційний термінал власним складним скриптом для розпізнавання наближення гравця, до них просто додається універсальний дочірній компонент взаємодії. Це дозволяє уникнути дублювання коду та помилок.

Практичну реалізацію архітектурного патерну «Компонент» у розробленому проєкту виконано мовою GDScript. Для цього було створено ізольований клас InteractableComponent, який успадковує властивості просторового тригера Area3D. Програмний код цього базового компонента наведено у лістингу 2.1.

Лістинг 2.1

Програмна реалізація базового компонента взаємодії мовою GDScript

```
class_name InteractableComponent
extends Area3D

# Сигнал (подія), що сповіщає інші об'єкти про факт взаємодії
signal interacted(body: Node3D)

# Експортована змінна для налаштування тексту підказки в інспекторі
@export var prompt_message: String = "Натисніть [E] для взаємодії"

# Публічний метод, який викликається променем (RayCast3D) гравця
func interact(body: Node3D) -> void:
    # Dynamic Emit події для всіх підписаних об'єктів (Observer)
    interacted.emit(body)
```

Алгоритм виявлення намірів гравця у тривимірному просторі будується на основі математичного випромінювання променя-вектора безпосередньо з центру віртуальної камери. На кожному розрахунковому кадрі фізичний рушій здійснює швидкий запит у простір. Якщо промінь перетинає полігональний об'єкт, система не намагається одразу викликати метод відкриття. Натомість вона перевіряє наявність на цьому об'єкті компонента взаємодії, активно використовуючи парадигму динамічної типізації мови GDScript.

У разі успішної ідентифікації компонента та паралельного отримання сигналу від пристрою введення користувача, скрипт променя ініціює виклик абстрактного методу взаємодії на знайденому об'єкті. Своєю чергою, цей інтерактивний об'єкт використовує класичний патерн «Спостерігач», який у середовищі Godot реалізований на базовому рівні через вбудовані сигнали, для швидкого оповіщення інших ізольованих систем про зміну свого внутрішнього стану [24, с. 293 - 303]. Наприклад, при натисканні на перемикач, об'єкт вимикача лише генерує сигнал. Об'єкт дверей, який

попередньо був налаштований на прослуховування цього конкретного сигналу, отримує його і самостійно ініціює анімацію відкриття [25, с. 142 - 145]. Такий глибоко децентралізований підхід забезпечує модульність не лише графічної частини проєкту, а й програмної логіки, дозволяючи ігровому дизайнеру конструювати нові складні ланцюги просторових взаємодій прямо в редакторі без жодної зміни вихідного коду скриптів.

Практичну імплементацію патерна «Спостерігач» на прикладі інтерактивних дверей, які реагують на подію від компонента, наведено у лістингу 2.2.

Лістинг 2.2

Реалізація патерна «Спостерігач» для обробки сигналів у класі дверей

```
extends Node3D

# Посилання на підключений компонент та вузол анімації
@export var interactable_component: InteractableComponent
@export var animation_player: AnimationPlayer

func _ready() -> void:
    # Динамічна підписка на сигнал компонента (реалізація Observer)
    if interactable_component:
        interactable_component.interacted.connect(_on_door_interacted)

# Метод-обробник, який спрацьовує тільки при отриманні сигналу
func _on_door_interacted(_body: Node3D) -> void:
    if animation_player:
        animation_player.play("open_door")
```

У цьому випадку клас дверей виступає незалежним слухачем. Використання вбудованого методу connect() забезпечує динамічне зв'язування об'єктів під час ініціалізації сцени _ready(). Таким чином, тривимірна модель об'єкта та її анімації повністю відділені від системної логіки виявлення гравця чи обробки пристроїв введення, що гарантує відсутність критичних помилок при масштабуванні проєкту.

Висновки до розділу 2

Другий розділ присвячено обґрунтуванню теоретичних принципів та експериментальних методів проєктування, візуалізації та оптимізації тривимірної ігрової локації. За результатами дослідження сформульовано такі висновки:

1. Дослідження принципів модульного проєктування підтвердило доцільність формування локації з уніфікованих блоків за допомогою інструменту Gridmap. Для маскування ефекту візуальної монотонності успішно застосовано вузли Decal. Такий архітектурний підхід дозволив скоротити споживання відеопам'яті на ~97,4% порівняно з монолітним моделюванням.

2. Використання алгоритмів фізично коректного рендерингу (PBR) на основі мікрофацетної моделі забезпечує математично точну імітацію взаємодії поверхонь зі світлом.Застосування базового набору текстурних карт (базового кольору, металевості, жорсткості та нормалей) дає змогу відтворювати складний рельєф об'єктів без збільшення полігонального навантаження.

3. Впровадження комплексу просторової оптимізації, що включає динамічну зміну рівнів деталізації та алгоритми відсікання перекритих об'єктів, зменшило кількість оброблюваних вершин у 14 разів та стабілізувало частоту кадрів. Завдяки апаратному групуванню кількість викликів малювання для базової геометрії скоротилася з 219 до 3.

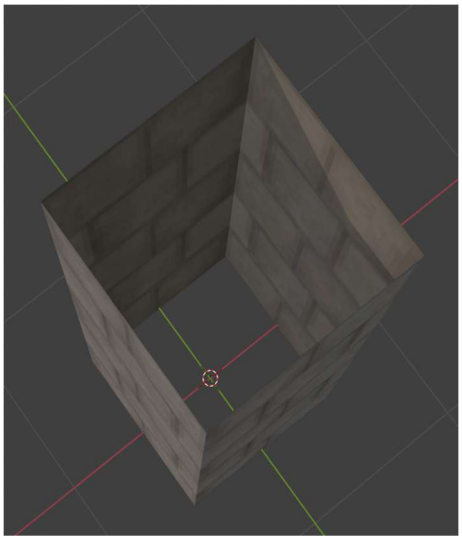
4. Програмну систему взаємодії побудовано на базі архітектурних патернів «Компонент» та «Спостерігач». Використання ізольованих вузлів взаємодії та математичного променя-вектора дозволило уникнути жорсткої зв'язності коду, повністю відокремивши логіку ідентифікації дій від графічних моделей та їх анімацій.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ІГРОВОЇ ЛОКАЦІЇ ТА ІНТЕРАКТИВНИХ ЕЛЕМЕНТІВ

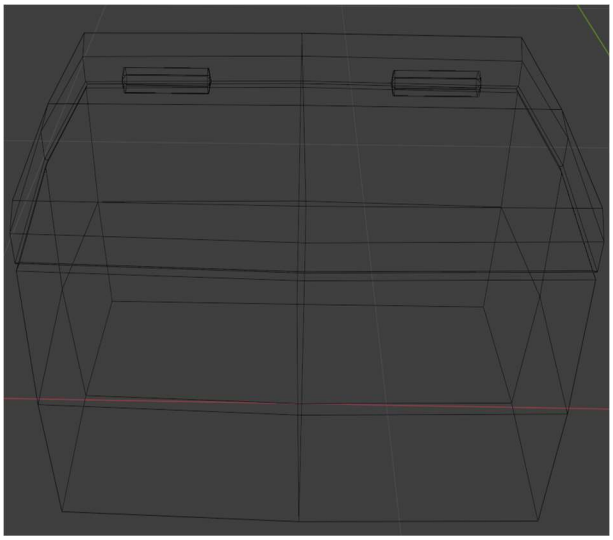
3.1 Створення геометричних об'єктів та текстурування у середовищі Blender

Практичний етап реалізації проєкту логічно розпочався зі створення набору модульних блоків у відкритому програмному комплексі тривимірного моделювання Blender. Першочергове інженерне налаштування робочого простору програми включало встановлення метричної системи одиниць та жорстку абсолютну прив'язку інструментів до просторової сітки. Цей критичний крок дозволив математично точно спроєктувати базові архітектурні елементи: прямі секції стін, кутові з'єднання, стіни з отворами для інтерактивних дверей, масивні плити перекриття та підлоги, які згодом ідеально стикуються між собою у рушії. [26]

Процес геометричного моделювання у межах даного проєкту базувався на підході прямого створення низькополігональних моделей [27, с. 88 - 91]. Такий вибір обґрунтований необхідністю жорсткої економії системних ресурсів графічного конвеєра та специфікою модульної архітектури рівня. Замість ресурсомісткого циклу створення високополігональних моделей із подальшою ручною ретопологією, візуальна деталізація об'єктів переносилася безпосередньо на етап текстурування за допомогою карт нормалей. Геометрична оптимізація базових форм здійснювалася на етапі полігонального моделювання шляхом ручного видалення прихованих граней та злиття копланарних вершин. Візуальне підтвердження оптимізованої топології розроблених моделей наведено на рис. 3.1.



а)



б)

Рисунок 3.1. Приклади ручної геометричної оптимізації низькополігональних моделей: а) колона з видаленими торцевими гранями; б) ящик із мінімізованою внутрішньою порожниною

Наступним важливим кроком було здійснення розгортки текстурних координат для математичного перетворення складної тривимірної поверхні у плоский двовимірний простір. З метою глобальної оптимізації та додаткового зменшення кількості викликів малювання у рушії, для архітектурних модулів було застосовано метод текстурних атласів [28, с. 105 - 108]. Це інженерне рішення дозволило об'єднати текстури стін, підлоги та стелі в єдиний графічний файл матеріалу. Під час розгортки суворо дотримувалася концепція єдиної щільності текселів, що гарантує однаковий фізичний розмір пікселя на всіх модулях, унеможливлюючи ситуацію, коли поруч знаходяться об'єкти з деталізованими та розмитими текстурами. Процес позиціонування UV-островів на текстурному атласі та візуальний результат текстурування архітектурного модуля наведено на рис. 3.2.

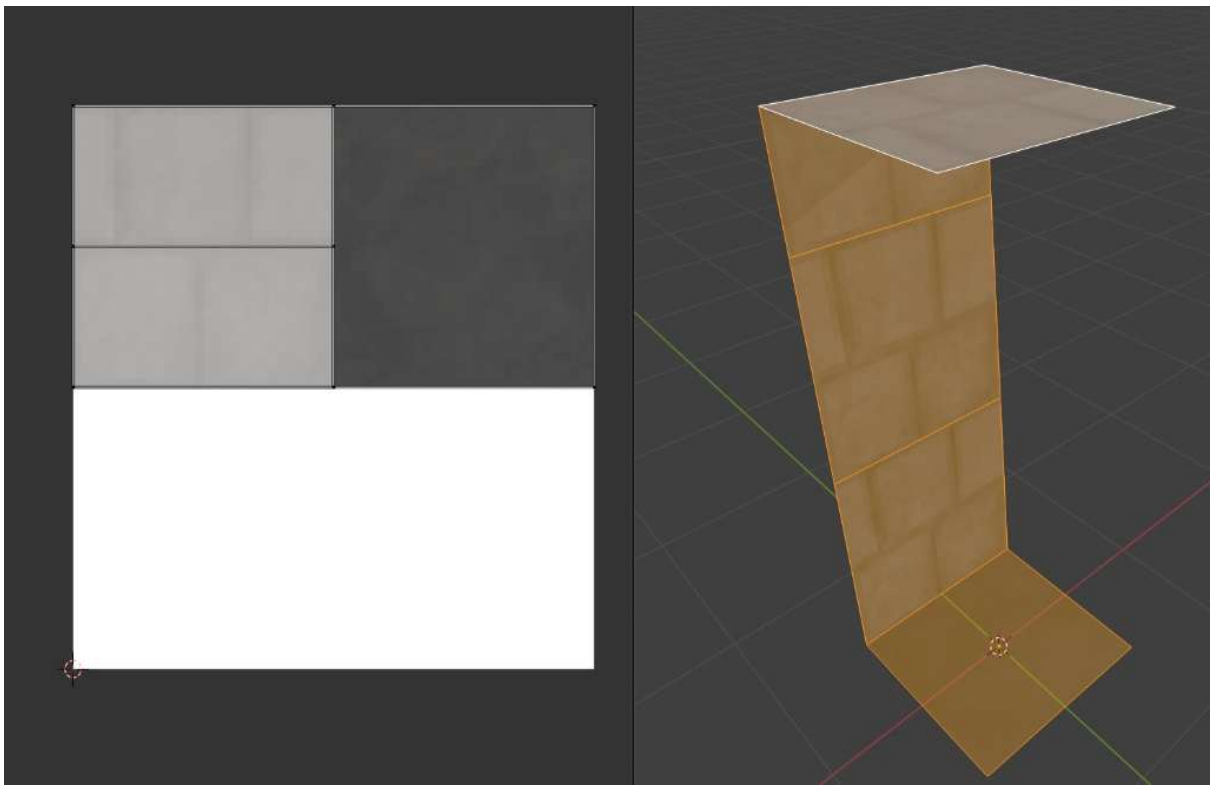


Рисунок 3.2. Процес створення текстурних координат

Далі було виконано обчислювальний процес запікання інформації. Під час цієї процедури геометрична глибина з високополігональної моделі була прорахована алгоритмом і перенесена на двовимірну карту нормалей низькополігональної моделі. Створення фізично коректних матеріалів включало паралельну генерацію карт базового кольору, шорсткості та металевості за допомогою технік процедурного накладання шумів та малювання шарами безпосередньо по тривимірній моделі.

Готові оптимізовані активи були семантично згруповані у колекції та експортовані з використанням сучасного формату обміну даними glTF 2.0. Цей формат є офіційно рекомендованим стандартом для Godot, оскільки він ефективно та надійно пакує полігональну геометрію, ієрархію вузлів та закодовані текстурні дані в єдиний двійковий файл, мінімізуючи будь-які втрати інформації при перенесенні між програмами.

3.2 Експорт ресурсів та налаштування параметрів сцени в Godot

Після операції імпорту файлів формату glTF до файлової структури проєкту, внутрішні алгоритми рушія Godot автоматично проаналізували топологію моделей, згенерували оптимізовані рівні деталізації для кожної сітки та розпізнали параметри фізично коректних матеріалів, безпомилково конвертувавши їх у свій внутрішній стандартний клас відображення тривимірних поверхонь.

Архітектура глобальної сцени локації була побудована суворо за принципом ієрархічної вкладеності. Для забезпечення зручності маніпулювання під час дизайну рівня, кожен унікальний імпортований архітектурний модуль було збережено в системі як окрему повноцінну сцену-компонент. Лише після цього з екземплярів цих компонентів було сконструйовано безперервну загальну геометрію ігрового рівня з дотриманням координатної сітки прив'язки.

Для створення відповідної психологічної атмосфери похмурого підземелля була ретельно налаштована складна система освітлення середовища. Її обчислювальну основу склала технологія динамічного глобального освітлення на основі полів підписаних відстаней. Ця передова технологія аналізує просторову конфігурацію масивних об'єктів у реальному часі та прораховує багаторазове відбиття світлових променів від стін та підлоги [29]. Вона генерує реалістичні м'які тіні та кольорові рефлекси без необхідності витрачати години на статичне попереднє запікання карт освітлення, що робить процес дизайну швидким та ітеративним. Наочне порівняння та результати візуалізації сцени з використанням налаштованої системи глобального освітлення наведено на рис. 3.3.

а) б)

Рисунок 3.3. Візуалізація освітлення в середовищі Godot: а) базове пряме освітлення; б) сцена з активованою системою глобального освітлення

Додаткову оптичну глибину замкнутому простору забезпечило програмне впровадження об'ємного туману. Ця технологія математично візуалізує розсіювання світла у заповненому мікрочастинками повітрі та ефектно підкреслює наявність локальних штучних джерел світла, зокрема променя направленого ліхтарика гравця. Практичний результат налаштування щільності та розсіювання об'ємного туману в середовищі локації продемонстровано на рис. 3.4.

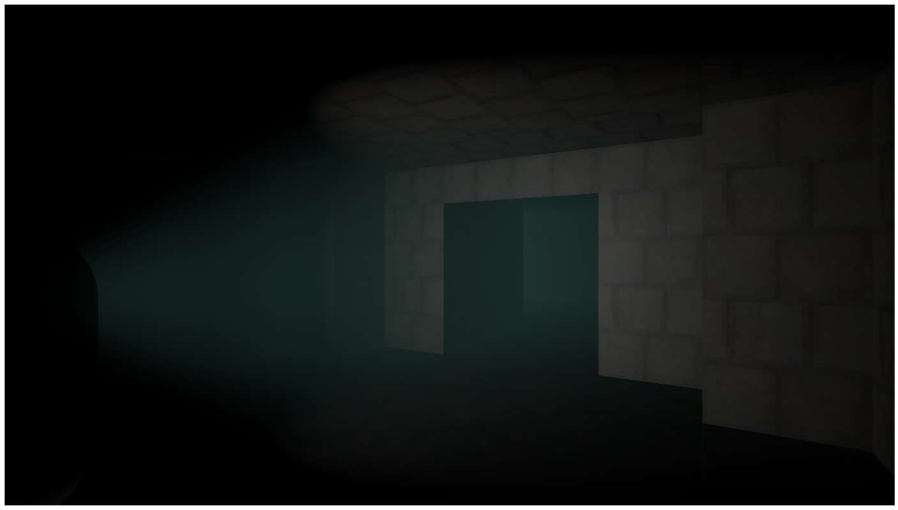


Рисунок 3.4. Візуалізація ефекту об'ємного туману, що підкреслює направлений промінь джерела світла у підземеллі

Для забезпечення коректної фізичної взаємодії гравця з середовищем, кожному імпортованому архітектурному модулю ще на етапі налаштування шаблону було додано статичні просторові оболонки колізій. Ці математичні сітки точно повторюють зовнішні контури стін та масивних перешкод, жорстко запобігаючи випадінню рухомих динамічних об'єктів за визначені межі локації під дією віртуальної гравітації.

3.3 Програмна реалізація контролера гравця та інтерактивних механік

В основі керування віртуальним простором лежить самостійно розроблений контролер персонажа. Він базується на спеціальному вбудованому класі рушія CharacterBody3D, призначеному для рухомих фізичних об'єктів, керованих користувачем. Програмний скрипт на мові GDScript постійно зчитує вхідні вектори з периферійних пристроїв та перетворює їх на рух у тривимірному просторі. При цьому скрипт застосовує до вектора швидкості базові фізичні закони: константу гравітації, прискорення та силу тертя поверхні для плавної зупинки. Рух комп'ютерної миші апаратно перехоплюється системою та трансформується у математичні кути Ейлера для обертання віртикальної голови персонажа, забезпечуючи гравцеві безперешкодний просторовий огляд. Фрагмент програмного коду, що реалізує апаратне перехоплення введення миші, наведено в лістингу 3.1. Повний вихідний код реалізації контролера персонажа та базової обробки введення наведено у додатку Б.

Лістинг 3.1.

Програмна реалізація перехоплення руху комп'ютерної миші

```
func _input(event):
if event is InputEventMouseMotion and Input.get_mouse_mode() == Input.MOUSE_MODE_CAPTURED:
# Обертання тіла персонажа навколо осі Y (вліво-вправо)
  rotate_y(-event.relative.x * 0.003)
  # Обертання вузла камери навколо осі X (вгору-вниз) з обмеженням кута
  head.rotate_x(-event.relative.y * 0.003)
  head.rotation.x = clamp(head.rotation.x, deg_to_rad(-80), deg_to_rad(80))
```

Архітектура системи взаємодії з навколишнім середовищем побудована на основі математичного променя-вектора, який жорстко прикріплений до локальної осі координат віртуальної камери і завжди спрямований у центр екрана. Задана в інспекторі довжина цього променя чітко визначає максимальну фізичну дистанцію, з якої гравець може впливати на об'єкти. Для оптимізації обчислювального навантаження на процесор та уникнення хибного перетину променя з тілом самого персонажа гравця, цей алгоритм функціонує на виділеному фізичному шарі колізій, ігноруючи статичні стіни та геометрію підлоги. Деталізовану схему алгоритму функціонування системи RayCast-взаємодії контролера гравця з інтерактивними об'єктами сцени наведено на рис. 3.5.

Усі об'єкти на локації, з якими логічно передбачена взаємодія, містять вбудований дочірній вузол спеціального класу, що універсально ідентифікує їх для системи як інтерактивні елементи. У момент, коли гравець наводить центральний приціл на такий об'єкт, векторний промінь перетинає цей вузол і викликає сигнал оновлення графічного інтерфейсу користувача. Ця подія автоматично виводить на екран текстову підказку з описом дії, адаптуючи її під конкретний об'єкт.

Практична реалізація логіки взаємодії діє за наступним алгоритмом: при натисканні відповідної клавіші взаємодії, скрипт променя перевіряє наявність публічного методу активації у об'єкта перетину. Слід зазначити, що для простої локальної взаємодії з об'єктами патерн «Спостерігач» не

застосовувався задля уникнення надмірної програмної складності. Натомість було використано алгоритм прямого виклику методів. У випадку масивних дверей, виклик цього методу ініціює плавну анімацію зміни просторових координат за допомогою математичного інтерполятора (лістинг 3.2).

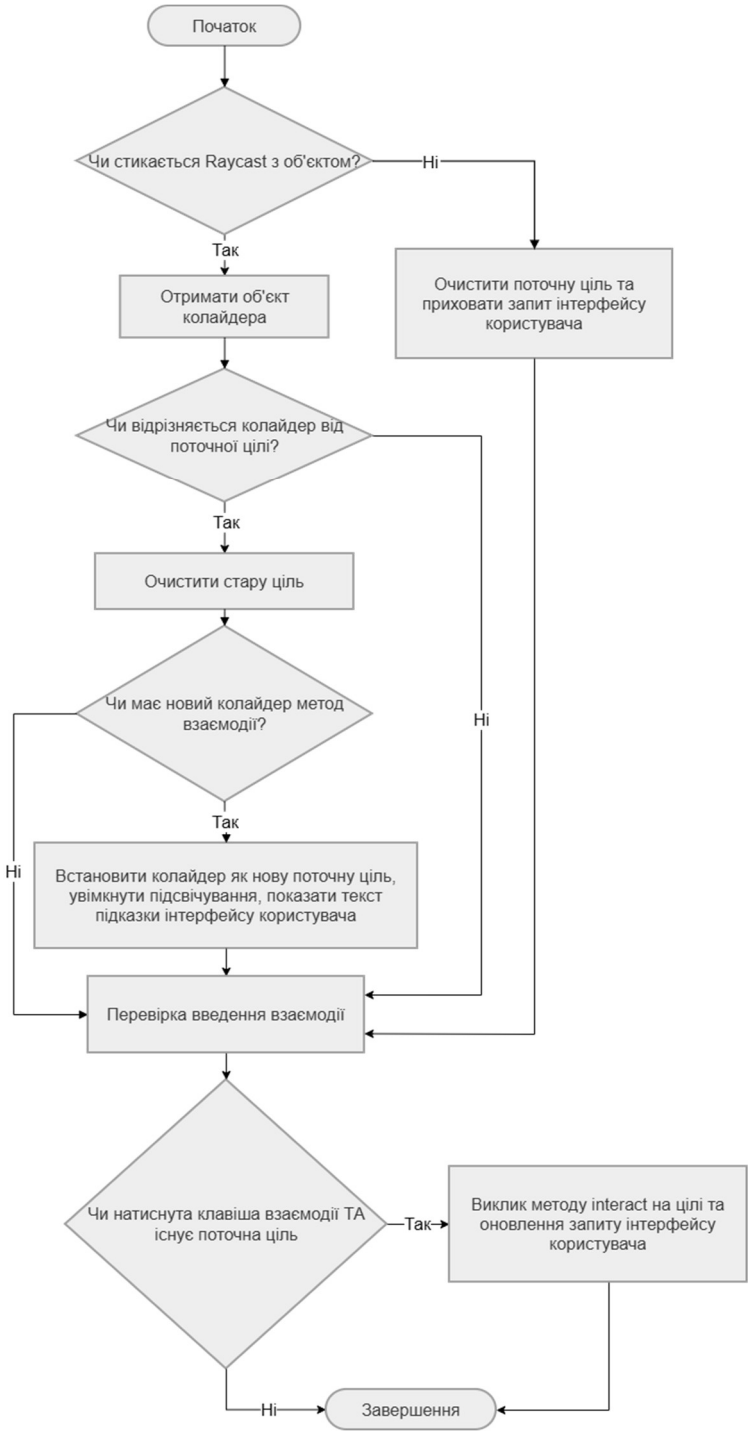


Рисунок 3.5. Блок-схема алгоритму обробки RayCast-взаємодії та оновлення стану інтерфейсу користувача

Лістинг 3.2

Фрагмент програмної реалізації плавного відкриття дверей

```
# Метод взаємодії, що викликається контролером гравця
func interact(_player: Node3D) -> void:
	is_open = !is_open
	var target_rotation = 90.0 if is_open else 0.0

	# Ініціалізація інтерполятора для зміни кута повороту Y
	var tween = create_tween()
	tween.tween_property(
		self, "rotation:y",
		deg_to_rad(target_rotation), 1.0
	).set_trans(Tween.TRANS_QUAD).set_ease(Tween.EASE_OUT)
```

У випадку невеликих декоративних ящиків, які у дереві сцен створені на базі класу симуляції твердих тіл, промінь застосовує до них векторний фізичний імпульс сили. Для забезпечення коректного розрахунку динаміки та оптимізації навантаження на фізичний рушій, складну полігональну колізію таких рухомих об'єктів було замінено на базові геометричні примітиви. Це змушує об'єкт динамічно відлітати від гравця з урахуванням маси ящика та точного кута просторового зіткнення променя з його поверхнею, створюючи переконливий ефект фізичної присутності (лістинг 3.3). Повні програмні скрипти розроблених інтерактивних механік винесено у додаток В.

Лістинг 3.3

Фрагмент програмної реалізації механіки штовхання ящика

```
# Метод прикладання сили до об'єкта класу RigidBody3D
func interact(player: Node3D) -> void:
	# Розрахунок напрямку від персонажа до фізичного тіла
	var p_pos = player.global_transform.origin
	var push_dir = (global_transform.origin - p_pos).normalized()

	push_dir.y = 0.0 # Фіксація імпульсу у горизонтальній площині

	# Застосування імпульсу з урахуванням маси об'єкта
	apply_central_impulse(push_dir * push_force)
```

3.4 Тестування продуктивності та налаштування глобального освітлення

Завершальним, але критично важливим етапом розробки стала перевірка стабільності симуляції локації за допомогою вбудованого в редактор

профайлера продуктивності. Глибинний аналіз метрик показав, що велика кількість динамічних джерел світла у поєднанні з тисячами полігональних моделей створює надмірне обчислювальне навантаження на графічний процесор під час розрахунку тіней [2, с. 550 - 555; 8, с. 815 - 820].

Для стабілізації частоти кадрів та забезпечення плавності ігрового процесу було застосовано комплексний інженерний підхід. По-перше, на рівні всього рівня було активовано алгоритм відсікання перекритих об'єктів. Для цього всередині геометрії модульних стін було згенеровано спеціальну обчислювальну математичну структуру, яка динамічно блокує рендеринг цілих приміщень та об'єктів у них, якщо вони просторово знаходяться поза зоною прямої видимості гравця, перекриті іншими стінами [30]. Емпіричні заміри ефективності даного алгоритму проводилися відповідно до розробленої методики: фіксація метрик графічного конвеєра виконувалася за допомогою вбудованого інструменту моніторингу під час безперервного 60-секундного проходження контролера гравця за заздалегідь визначеним маршрутом локації. Повні технічні характеристики апаратної конфігурації робочої станції, на якій розгортався тестовий стенд, наведено в табл. 3.1.

Таблиця 3.1

Апаратна конфігурація тестового стенда

Апаратний компонент	**Технічні характеристики системи**
Центральний процесор (CPU)	Intel Core i5-11400H (6 ядер, 12 потоків, тактова частота до 4.5 ГГц)
Оперативна пам'ять (RAM)	16 GB DDR4, функціонування у двоканальному режимі, частота 3200 MHz
Графічний прискорювач (GPU)	NVIDIA GeForce RTX 3050 Laptop (обсяг відеопам'яті 4 GB VRAM)
Операційна система (OS)	Windows 11 Home (64-bit розрядність архітектури)

Зведені чисельні результати випробувань конвеєра рендерингу до та після активації алгоритму відсікання наведено в табл. 3.2.

Таблиця 3.2

Результати вимірювання щільності викликів малювання

Параметр рендерингу локації	**Режим без оптимізації (Базовий)**	**Оптимізований режим (Occlusion Culling)**	**Коефіцієнт ефективності (Результат виміру)**
Виклики малювання у пікових зонах	215	86	Скорочення на 60,0%
Середній час рендерингу кадру	13,5 мс	6,8 мс	Зменшення навантаження на ~49,6%

По-друге, було глобально увімкнено функцію динамічної зміни рівня деталізації геометрії. Завдяки цьому складні механізми, труби та ящики автоматично перетворюються на примітивні низькополігональні фігури при значному віддаленні віртуальної камери, різко зменшуючи кількість оброблюваних вершин на графічному процесорі, що абсолютно не помітно для людського ока з такої відстані.

Підсумкове тестування тестування готового прототипу ігрової локації засвідчило, що за умови дотримання стандартів модульного проєктування, грамотної оптимізації полігональної сітки ще на етапі моделювання та системного застосування методів просторового відсікання рушія, розроблена система демонструє високі та стабільні показники продуктивності. Кількісні показники ефективності функціонування тривимірного простору в реальному часі, зафіксовані внутрішніми інструментами моніторингу рушія під час стрес-тестування, наведено в табл. 3.3.

Таблиця 3.3

Кількісні показники продуктивності та навантаження обчислювальних ресурсів системи

Контрольований параметр телеметрії	**Базовий режим**	**Оптимізований режим**	**Цільове/Нормативне значення**
Частота оновлення кадрів (FPS)	74 FPS	144 FPS	~ 60 FPS
Використання оперативної пам'яті (RAM)	1,22 ГБ	0,85 ГБ	< 4,0 ГБ
Використання відеопам'яті (VRAM)	1,93 ГБ	0,78 ГБ	< 4,0 ГБ
Середнє навантаження процесора (CPU)	35,4%	18,2%	< 80,0%
Середнє навантаження відеокарти (GPU)	92,1%	42,5%	< 90,0%

Висновки до розділу 3

Третій розділ присвячено практичним етапам розробки, геометричному моделюванню, програмуванню логіки та тестуванню продуктивності ігрової локації. За результатами роботи сформульовано такі висновки:

1. Практична розробка тривимірних об'єктів у середовищі Blender базувалася на створенні низькополігональних модульних блоків із жорсткою прив'язкою до просторової сітки (підрозділ 3.1). Застосування текстурних атласів з єдиною щільністю текселів та подальший експорт у форматі glTF дозволили мінімізувати кількість викликів малювання та оптимізувати процес перенесення активів до рушія.

2. Налаштування візуального середовища в Godot (підрозділ 3.2) здійснено з використанням ієрархічної структури сцен. Для реалістичного відтворення світла інтегровано систему динамічного глобального освітлення на основі полів підписаних відстаней у поєднанні з об'ємним туманом, що забезпечило коректний розрахунок тіней без статичного запікання.

3. Програмну реалізацію контролера гравця та інтерактивних механік (підрозділ 3.3) виконано мовою GDScript. Завдяки системі взаємодії на основі математичного променя-вектора на виділеному фізичному шарі реалізовано як прості тригерні події (анімація відкриття дверей), так і повноцінну фізичну симуляцію (штовхання ящиків за допомогою векторних імпульсів сили).

4. Експериментальне профілювання прототипу (підрозділ 3.4) підтвердило високу ефективність обраних методів оптимізації. Активація алгоритмів відсікання перекритих об'єктів та автоматичної зміни рівнів деталізації геометрії дозволила знизити середнє навантаження на графічний процесор з 92,1% до 42,5% і забезпечити стабільну частоту кадрів на рівні 144 FPS.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра виконано проєктування, алгоритмічну оптимізацію та програмну реалізацію тривимірної ігрової локації з системою об'єктної взаємодії. За результатами аналітичного дослідження та практичної розробки сформовано наступні висновки:

1. На основі огляду сучасних тенденцій розвитку інтерактивних віртуальних середовищ визначено особливості інтеграції візуального контенту та механік фізичної взаємодії об'єктів для підвищення реалістичності оточення. Встановлено, що архітектурна парадигма оповідання через оточення ефективно реалізується за умови застосування моделей фізично коректної візуалізації матеріалів, які забезпечують прогнозовану взаємодію поверхонь із джерелами освітлення.

2. Сформовано аналітичне обґрунтування вибору ігрового рушія Godot Engine як платформи для реалізації індивідуального проєкту тривимірної локації. Визначено архітектурні переваги платформи, серед яких ієрархічна структура вузлів, відкритість вихідного коду, підтримка сучасних графічних API та гнучкість інтегрованої мови програмування GDScript.

3. Реалізовано методологію модульного проєктування під час створення геометрії об'єктів у середовищі Blender. Дотримання єдиної метричної сітки, уніфікованих точок прив'язки (Pivot Points) та стандартизованої щільності текстур (Texel Density) забезпечило формування реюзабельного набору архітектурних елементів, які коректно імпортовано в ігровий рушій через формат glTF із збереженням ієрархії об'єктів та параметрів матеріалів.

4. Описано та програмно реалізовано гібридну архітектуру об'єктної взаємодії на локації. Застосування компонента розпізнавання на основі променя-вектора (RayCast3D) дозволило розділити логіку контролера персонажа та інтерактивних елементів (дверей, фізичних ящиків), що підтверджено розробленими лістингами коду у підрозділі 3.3 та матеріалами

Додатка Б. Для уникнення надмірної програмної складності просту локальну взаємодію побудовано на базі прямого виклику методів об'єктів.

5. Реалізовано оптимізацію процесу рендерингу сцени через впровадження алгоритмів відсікання перекритих об'єктів та автоматичного перемикання рівнів деталізації. Ефективність оптимізації експериментально підтверджено інструментами профілювання (табл. 3.2, 3.3), що зафіксували скорочення активних викликів малювання на 60%, зниження часу рендерингу кадру з 13,5 мс до 6,8 мс, а також зростання частоти кадрів з 74 FPS до 144 FPS при стабілізації використання оперативної та відеопам'яті. .

Загалом, розроблений прототип тривимірної ігрової локації відповідає встановленим критеріям продуктивності та функціональності, що підтверджено результатами тестування, чисельними показниками телеметрії та ілюстративними матеріалами пояснювальної записки. Створена просторова структура та програмні скрипти демонструють практичну доцільність застосування інструментів з відкритим вихідним кодом (Godot Engine, Blender) як доступної альтернативи для швидкого прототипування та розробки індивідуальних інтерактивних тривимірних застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Slater M. Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. *Philosophical Transactions of the Royal Society B: Biological Sciences*. 2009. Vol. 364, no. 1535. P. 3549-3557.

2. Gregory J. Game Engine Architecture. 3rd ed. Boca Raton : CRC Press, 2018. 1240 p.

3. Järvenpää S. Capabilities of the Open-Source Godot Engine in Game Development Compared to Unity and Unreal Engine : Bachelor's thesis. Tampere: Tampere University, 2021. 42 p.

4. Totten C. W. An Architectural Approach to Level Design. 2nd ed. Boca Raton: CRC Press, 2019. 592 p.

5. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 3rd ed. Cambridge: Morgan Kaufmann, 2016. 1266 p.

6. Walter B., Marschner S. R., Li H., Torrance K. E. Microfacet Models for Refraction through Rough Surfaces. Proceedings of the 18th Eurographics Symposium on Rendering. Grenoble, 2007. P. 195-206.

7. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 4th ed. Cambridge: MIT Press, 2023. 1312 p.

8. Akenine-Möller T., Haines E., Hoffman N. Real-Time Rendering. 4th ed. Boca Raton: A K Peters/CRC Press, 2018. 1198 p.

9. Milam D., El Nasr M. S. Design patterns to guide player movement in 3D games. *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games*. Los Angeles, 2010. P. 37-42.

10. Ahearn L. 3D Game Environments: Create Professional 3D Game Worlds. 2nd ed. Boca Raton: CRC Press, 2017. 361 p.

11. Statham W. Z. Game environment art with modular architecture. Entertainment Computing. 2022. Vol. 41. Article 100476. URL: https://www.sciencedirect.com/science/article/pii/S1875952121000732?via%3Dihub (дата звернення: 18.05.2026).

12. Thorn A. Game Development with Godot 4: A Complete Introduction. Boca Raton: CRC Press, 2025. 376 p.

13. Nystrom R. Game Programming Patterns. Geneseo: Genever Benning, 2014. 354 p.

14. Nodes and Scenes. Godot Engine 4 documentation. URL: https://docs.godotengine.org/en/stable/getting_started/step_by_step/nodes_and_scenes.html (дата звернення: 18.05.2026).

15. glTF™ 2.0 Specification. The Khronos Group Inc. URL: https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html (дата звернення: 18.05.2026).

16. Salmond M. Video Game Design: Principles and Practices from the Ground Up. London: Bloomsbury Visual Arts, 2017. 256 p.

17. Hamdani A. 3D Environment Design with Blender: Enhance your modeling, texturing, and lighting skills to create realistic 3D scenes. Birmingham: Packt Publishing, 2023. 344 p.

18. Kremers R. Level Design: Concept, Theory, and Practice. Natick: A K Peters, 2009. 408 p.

19. Cook R. L., Torrance K. E. A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics*. 1982. Vol. 1, no. 1. P. 7-24.

20. Двопроменева функція розподілу відбивної здатності. Вікіпедія: вільна енциклопедія. URL: https://uk.wikipedia.org/wiki/Двопроменева_функція_відбивної_здатності (дата звернення: 18.05.2026).

21. Eberly D. H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. 2nd ed. San Francisco: Morgan Kaufmann, 2006. 1040 p.

22. Cohen-Or D., Chrysanthou Y., Silva C. T., Durand F. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*. 2003. Vol. 9, no. 3. P. 412-431.

23. Luebke D., Reddy M., Cohen J. D., Varshney A., Watson B., Huebner R. Level of Detail for 3D Graphics. San Francisco: Morgan Kaufmann, 2002. 432 p.

24. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1994. 395 p.

25. Bradfield C. Godot 4 Game Development Projects. 2nd ed. Birmingham: Packt Publishing, 2023. 264 p.

26. Modeling. Blender 4 Reference Manual. URL: https://docs.blender.org/manual/en/latest/modeling/index.html (дата звернення: 18.05.2026).

27. Blain J. M. The Complete Guide to Blender Graphics: Computer Modeling and Animation. 8th ed. Vol. 1. Boca Raton: CRC Press, 2023. 454 p.

28. Бобришев Є. Методи оптимізації тривимірних моделей для реального часу. *Технічні науки та технології*. 2026. № 1(43). С. 105-113.

29. Using Signed Distance Field Global Illumination (SDFGI). Godot Engine 4 documentation. URL: https://docs.godotengine.org/en/stable/tutorials/3d/global_illumination/using_sdfgi.html (дата звернення: 18.05.2026).

30. Occlusion culling. Godot Engine 4 documentation. URL: https://docs.godotengine.org/en/stable/tutorials/3d/occlusion_culling.html (дата звернення: 18.05.2026).

ДОДАТОК А

Порівняльна характеристика сучасних ігрових рушіїв на базі офіційної документації та незалежних досліджень

Характеристика	Unreal Engine 5	Unity	Godot Engine 4
Основні мови програмування логіки	C++, потужна система візуальних вузлів	C#, сторонні системи візуального програмування	Вбудована скриптова мова, C#
Модель ліцензування та розповсюдження	Власницька ліцензія з відрахуванням відсотків з прибутку	Власницька ліцензія на основі підписки або фіксованої плати	Повністю вільна ліцензія без прихованих платежів
Мінімальні вимоги до апаратного забезпечення робочої станції	Процесор (CPU): 4-ядерний процесор Intel або AMD з тактовою частотою 2.5 ГГц або швидше. Оперативна пам'ять (RAM): мінімально необхідно 8 ГБ. Відеокарта (GPU): обов'язкова відеокарта з 8 ГБ відеопам'яті або більше., сумісна з DirectX 11 або 12.	Процесор (CPU): багатоядерний процесор архітектури X64 із підтримкою набору інструкцій SSE2, або ж процесори на базі архітектури ARM64 / Apple M1 та вище. Оперативна пам'ять (RAM): мінімально необхідно 8 ГБ. Відеокарта (GPU): обов'язкова відеокарта, що підтримує DirectX 10, 11 або 12 (для ОС Windows)	Процесор (CPU): процесори з підтримкою інструкцій SSE2. Оперативна пам'ять (RAM): мінімально необхідно 4 ГБ. Відеокарта (GPU): відеокарта не є обов'язковою, рушій оптимізований для повноцінної роботи навіть на інтегрованих графічних процесорах
Фундаментальний архітектурний підхід	Компонентно-орієнтований підхід до побудови сутностей	Компонентно-орієнтований підхід до побудови сутностей	Ієрархія взаємопов'язаних вузлів у дереві сцен
Середній обсяг виконуваного файлу редактора	Більше 10 Гб	Більше 2 Гб	Менше 150 Мб

ДОДАТОК Б

Програмний код контролера персонажа

```
extends CharacterBody3D

@export var speed: float = 5.0
@export var jump_velocity: float = 4.5
@export var mouse_sensitivity: float = 0.002

@onready var camera: Camera3D = $Camera3D
@onready var raycast: RayCast3D = $Camera3D/RayCast3D

func _ready() -> void:
	# Захоплення курсора миші всередині вікна гри
	Input.mouse_mode = Input.MOUSE_MODE_CAPTURED

func _unhandled_input(event: InputEvent) -> void:
	# Обробка обертання камери за допомогою миші
	if event is InputEventMouseMotion and Input.mouse_mode ==
Input.MOUSE_MODE_CAPTURED:
		rotate_y(-event.relative.x * mouse_sensitivity)
		camera.rotate_x(-event.relative.y * mouse_sensitivity)
		camera.rotation.x = clamp(camera.rotation.x, deg_to_rad(-89),
deg_to_rad(89))

func _physics_process(delta: float) -> void:
	# Додавання сили гравітації, якщо гравець у повітрі
	if not is_on_floor():
		velocity += get_gravity() * delta

	# Обробка стрибка
	if Input.is_action_just_pressed("ui_accept") and is_on_floor():
		velocity.y = jump_velocity

	# Отримання вектора напрямку руху з клавіатури
	var input_dir = Input.get_vector("move_left", "move_right",
"move_forward", "move_back")
	var direction = (transform.basis * Vector3(input_dir.x, 0,
input_dir.y)).normalized()

	if direction:
		velocity.x = direction.x * speed
		velocity.z = direction.z * speed
	else:
		velocity.x = move_toward(velocity.x, 0, speed)
		velocity.z = move_toward(velocity.z, 0, speed)

	move_and_slide()
	_process_interaction()

func _process_interaction() -> void:
	# Логіка визначення інтерактивних об'єктів перед гравцем
	if raycast.is_colliding():
		var collider = raycast.get_collider()
		if collider and collider.has_method("set_highlight"):
			collider.set_highlight(true)

		if Input.is_action_just_pressed("interact") and collider and
collider.has_method("interact"):
			collider.interact(self)
```

ДОДАТОК В

Програмна реалізація інтерактивних об'єктів ігрової локації

```
# ========================================
# Скрипт інтерактивних дверей (Door.gd)
# ========================================
extends StaticBody3D

var is_open: bool = false

func interact(_player: Node3D) -> void:
	is_open = !is_open
	var target_rotation = 90.0 if is_open else 0.0

	var tween = create_tween()
	tween.tween_property(
		self, "rotation:y",
		deg_to_rad(target_rotation), 1.0
	).set_trans(Tween.TRANS_QUAD).set_ease(Tween.EASE_OUT)

func get_prompt_text() -> String:
	return "Натисніть [E], щоб зачинити двері" if is_open else "Натисніть
[E], щоб відчинити двері"


# ========================================
# Скрипт фізичного металевого ящика (MetalBox.gd)
# ========================================
extends RigidBody3D

@export var push_force: float = 8.0
@onready var box_mesh: MeshInstance3D = $metal_box
var base_material: Material

func _ready() -> void:
	if box_mesh and box_mesh.mesh:
		base_material = box_mesh.get_active_material(0)

func interact(player: Node3D) -> void:
	var p_pos = player.global_transform.origin
	var push_dir = (global_transform.origin - p_pos).normalized()
	push_dir.y = 0.0
	apply_central_impulse(push_dir * push_force)

func set_highlight(enabled: bool) -> void:
	if not box_mesh or not base_material:
		return
	if enabled:
		var high_mat = base_material.duplicate()
		high_mat.albedo_color = Color(1.5, 1.5, 1.5)
		box_mesh.set_surface_override_material(0, high_mat)
	else:
		box_mesh.set_surface_override_material(0, null)

func get_prompt_text() -> String:
	return "Натисніть [E], щоб штовхнути ящик"
```