ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА ІМЕНІ О. М. БЕКЕТОВА

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему:

СТВОРЕННЯ ТА ОПТИМІЗАЦІЯ ШЕЙДЕРІВ ДЛЯ ВІЗУАЛЬНИХ ЕФЕКТІВ У 2D-ІГРОВОМУ СЕРЕДОВИЩІ РОЗРОБКИ UNITY

Виконав: здобувач вищої освіти
групи КН 2021-1
спеціальності
122 – Комп’ютерні науки
Микита ЖЕРНОВИЙ
Керівник: Олександр КОСТЕНКО
Рецензент: Наталія БРАТЕРСЬКА

м. Харків – 2025

Харківський національний університет міського господарства імені О. М. Бекетова
(повне найменування закладу вищої освіти)
Навчально-науковий Інститут енергетичної, інформаційної та транспортної інфраструктури
Кафедра комп'ютерних наук та інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 Комп'ютерні науки
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КНтаІТ

Марина НОВОЖИЛОВА
«_26__» _____06_____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Жернового Микити Олександровича
(прізвище, ім'я, по батькові)

1. Тема роботи Створення та оптимізація шейдерів для візуальних ефектів у 2D-ігровому проєкті в середовищі разробки UNITY

керівник роботи Костенко Олександр Борисович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «09» травня 2025 р. № 341-03

2. Термін подання студентом роботи 26.06.2025р.

3. Вихідні дані до роботи Проєкт 2D-гри на платформі UNITY, що потребує реалізації та оптимізації набору візуальних ефектів з використанням кастомних шейдерів у середовищі Universal Render Pipeline (URP)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження існуючих підходів, розробка та програмна реалізація набору оптимізованих шейдерів для візуальних ефектів у UNITY. Аналіз продуктивності розроблених рішень. Розгляд питань охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація – 22 аркушів

6. Консультанти розділів роботи

Розділ	Ім’я та Прізвище, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Олександр КОСТЕНКО, к. ф.-м. н., доцент кафедри КН та ІТ	12.05.2025	24.05.2025
2	Олександр КОСТЕНКО, к. ф.-м. н., доцент кафедри КН та ІТ	24.05.2025	31.05.2025
3	Олександр КОСТЕНКО, к. ф.-м. н., доцент кафедри КН та ІТ	31.05.2025	09.06.2025
4	Вікторія МАЛИШЕВА, к. т. н., доцент кафедри ОП та БЖ	10.06.2025	16.06.2025

7. Дата видачі завдання 12 травня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми дипломної роботи	05.05.2025	Виконано
2	Затвердження тем, наукових керівників, завдань та календарного плану підготовки кваліфікаційної роботи	09.05.2025	Виконано
3	Написання І розділу	24.05.2025	Виконано
4	Написання ІІ розділу	31.05.2025	Виконано
5	Написання ІІІ розділу	09.06.2025	Виконано
6	Написання IV розділу	16.06.2025	Виконано
7	Подання дипломної роботи керівнику	17.06.2025	Виконано
8	Робота по усуненню зауважень керівника, уточнення і доповнення практичного матеріалу, оформлення додатків до роботи	23.06.2025	Виконано
9	Подання доопрацьованого варіанту роботи керівнику	25.06.2025	Виконано
10	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	27.06.2025	Виконано

Студент _____ (підпис) Микита ЖЕРНОВИЙ (ім’я та прізвище)

Керівник роботи _____ (підпис) Олександр КОСТЕНКО (ім’я та прізвище)

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра на тему «Створення та оптимізація шейдерів для візуальних ефектів у 2D-ігровому проєкті в середовищі розробки Unity» містить 4 розділи, 23 рисунки, 4 таблиці та список із 22 використаних джерел.

Предмет дослідження: процес розробки та оптимізації шейдерів для візуальних ефектів у 2D-іграх.

Мета роботи: розробити та оптимізувати набір шейдерів для ключових візуальних ефектів у 2D-проєкті на платформі Unity.

Методи дослідження: аналіз предметної області, математичне моделювання, проєктування та аналіз продуктивності.

Програмне забезпечення: Unity 2022.3 (URP), C#, HLSL/ShaderLab, Unity Profiler.

Результати: реалізовано набір оптимізованих шейдерів для візуальних ефектів (світіння, корекція кольору, зернистість). Створено демонстраційну сцену та керівництво з їх інтеграції.

Рекомендації щодо використання: розроблені шейдери можуть застосовуватися для покращення візуальної якості 2D-ігор або як навчальний матеріал з комп'ютерної графіки.

Галузь застосування: індустрія розробки ігор, освітні проєкти.

Актуальність роботи: полягає у створенні набору готових до використання, оптимізованих шейдерів та демонстрації повного циклу розробки графічних рішень, які є в активі сучасної ігрової індустрії.

Ключові слова: UNITY, ШЕЙДЕРИ, HLSL, URP, ОПТИМІЗАЦІЯ, ПОСТ-ОБРОБКА, КОМП’ЮТЕРНА ГРАФІКА, 2D-ГРА, СВІТІННЯ.

ABSTRACT

The explanatory note to the bachelor's qualification thesis on the topic "Development and optimization of a set of shaders for implementing key visual effects in a demonstration 2D game project on the Unity platform" consists of 4 sections, 15 figures, 3 tables, and 25 references.

Subject of research: The process of developing and optimizing shaders for visual effects in 2D games.

Objective of the work: To develop and optimize a set of shaders for key visual effects in a 2D project on the Unity platform.

Research methods: Subject area analysis, mathematical modeling, software design, and performance analysis.

Software: Unity 2022.3 (URP), C#, HLSL/ShaderLab, Unity Profiler.

Results: An optimized set of shaders for visual effects (bloom, color correction, film grain) was implemented. A demonstration scene and an integration guide were created.

Recommendations for use: The developed shaders can be used to enhance the visual quality of 2D games or as educational material in computer graphics.

Field of application: Game development industry, educational projects.

Significance: The work's significance lies in creating a set of ready-to-use, optimized shaders and demonstrating the full development cycle of graphical solutions relevant to the modern game industry.

Keywords: UNITY, SHADERS, HLSL, URP, OPTIMIZATION, POST-PROCESSING, COMPUTER GRAPHICS, 2D GAME, BLOOM.

ЗМІСТ

ВСТУП

Сучасна індустрія комп'ютерних ігор є однією з найбільш динамічно зростаючих та технологічно насичених галузей цифрової економіки. Ігри давно перестали бути простою розвагою, перетворившись на складні інтерактивні продукти, що поєднують у собі мистецтво, технології та дизайн. Особливе місце в цьому процесі займає візуальна складова, яка значною мірою визначає перше враження гравця, атмосферу ігрового світу та загальну привабливість продукту. У контексті 2D-ігор, незважаючи на їх уявну простоту порівняно з тривимірними аналогами, візуальні ефекти та унікальний графічний стиль відіграють ключову роль у створенні захопливого ігрового досвіду.

Розробка візуальних ефектів (VFX) - це багатогранний процес, що вимагає не лише художнього бачення, але й глибоких технічних знань, зокрема у сфері програмування шейдерів [15]. Шейдери, будучи невеликими програмами, що виконуються на графічному процесорі (GPU) [17], надають розробникам потужні інструменти для маніпулювання графікою в реальному часі, дозволяючи створювати ефекти, які раніше були недосяжні або вимагали значних обчислювальних ресурсів центрального процесора. Актуальність даної роботи полягає в дослідженні можливостей створення та оптимізації шейдерів для реалізації широкого спектру візуальних ефектів у 2D-ігрових проєктах, розроблених на популярній платформі Unity.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Предметне середовище розробки шейдерів та візуальних ефектів для 2D-ігор охоплює комплекс технологій, інструментів та методологій, спрямованих на створення та інтеграцію динамічних графічних елементів, що покращують візуальне сприйняття та інтерактивність ігрового процесу.

1.1.1. Роль та значення візуальних ефектів у 2D-іграх

Візуальні ефекти в 2D-іграх (рис. 1.1) виконують низку важливих функцій, що виходять за рамки простого естетичного оздоблення. По-перше, вони суттєво впливають на атмосферу та занурення гравця у віртуальний світ. Наприклад, ефекти погоди (дощ, сніг, туман) [1], динамічне освітлення, частинки (іскри, дим, магічні спалахи) [16] можуть кардинально змінити сприйняття однієї й тієї ж сцени, надаючи їй більшої глибини та реалістичності, навіть у стилізованій 2D-графіці.

По-друге, візуальні ефекти слугують важливим інструментом зворотного зв'язку для гравця. Вони можуть інформувати про результат дій персонажа (наприклад, спалах при влучанні, ефект поглинання шкоди), сигналізувати про небезпеку (червоне миготіння екрану при низькому здоров'ї), підкреслювати важливі ігрові об'єкти або події (світіння навколо ключового предмету, анімація активації порталу). Такий зворотний зв'язок робить ігровий процес більш інтуїтивним та зрозумілим.

По-третє, візуальні ефекти є невід'ємною частиною унікального візуального стилю гри [15]. Оригінальні та добре реалізовані ефекти можуть стати візитною карткою проєкту, виділяючи його серед безлічі інших ігор. Це особливо важливо для незалежних розробників, які прагнуть створити впізнаваний та запам'ятовуваний продукт.

По-четверте, VFX можуть підсилювати емоційний вплив ігрових моментів. Драматичні сцени можуть супроводжуватися відповідними ефектами (наприклад, сповільнення часу з розмиттям, ефект "старого кіно" для спогадів), а моменти тріумфу - яскравими та динамічними спалахами.

Отже, якісні візуальні ефекти є не розкішшю, а необхідністю для сучасних 2D-ігор, що прагнуть забезпечити гравцям глибокий, захопливий та емоційно насичений досвід.

Рисунок 1.1 – Приклади використання візуальних ефектів у популярних 2D-іграх (Ori)

1.1.2. Шейдери як інструмент створення візуальних ефектів

Основним інструментом для реалізації складних візуальних ефектів [2] у реальному часі є шейдери. Шейдер (англ. shader) - це програма, написана спеціальною мовою [4] (наприклад, HLSL, GLSL, Cg) [16], яка виконується безпосередньо на графічному процесорі (GPU). Шейдери дозволяють розробникам контролювати різні етапи графічного конвеєра [1], визначаючи, як об'єкти будуть відображені на екрані.

Графічний конвеєр (Rendering Pipeline) [2] - це послідовність етапів, через які проходять дані про 3D- або 2D-сцену перед тим, як вони перетворяться на фінальне зображення на екрані. У спрощеному вигляді, дані відправляються з центрального процесора (CPU) [17] на графічний процесор (GPU), де вони обробляються різними стадіями конвеєра (рис. 1.2). Ключовими програмованими стадіями, де використовуються шейдери, є:

1. Вершинний шейдер (Vertex Shader) [4] : Цей шейдер обробляє кожну вершину(точку) геометричного об'єкта (наприклад, кути спрайту або трикутника, що складає модель). Його основні завдання - це трансформація координат вершин [7] (перетворення з локального простору в простір камери, а потім у простір відсікання), обчислення текстурних координат, розрахунок освітлення на вершинах тощо.

2. Фрагментний (або піксельний) шейдер [4] (Fragment/Pixel Shader): Після того, як геометрія оброблена та растеризована (перетворена на набір пікселів/фрагментів), фрагментний шейдер визначає колір кожного окремого пікселя на екрані. Фрагментні шейдери є найбільш обчислювально інтенсивними, оскільки вони можуть виконуватися мільйони разів за кадр (по одному разу для кожного пікселя, покритого об'єктом).

Існують також інші типи шейдерів (геометричні, теселяційні, обчислювальні), але для створення більшості візуальних ефектів у 2D-іграх в Unity основними є вершинні та фрагментні шейдери.

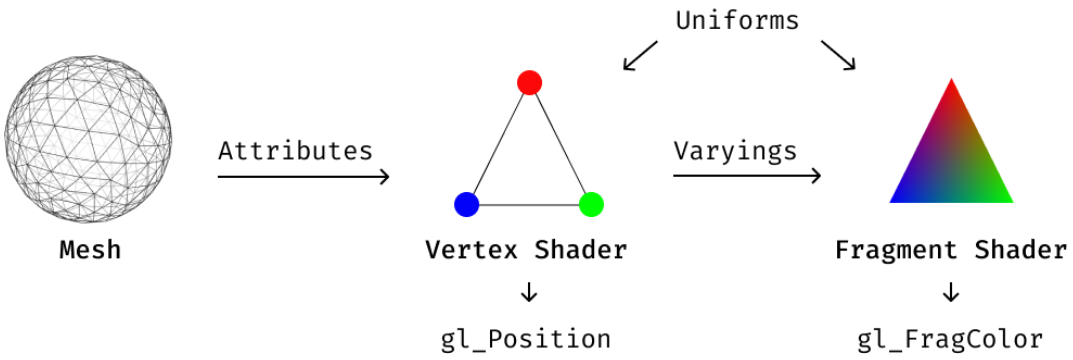


Рисунок 1.2 – Спрощена схема графічного конвеєра та місце шейдерів

Платформа Unity надає гнучку систему для роботи з шейдерами, дозволяючи використовувати як вбудовані шейдери, так і створювати власні за допомогою мови ShaderLab [3], яка, в свою чергу, інкапсулює код на HLSL [4] (High-Level Shading Language) або Cg (C for Graphics).

1.1.3. Специфіка застосування шейдерів у 2D-графіці та в Unity

Застосування шейдерів у 2D-графіці має свою специфіку, зумовлену особливостями рендерингу двовимірних об'єктів [7] , таких як спрайти та елементи користувацького інтерфейсу (UI).

У Unity 2D-об'єкти, зокрема спрайти, зазвичай представлені простими чотирикутниками (квадами), на які натягується текстура. Стандартний Sprite Renderer компонент використовує шейдери для відображення цих спрайтів, включаючи обробку прозорості, кольору та атласів текстур. Створюючи власні шейдери для спрайтів, розробник може розширити стандартну функціональність, наприклад:

- додати ефекти анімації появи/зникнення (dissolve, fade);
- реалізувати динамічні контури або підсвічування;
- створити ефекти спотворення (вода, теплова хвиля) безпосередньо на спрайті;
- застосувати складні ефекти маскування або змішування текстур.

Ефекти пост-обробки [19] (Post-processing effects) - це окремий клас візуальних ефектів, які застосовуються до всього зображення сцени після того, як вона була відрендерена, але перед тим, як вона буде відображена на екрані. Це потужний інструмент для покращення загального візуального вигляду гри, корекції кольору [20] , додавання атмосферних ефектів або стилізації зображення (рис 1.3).

В Unity ефекти пост-обробки зазвичай реалізуються за допомогою C# скриптів, прикріплених до камери, які використовують метод OnRenderImage(RenderTexture source, RenderTexture destination). Цей метод отримує відрендерене зображення сцени як текстуру (source), обробляє його

за допомогою одного або кількох шейдерів [1] (застосовуючи Graphics.Blit з відповідним матеріалом), і передає результат (destination) на наступний етап або на екран.

Рисунок 1.3 – Приклад застосування ефекту пост-обробки

Типові ефекти пост-обробки [2] , що реалізуються за допомогою шейдерів, включають:

- корекцію кольору: зміна тону, насиченості, контрастності зображення;
- світіння : створення ефекту світіння навколо яскравих об'єктів [18];
- розмиття : різні типи розмиття, наприклад, Motion Blur (розмиття в русі) або Depth of Field (глибина різкості);
- хроматичну аберацію : ефект кольорового розшарування по краях об'єктів;
- віньєтування : затемнення країв екрану для фокусування уваги на центрі;
- стилізаційні ефекти: ефект старого кіно [15] , пікселізація, CRT-ефект тощо.

Використання шейдерів для 2D-графіки та пост-обробки дозволяє досягти високого рівня візуальної якості та унікальності, що є важливим фактором конкурентоспроможності сучасних ігрових проєктів.

1.2. Аналіз підходів до реалізації візуальних ефектів

Ринок комп'ютерних ігор, зокрема 2D-сегмент, насичений проєктами з різноманітними візуальними стилями та ефектами. Аналіз існуючих рішень та підходів дозволяє визначити популярні тренди, ефективні техніки та потенційні напрямки для власних розробок.

1.2.1. Класифікація популярних візуальних ефектів у 2D-іграх

Візуальні ефекти, що застосовуються в 2D-іграх, можуть бути класифіковані за їх функціональним призначенням та техніками реалізації. Розглянемо деякі з них, що є релевантними для даної роботи.

До групи ефектів освітлення та атмосфери належить світіння (рис 1.4) [18]. Даний ефект імітує розсіювання світла від яскравих джерел, створюючи характерне м'яке сяйво та покращуючи візуальну привабливість сцени . Його реалізація часто включає виділення яскравих пікселів, їх розмиття (наприклад, фільтром Гаусса [2]) та адитивне накладання на зображення.

Рисунок 1.4 – Приклад застовування світіння

Категорія ефектів спотворення та стилізації зображення включає хроматичну аберацію, що імітує оптичні недосконалості лінз шляхом зміщення колірних каналів (рис 1.5). Цей ефект може використовуватися для передачі специфічних станів або стилізації. Для створення ретро-стилю застосовується комплексний ефект старого кіно або CRT-монітора, що

поєднує накладання шумів, сканувальних ліній, віньєтування та спотворення геометрії .

Рисунок 1.5 – Приклад хроматичної аберації

Ефекти трансформації та динаміки об'єктів представлені, зокрема, ефектом розчинення спрайту (Dissolve) [15] . Він забезпечує анімацію плавного зникнення або появи об'єкта шляхом маніпуляції видимістю пікселів на основі текстури шуму та порогового значення (рис 1.6). Для «оживлення» 2D-спрайтів використовуються ефекти спотворення (Distortion), що полягають у маніпуляції текстурними координатами для створення візуальних деформацій, таких як «теплова хвиля» або «пульсація».

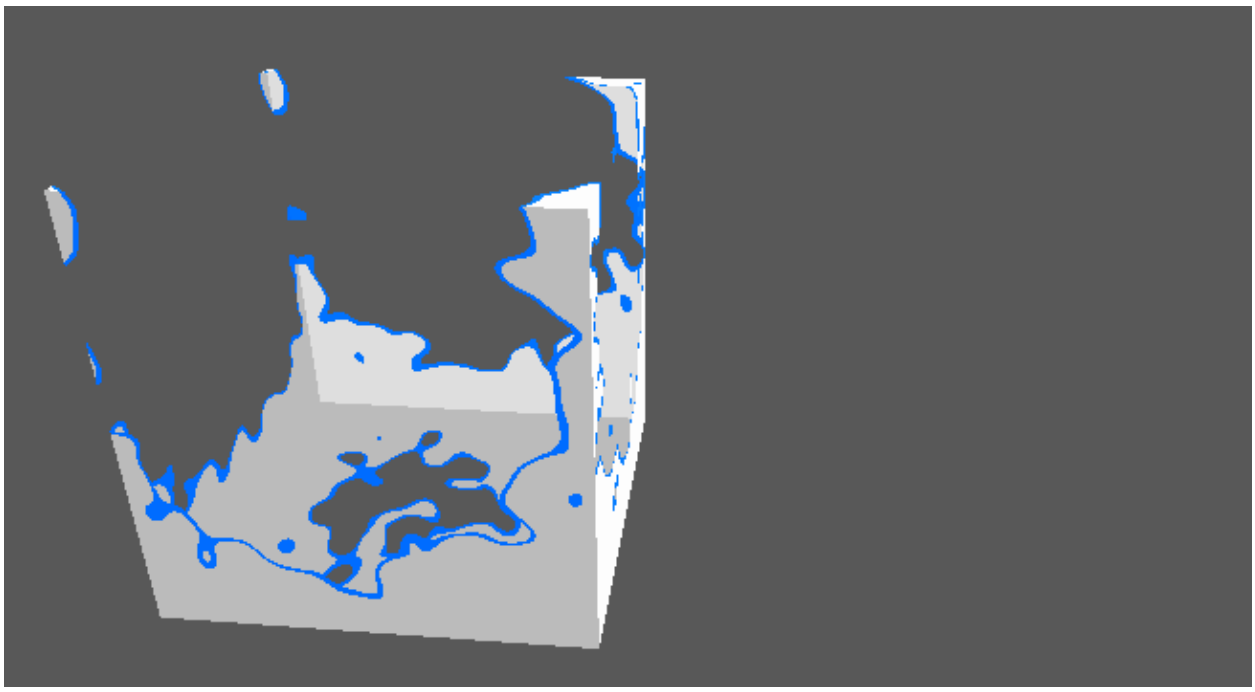


Рисунок 1.6 – Приклад розчинення спрайту

1.2.2. Огляд існуючих та самостійних рішень в Unity

Розробники в Unity мають декілька варіантів інтеграції візуальних ефектів (таблиця 1.1):

1. платформа пропонує стандартні шейдери та пакети для пост-обробки (наприклад, "Post-Processing Stack" або інтегровані рішення в URP/HDRP), що надають готові ефекти (Bloom, Color Grading тощо) з можливістю налаштування. Переваги: простота інтеграції, оптимізація. Недоліки: обмежена гнучкість для унікальних ефекті;

2. ринок пропонує широкий вибір платних та безкоштовних пакетів шейдерів та VFX. Переваги: економія часу, висока якість. Недоліки: вартість, можливі проблеми сумісності, обмеженість модифікації;

3. самостійна розробка шейдерів надає максимальну гнучкість, контроль та можливості для оптимізації. Використовуються техніки маніпуляції UV-координатами, noise-текстури, математичні функції, багатопрохідні шейдери. Даний підхід є центральним для цієї роботи.

Таблиця 1.1 – Порівняння підходів до створення візуальних ефектів в Unity

Критерій	Вбудовані засоби та пакети Unity	Ассети з Unity Asset Store	Самостійна розробка шейдерів
1	2	3	4
Переваги	Простота інтеграції. Базовий набір готових ефектів. Зазвичай добре оптимізовані Підтримка від Unity	Значна економія часу. Широкий вибір ефектів. Часто висока якість та наявність документації/підтрим ки	Максимальний контроль та гнучкість. Можливість реалізації унікальних ефектів Глибоке розуміння роботи ефекту Оптимізація під конкретні потреби проєкту
Недоліки	Обмежений набір ефектів. Недостатня гнучкість для унікальних завдань. Може не вистачати параметрів для тонкого налаштування	Вартість (багато якісних ассетів платні. Можливі проблеми сумісності. "Чорна скринька" (складність модифікації та розуміння. Якість може варіюватися	Вимагає значних часових затрат. Потрібні глибокі технічні знання (HLSL, ShaderLab). Вищий поріг входження

Продовження таблиці 1.1

1	2	3	4
Рівень контролю/гнучкості	Низький/ Середній	Середній (залежить від ассету)	Високий
Поріг входження	Низький	Низький / Середній (для використання)	Високий (для розробки)
Можливості оптимізації	Обмежені (залежить від Unity)	Обмежені (залежить від розробника ассету)	Високі
Унікальність результату	Низька (стандартні ефекти)	Середня / Висока (залежить від популярності ассету та налаштувань)	Дуже висока

1.2.3. Мови та інструменти для написання шейдерів в Unity

Основою для будь-якого шейдера є ShaderLab - власна декларативна мова Unity, що використовується для опису структури файлів з розширенням .shader. По суті, ShaderLab виступає як "контейнер" або "обгортка", яка визначає загальну архітектуру шейдера: його публічні властивості, що будуть доступні в інспекторі для налаштування художниками та дизайнерами, варіанти реалізації для різного обладнання (підшейдери), а також послідовність операцій рендерингу (проходи). Саме всередині цих проходів і розміщується основна логіка, написана на спеціалізованій мові програмування.

Основною мовою для написання самих обчислень, що виконуються на GPU, в Unity є HLSL (High-Level Shading Language) [4]. Ця потужна, C-подібна мова, розроблена Microsoft, є індустріальним стандартом для програмування під DirectX. Сильною стороною Unity є здатність забезпечувати крос-платформену компіляцію коду на HLSL для інших графічних API, таких як OpenGL чи Metal, що дозволяє створювати ефекти, які працюватимуть на широкому спектрі пристроїв. Робота з HLSL вимагає розуміння специфіки програмування для GPU, зокрема роботи з оптимізованими типами даних, векторною математикою, структурами для передачі даних та семантикою.

Для розробників, які прагнуть спростити процес створення шейдерів або віддають перевагу візуальному підходу, Unity пропонує інструмент Shader

Graph [6] . Це система візуального програмування, де шейдери створюються шляхом логічного з'єднання функціональних вузлів (нодів) у графічному редакторі (рис 1.7). Такий підхід значно знижує поріг входження та дозволяє швидко прототипувати складні візуальні ефекти, не заглиблюючись у написання коду вручну. Хоча Shader Graph є надзвичайно потужним, він може дещо поступатися в гнучкості та можливостях для глибокої оптимізації порівняно з прямим кодуванням на HLSL.

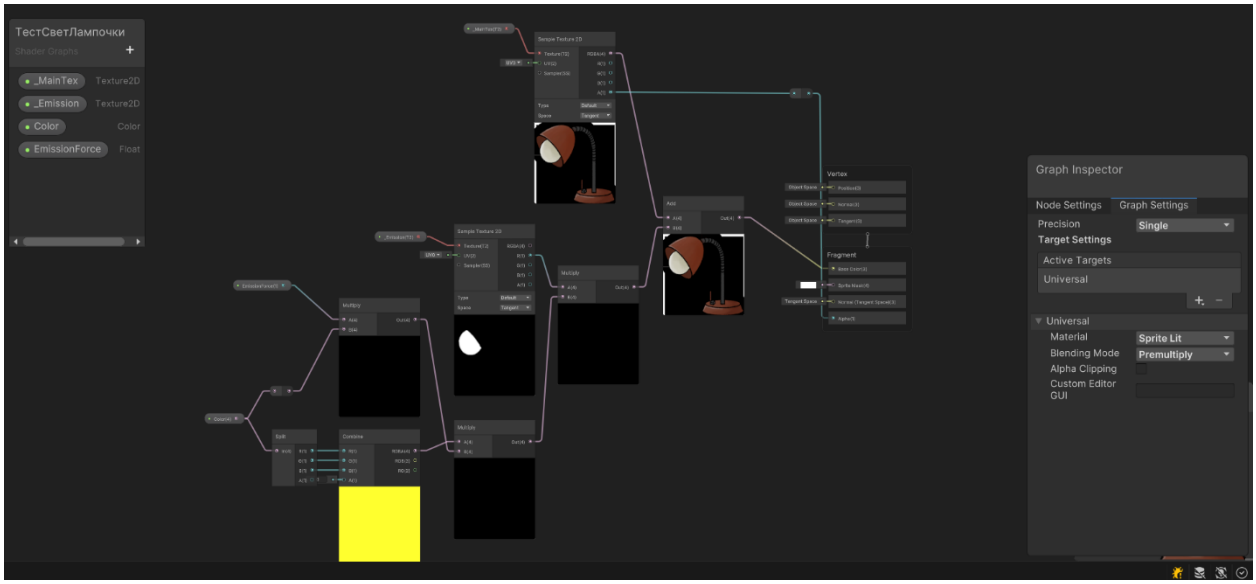


Рисунок 1.7 – Приклад створення шейдера за допомогою Shader Graph

У рамках даної кваліфікаційної роботи основний фокус зроблено на використанні Shader Graph для розробки логіки шейдерів, що дозволяє наочно продемонструвати процес створення ефектів, та ShaderLab для тонкого налаштування їхньої структури та інтеграції.

1.3. Постановка задачі дослідження та розробки

На основі проведеного аналізу предметного середовища та існуючих підходів, формулюється основна задача даної кваліфікаційної роботи, яка полягає у створенні та оптимізації набору шейдерів для реалізації ключових

візуальних ефектів у демонстраційному 2D-ігровому проєкті на платформі Unity.

1.3.1. Обґрунтування вибору конкретних візуальних ефектів

Для детального дослідження та демонстрації були обрані наступні візуальні ефекти:

- Світіння: один з найпопулярніших пост-ефектів, що значно покращує візуальну привабливість сцени, додаючи "м'якості" та "атмосферності" яскравим ділянкам. Його реалізація включає такі техніки, як виділення яскравих пікселів, розмиття та адитивне змішування, що є гарним прикладом багатопрохідного шейдера.
- Корекція кольору: потужний інструмент для художнього контролю над тональним діапазоном зображення, що дозволяє точно налаштовувати колірні канали або загальну яскравість для створення унікальних палітр та настроїв. Реалізується через lookup-текстури (LUT) [20] або обчислення на основі параметрів кривих.
- Зернистість: ефект, що імітує фотографічну зернистість або шум відеозапису, додаючи зображенню текстури та "аналогового" відчуття. Реалізується шляхом накладання процедурно згенерованого або засемпленого шуму.

Вибір саме цих ефектів зумовлений їх поширеністю в іграх, різноманітністю технік, що використовуються для їх реалізації (пост-обробка, спрайтові шейдери, робота з текстурами, математичні обчислення), а також потенціалом для застосування різних методів оптимізації.

1.3.2. Формулювання вимог до розроблюваних шейдерів

Для забезпечення якості та практичної застосовності розроблених рішень, до шейдерів висувається комплекс вимог:

1. кожен шейдер повинен функціонально коректно реалізовувати заявлений візуальний ефект, забезпечуючи при цьому високу якість візуалізації без артефактів.

2. критично важливою є гнучкість налаштування. Шейдери повинні мати набір публічних параметрів (інтенсивність, колір, швидкість тощо), доступних через інспектор Unity, що дозволить художникам та дизайнерам легко адаптувати ефекти під конкретні задачі без втручання в код.

3. ключовою вимогою є продуктивність. Шейдери мають бути розроблені з урахуванням оптимізації, щоб їх використання не спричиняло значних втрат частоти кадрів. Нарешті, вони повинні коректно інтегруватися в робочий процес Unity та мати читабельну структуру, що спростить їх подальшу підтримку та модифікацію.

1.3.3. Визначення критеріїв та методів оцінки оптимізації

В основі оцінки ефективності лежить аналіз впливу шейдера на ресурси графічного процесора [21]. Ключовим кількісним показником є час виконання шейдера на GPU (GPU time) [21], що вимірюється в мілісекундах (мс) на один кадр. Цей параметр безпосередньо впливає назагальну частоту кадрування (FPS) і є головним індикатором продуктивності візуального ефекту.

Для глибшого розуміння причин навантаження аналізуватимуться й більш низькорівневі метрики. До них належить, по-перше, кількість виконаних інструкцій у скомпільованому шейдері, особливо у його фрагментній частині, оскільки саме вона виконується для кожного пікселя. По-друге, це кількість вибірок з текстур [2] (texture samples), оскільки звернення до відеопам'яті є однією з найбільш ресурсоємних операцій. Окрім обчислювального навантаження, до уваги братиметься і використання відеопам'яті (VRAM) [17] , що є актуальним для ефектів, які використовують проміжні текстури великої роздільної здатності, як-от «Світіння» (Bloom).

Для реалізації такого комплексного аналізу задіяно набір спеціалізованих інструментів, інтегрованих у середовище розробки Unity. Основним засобом для отримання кількісних даних про продуктивність є Unity Profiler, а саме його вкладка для аналізу GPU. Цей інструмент дозволяє в реальному часі відстежувати час виконання кожного етапу рендерингу та ідентифікувати найбільш ресурсоємні шейдери. Для більш детального,

покрокового аналізу використовується Frame Debugger. Він надає можливість дослідити кожен виклик рендерингу (draw call) в межах одного кадру, переглянути стан графічного конвеєра, використані ресурси та, що найважливіше, отримати інформацію про кількість інструкцій скомпільованого шейдера.

Важливо зазначити, що оптимізація не є самоціллю, а пошуком оптимального балансу між продуктивністю та візуальною якістю. Тому невід'ємною частиною оцінки є візуальний аналіз, який полягає в суб'єктивному порівнянні зображення до та після оптимізації. Мета - переконатися, що приріст продуктивності не призвів до неприйнятного погіршення візуальної якості ефекту.

Висновки до розділу

У цьому розділі було проведено аналіз предметного середовища, який показав, що візуальні ефекти є невід'ємною частиною сучасних 2D-ігор. Вони виконують ключові функції: створюють атмосферу, забезпечують зворотний зв'язок для гравця та формують унікальний візуальний стиль. Основним технічним інструментом для їх реалізації є шейдери - програми, що виконуються на графічному процесорі та дозволяють гнучко керувати відображенням як окремих об'єктів (спрайтів), так і всієї сцени за допомогою ефектів пост-обробки.

На основі аналізу існуючих підходів, що включають використання вбудованих засобів Unity, ассетів та самостійної розробки, було зроблено висновок про переваги останнього методу в контексті гнучкості та оптимізації. Таким чином, сформульовано головну задачу роботи: створення набору кастомізованих та оптимізованих шейдерів (світіння, корекція кольору, зернистість). Ключовими вимогами до них є продуктивність та наявність параметрів для налаштування, а оцінка ефективності буде проводитись шляхом аналізу впливу на час рендерингу GPU та візуальної якості.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

Перехід від теоретичного огляду до практичної реалізації вимагає глибокого аналізу інформаційних потоків та математичних моделей, що лежать в основі роботи шейдерів. Цей розділ присвячений детальному розгляду вхідних та вихідних даних, що використовуються шейдерами, а також алгоритмічному та математичному апарату, необхідному для створення обраних візуальних ефектів. Розуміння цих аспектів є фундаментом для ефективного проектування, реалізації та подальшої оптимізації програмного коду шейдерів.

2.1. Аналіз предметної області візуальних ефектів

Процес рендерингу, керований шейдерами, по суті, є перетворенням даних. Шейдери отримують набір вхідної інформації про геометрію, текстури та параметри сцени, виконують над нею обчислення та повертають результат, який і формує фінальне зображення. Детальний аналіз цих даних є першим кроком у проектуванні будь-якого шейдера.

2.1.1. Вхідні дані для шейдерів

Вхідні дані для шейдерів можна умовно поділити на кілька категорій, кожна з яких відіграє свою унікальну роль у створенні візуальних ефектів.

Дані вершин (Vertex Data) є базовою інформацією про геометрію об'єкта. Для 2D-ігор, де об'єкти (спрайти) та ефекти пост-обробки (повноекранний квад) представлені простими чотирикутниками, основними даними є координати вершин (POSITION) та їхні текстурні координати (TEXCOORD0) [7] . Вершинний шейдер використовує ці дані для правильного позиціонування об'єкта на екрані та для передачі текстурних координат у фрагментний шейдер, що дозволяє коректно накласти текстуру (рис 2.1).

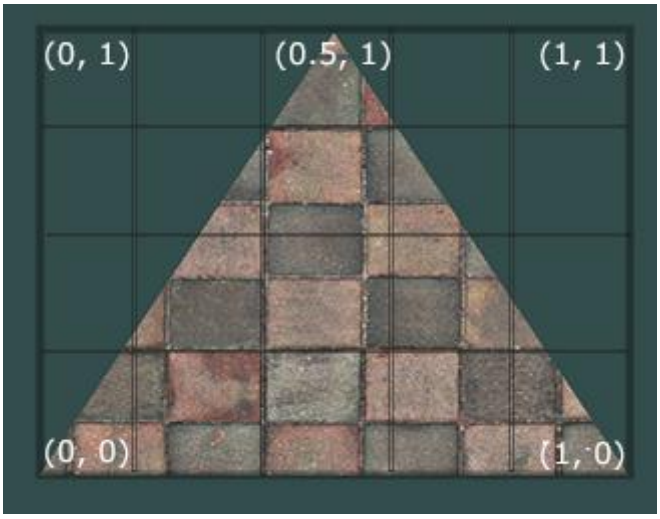


Рисунок 2.1 – Приклад схематичного накладання текстури

Текстурні дані є одними з найважливіших ресурсів для шейдерів, особливо у 2D. Залежно від ефекту, використовуються різні типи текстур:

- основна текстура екрану: для ефектів пост-обробки, таких як Світіння, Корекція кольору та Зернистість, вхідними даними є весь відрендерений кадр сцени, що передається в шейдер як звичайна 2D-текстура. В Unity це реалізується через метод OnRenderImage, який надає доступ до цієї текстури;
- текстури шуму: критично важливі для ефекту Зернистості. Це можуть бути заздалегідь підготовлені текстури, що містять «аналоговий» шум, або процедурно згенеровані текстури. Вони зазвичай є безшовними (tileable), що дозволяє покривати ними екран будь-якого розміру без видимих стиків;
- lookup-таблиці (LUT): спеціалізовані 3D-текстури, що є основою для ефективної реалізації ефекту Корекції кольору. LUT (Look-Up Table) являє собою «кольоровий куб», де кожен піксель містить інформацію про те, як перетворити певний вхідний колір у вихідний. Використання LUT дозволяє застосовувати складні кольорові перетворення за одну операцію вибірки з текстури, що є значно швидшим, ніж виконання складних математичних обчислень для кожного пікселя.

Уніфіковані змінні (Uniform Variables або Properties) - це параметри, які визначаються розробником та передаються з CPU у шейдер для керування

ефектом. Вони дозволяють художникам та дизайнерам гнучко налаштовувати вигляд ефекту в редакторі Unity без зміни коду. Для обраних шейдерів ці змінні є критично важливими, що ілюструє таблиця 2.1.

Таблиця 2.1 – Приклади уніфікованих змінних для розроблюваних шейдерів

Шейдер	Приклади уніфікованих змінних (Properties)	Призначення
Світіння (Bloom)	_Threshold (float), _Intensity (float), _BlurSize (float)	Поріг яскравості для виділення пікселів, інтенсивність світіння, радіус розмиття.
Корекція кольору (Curves)	_LUT (3D Texture), _Intensity (float)	Lookup-текстура для перетворення кольорів, інтенсивність застосування ефекту.
Зернистість (Film Grain)	_Intensity (float), _Scale (float), _Speed (float)	Інтенсивність шуму, масштаб (розмір) "зерна", швидкість анімації шуму.

Глобальні змінні Unity (Built-in Variables) - це набір змінних, які Unity автоматично надає всім шейдерам. Вони містять корисну інформацію про поточний стан рендерингу. Наприклад, _Time містить час з моменту запуску гри і є незамінним для анімації ефектів. _ScreenParams містить інформацію про роздільну здатність екрану, що важливо для коректної роботи з екранними координатами.

2.1.2. Вихідні дані шейдерів

Результатом роботи шейдера, як правило, є кінцевий колір пікселя, що буде відображений на екрані. Однак, у деяких випадках вихідні дані можуть бути більш складними.

Колір пікселя (Pixel Color) є основним виходом фрагментного шейдера. Це значення типу float4 або half4, що містить інформацію про червоний, зелений, синій канали та альфа-канал (прозорість) для кожного конкретного пікселя.

Проміжні текстури (Render Textures) є специфічним типом вихідних даних для багатопрохідних ефектів, таких як Світіння (Bloom). Замість того, щоб одразу виводити результат на екран, шейдер може рендерити свій

результат у тимчасову текстуру (render texture). Наприклад, на першому етапі Bloom результат (маска яскравих пікселів) записується в одну текстуру, потім ця текстура розмивається і записується в іншу, і лише на останньому етапі результат комбінується з основним зображенням. Цей процес «рендерингу в текстуру» є потужним інструментом для реалізації складних ефектів.

2.1.3. Стисла характеристика демонстраційного 2D-проєкту

Для інтеграції та тестування розроблених шейдерів було створено демонстраційний проєкт, що являє собою статичну 2D-сцену, яка відтворює інтер'єр житлового приміщення в нічний час. Вибір саме такого середовища є обґрунтованим, оскільки воно містить елементи з різними характеристиками освітлення та поверхнями, що дозволяє наочно та всебічно продемонструвати роботу кожного з ефектів.

Сцена побудована таким чином, щоб забезпечити релевантний контекст для застосування та оцінки розроблених шейдерів:

- Світіння: ефект тестується на двох ключових типах джерел світла, що дозволяє оцінити гнучкість шейдера. По-перше, це локальне, інтенсивне джерело світла - електрична лампочка, яка дає змогу чітко налаштувати та оцінити дію параметра Threshold (Поріг яскравості) та побачити концентроване сяйво (рис 2.2). По-друге, це м'яке, розсіяне світло, що проникає з вікна іншої кімнати. Цей елемент дозволяє перевірити, як шейдер працює з менш інтенсивними, але більш масштабними ділянками світіння, демонструючи вплив параметрів Intensity (Інтенсивність) та радіуса розмиття (рис 2.3).

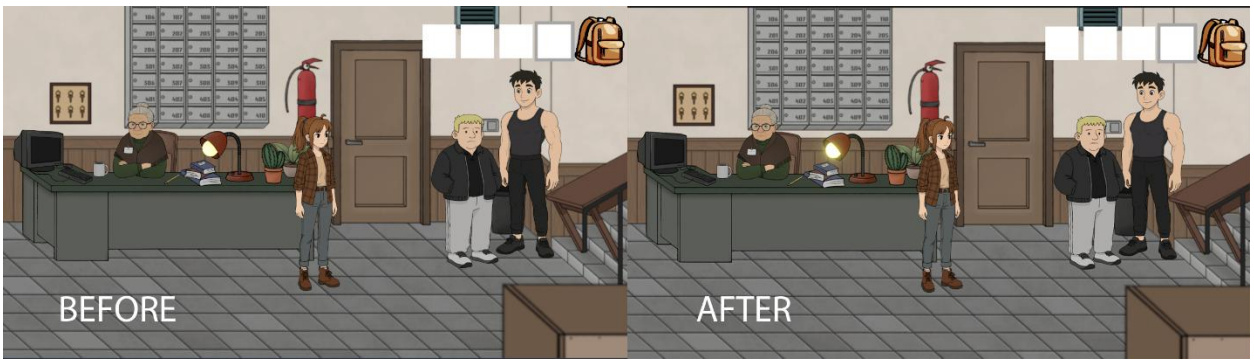


Рисунок 2.2 – Приклад роботи шейдера на лампочці

Рисунок 2.3 – Прикад роботи шейдера біля вікна

- Корекція кольору: хоча сцена є єдиним простором, вона містить ділянки з кардинально різним рівнем освітленості - від яскравих зон біля лампи та вікна до глибоких тіней у кутках кімнати. Це створює ідеальні умови для тестування шейдера корекції кольору. Застосовуючи його до всієї камери, можна наочно побачити, як змінюється загальна атмосфера сцени. Використання інструментів, що лежать в основі шейдера, як-от криві (Color Curves) та налаштування тіней, середніх тонів і світлих ділянок (Shadows, Midtones, Highlights), дозволяє оцінити його здатність комплексно працювати з усім тональним діапазоном зображення для досягнення бажаного художнього стилю (рис 2.4).

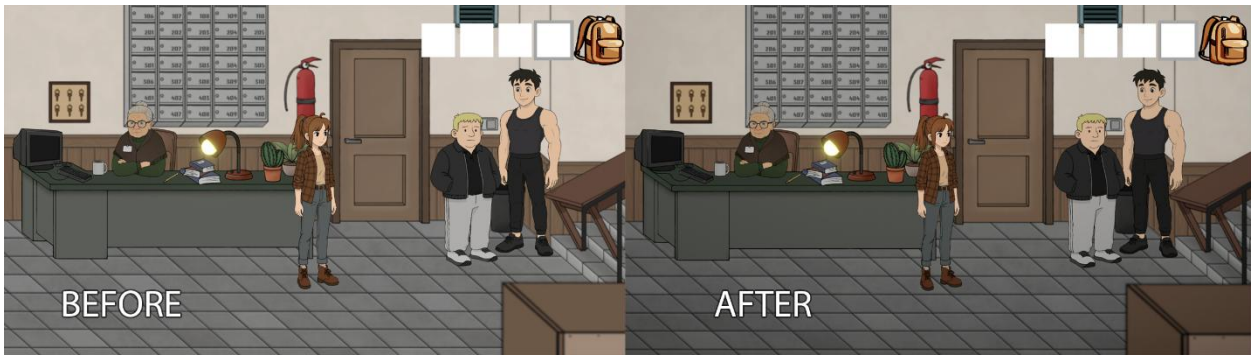


Рисунок 2.4 – Вплив эфекту корекції кольору

- Зернистість: даний ефект, що застосовується до всієї камери, є чудовим прикладом тонкої візуальної стилізації (рис 2.5). На відміну від агресивних фільтрів, що імітують стару плівку, реалізований шейдер зернистості створює

ледь помітну текстуру. Його мета - не кардинально змінити зображення, а делікатно позбавити його стерильної цифрової різкості та додати глибини.

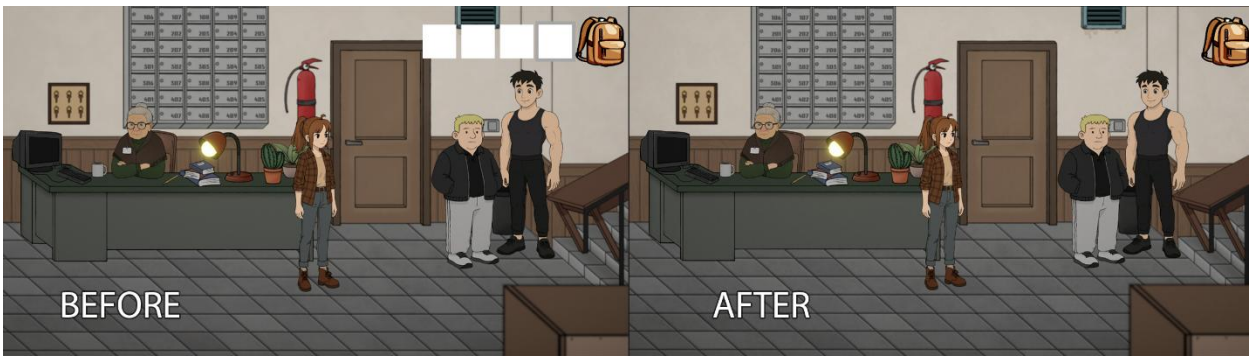


Рисунок 2.5 – Вплив эфекту зернистості

Така конфігурація демонстраційного проєкту забезпечить релевантний контекст для застосування та оцінки розроблених шейдерів, дозволяючи перевірити не тільки їх візуальну якість, але й функціональність у середовищі, наближеному до реальної гри.

2.3. Математичне та алгоритмічне забезпечення шейдерів

Якщо інформаційний аналіз відповідає на питання "що?", то математичне та алгоритмічне забезпечення відповідає на питання "як?". Цей підрозділ розкриває математичні принципи та алгоритми, що є ядром кожного з розроблюваних шейдерів.

2.3.1. Ключові змінні та типи даних в HLSL

Ефективність шейдера значною мірою залежить від правильного вибору типів даних для змінних. HLSL пропонує кілька типів даних для чисел з плаваючою комою, що відрізняються точністю та, відповідно, продуктивністю:

- float: 32-бітна точність. Є стандартним і найбільш точним типом, який слід використовувати для даних, що вимагають високої точності, наприклад, для координат вершин або складних математичних розрахунків.

- half: 16-бітна точність. Має менший діапазон значень і точність, але обробляється значно швидше на багатьох GPU, особливо на мобільних платформах. Ідеально підходить для зберігання текстурних координат (UV) та векторів напрямку, де надвисока точність не є критичною.
- fixed: 10-12-бітна точність. Найменш точний, але найшвидший тип. Його діапазон значень зазвичай обмежений [-2.0, 2.0]. Найкраще застосування - для зберігання даних про колір, оскільки людське око не здатне розрізнити дрібні відхилення в кольорі, які можуть виникнути через низьку точність.

Правильне використання цих типів даних (half та fixed [21] там, де це можливо) є однією з ключових технік оптимізації шейдерів, яка буде детально розглянута в наступному розділі. Окрім скалярних типів, широко використовуються векторні (float2, half3, fixed4), матричні (float4x4) та спеціальні типи для роботи з текстурами (sampler2D для 2D-текстур, sampler3D для 3D LUT) [7] .

2.3.2. Опис загальних алгоритмів

Розглянемо алгоритми для кожного з ефектів детальніше.

Алгоритм ефекту «Світіння» [18] . Bloom є класичним багатопрохідним ефектом. Його алгоритм можна розбити на наступні етапи:

1. Виділення яскравих ділянок (Bright-pass): на першому етапі на вхід подається текстура всієї сцени. Шейдер проходить по кожному пікселю і відфільтровує ті, яскравість яких перевищує певне порогове значення (_Threshold). Пікселі, що не пройшли перевірку, стають чорними. Результат записується у проміжну текстуру (рис 2.6).

```
// Фрагмент коду з фрагментного шейдера для Bright-pass
fixed4 frag(v2f i) : SV_Target
{
    // Отримуємо колір пікселя з текстури сцени
    fixed4 color = tex2D(_MainTex, i.uv);

    // Розраховуємо яскравість (спрощений варіант)
    fixed brightness = dot(color.rgb, fixed3(0.2126, 0.7152, 0.0722));

    // Якщо яскравість менша за поріг, робимо піксель чорним, інакше залишаємо
    // Використовуємо step, щоб уникнути умовних розгалужень
    fixed cutoff = step(_Threshold, brightness);

    return color * cutoff;
}
```

Рисунок 2.6 – Фрагмент коду реалізації Bright-pass фільтра на HLSL

2. Послідовне розмиття (Downsampling & Blurring): для досягнення м'якого та широкого розмиття при збереженні високої продуктивності, текстура з яскравими ділянками послідовно зменшується в розмірі (downsampling) в 2, 4, 8 разів. Кожна зменшена версія розмивається за допомогою фільтра (найчастіше, фільтра Гаусса). Це значно швидше, ніж розмивати повнорозмірне зображення.

3. Композитинг (Upsampling & Additive Blending): розмиті текстури послідовно збільшуються до початкового розміру (upsampling) і адитивно (додаванням) накладаються одна на одну та на оригінальне зображення сцени (рис 2.7). Це створює фінальний ефект м'якого світіння.

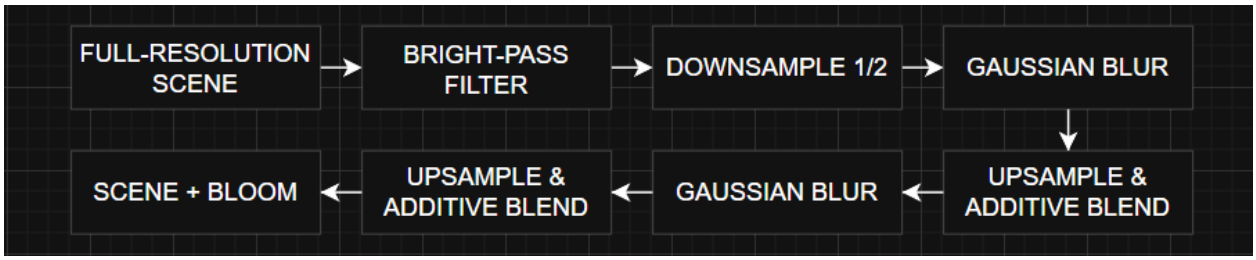


Рисунок 2.7 – Блок-схема алгоритму ефекту «Світіння»

Алгоритм ефекту «Корекція кольору» за допомогою кривих (LUT). Цей алгоритм базується на принципі перетворення кольору за допомогою попередньо обчисленої таблиці, що забезпечує високу продуктивність.

1. Підготовка LUT: поза грою, у графічному редакторі, до нейтральної LUT (текстури-градієнта) застосовуються необхідні корекції кольору (криві, рівні тощо). Змінена текстура зберігається і завантажується в Unity.

2. Перетворення кольору в шейдері [15] : фрагментний шейдер отримує оригінальний колір пікселя зі сцени. Значення його червоного (R), зеленого (G) та синього (B) каналів, що знаходяться в діапазоні [0, 1], використовуються як 3D-координати для вибірки нового кольору з 3D-текстури LUT. Наприклад, вхідний колір (r, g, b) перетворюється на вибірку tex3D(_LUT, float3(r, g, b)).

3. Змішування: отриманий з LUT новий колір змішується з оригінальним на основі параметра інтенсивності, що дозволяє плавно регулювати силу ефекту.

Алгоритм ефекту «Зернистість» (Film Grain) Ефект зернистості може бути реалізований двома основними способами:

1. Текстурний підхід: використовується заздалегідь підготовлена безшовна текстура шуму. У шейдері ця текстура семплюється з використанням екранних координат, і її значення (зазвичай, після зміщення в діапазон [-0.5, 0.5]) додається до кольору пікселя сцени. Для анімації "зерна" текстурні координати можна зміщувати з часом.

2. Процедурний підхід: шум генерується "на льоту" для кожного пікселя за допомогою математичних функцій (рис. 2.8). Це дозволяє уникнути використання додаткової текстури. Простий спосіб - використати хеш-функцію, яка на основі екранних координат та поточного часу генерує псевдовипадкове число. Це число і є інтенсивністю "зерна" для даного пікселя.

```
// Функція для генерації простого псевдовипадкового шуму
float random(float2 st)
{
    return frac(sin(dot(st.xy, float2(12.9898, 78.233))) * 43758.5453123);
}

// ... у фрагментному шейдері ...
fixed4 frag(v2f i) : SV_Target
{
    fixed4 originalColor = tex2D(_MainTex, i.uv);

    // Генеруємо шум, додаючи час для анімації
    float noise = random(i.uv + _Time.y) - 0.5; // Зміщуємо в діапазон [-0.5, 0.5]

    // Додаємо шум до оригінального кольору з певною інтенсивністю
    fixed3 finalColor = originalColor.rgb + noise * _Intensity;

    return fixed4(finalColor, originalColor.a);
}
```

Рисунок 2.8 – Приклад процедурної генерації шуму для ефекту зернистості

Висновки до розділу

У цьому розділі було проведено детальний аналіз інформаційної основи для розробки шейдерів, що є фундаментом для їх практичної реалізації. Було систематизовано вхідні дані, які включають дані вершин (позиція, текстурні координати), різноманітні текстури (основна текстура екрану, текстури шуму, 3D-таблиці LUT для корекції кольору) та параметри керування (уніфіковані та глобальні змінні Unity). Також визначено вихідні дані, що представлені не тільки фінальним кольором пікселя, але й проміжними текстурами, які є ключовими для реалізації багатопрохідних ефектів, як-от «Світіння». Створено концепцію демонстраційної сцени для адекватного тестування кожного ефекту в релевантному ігровому контексті.

Крім того, було розкрито математичне та алгоритмічне забезпечення, що лежить в основі обраних візуальних ефектів. Для ефекту «Світіння» описано багатоетапний алгоритм, що включає виділення яскравих ділянок, послідовне розмиття та композитинг. Для «Корекції кольору» представлено

високоефективний підхід на основі вибірки з LUT, а для «Зернистості» - два альтернативні методи: текстурний та процедурний. Було закладено основу для подальшої оптимізації, проаналізувавши ключові типи даних в HLSL (float, half, fixed) та їхній вплив на точність і продуктивність.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

Успішна реалізація та оптимізація шейдерів неможлива без відповідного інструментарію та чітко визначених вимог до програмного та апаратного середовища. Цей розділ присвячений детальному опису засобів розробки, що використовуються в даній роботі, інструментів для аналізу продуктивності, а також конфігурації технічного та програмного забезпечення, на базі якого проводиться дослідження. Вибір правильних інструментів та дотримання системних вимог є запорукою стабільного та ефективного процесу розробки.

3.1. Засоби розробки та аналізу шейдерів

Комплекс інструментів, що застосовується в даній роботі, охоплює як основне середовище розробки Unity з його внутрішніми компонентами, так і спеціалізовані засоби для аналізу продуктивності та написання коду.

3.1.1. Середовище розробки Unity

Unity є інтегрованим середовищем розробки (IDE), що надає повний набір інструментів для створення інтерактивного контенту, зокрема 2D- та 3D-ігор. Для розробки шейдерів ключовими є наступні компоненти та системи Unity (рис 3.1):

- Редактор Unity: основний робочий простір, де відбувається конфігурація сцен, об'єктів та компонентів. Редактор надає візуальний інтерфейс для роботи з матеріалами, що використовують розроблювані шейдери, дозволяючи в реальному часі бачити зміни та налаштовувати параметри ефектів через інспектор.
- Система матеріалів (Materials System): матеріал в Unity є ассетом, що пов'язує шейдер з конкретними параметрами (текстурами, кольорами, числовими значеннями). Саме через матеріали відбувається застосування

шейдерів до об'єктів сцени. Система матеріалів дозволяє створювати численні варіації одного шейдера з різними налаштуваннями, що є надзвичайно гнучким інструментом для художників та дизайнерів.

- Компонент Sprite Renderer: основний компонент для відображення 2D-графіки (спрайтів) у Unity. Він відповідає за рендеринг чотирикутника з натягнутою на нього текстурою. Для застосування кастомного ефекту, такого як Спотворення спрайту, стандартний матеріал цього компонента замінюється на матеріал, що використовує розроблений шейдер.

- Компонент Camera: камера є центральним елементом для реалізації ефектів пост-обробки. Ефекти Світіння, Корекція кольору та Зернистість застосовуються до всього зображення, яке бачить камера. В рамках Universal Render Pipeline (URP), який використовується в даній роботі, це досягається шляхом інтеграції кастомних ефектів у конвеєр рендерингу через систему Renderer Features. Камера збирає всю візуальну інформацію сцени, яка потім передається на обробку ланцюжку пост-ефектів.

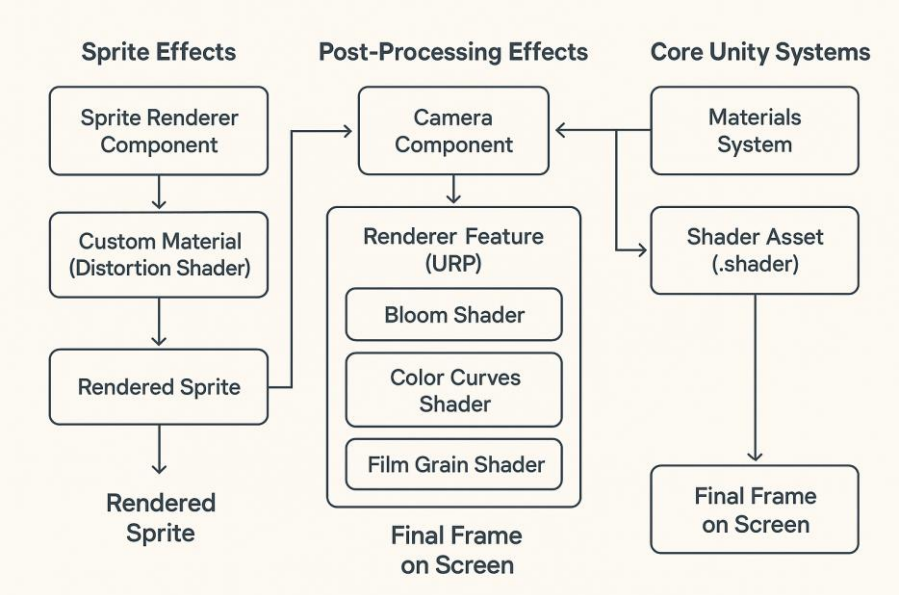


Рисунок 3.1 – Схема взаємодії ключових компонентів Unity при рендерингу ефектів

3.1.2. Інструменти для аналізу продуктивності

Оптимізація є невід'ємною частиною розробки шейдерів. Для об'єктивної оцінки продуктивності та виявлення «вузьких місць» використовуються спеціалізовані інструменти, вбудовані в Unity.

Unity Profiler [5] - це потужний інструмент для аналізу продуктивності програми в реальному часі. Для завдань даної роботи найбільш важливим є GPU Profiler (рис 3.2). Він дозволяє відстежувати час, який графічний процесор витрачає на різні етапи рендерингу кадру. За його допомогою можна точно визначити, скільки мілісекунд займає виконання конкретного пост-ефекту або рендеринг об'єктів з певним матеріалом. Це дозволяє кількісно оцінити вплив шейдера на загальну частоту кадрів (FPS) та порівнювати продуктивність різних версій шейдера до та після оптимізації. Аналіз таких метрик, як Draw Calls, SetPass Calls та час виконання окремих блоків рендерингу, є основою для прийняття рішень щодо оптимізації.

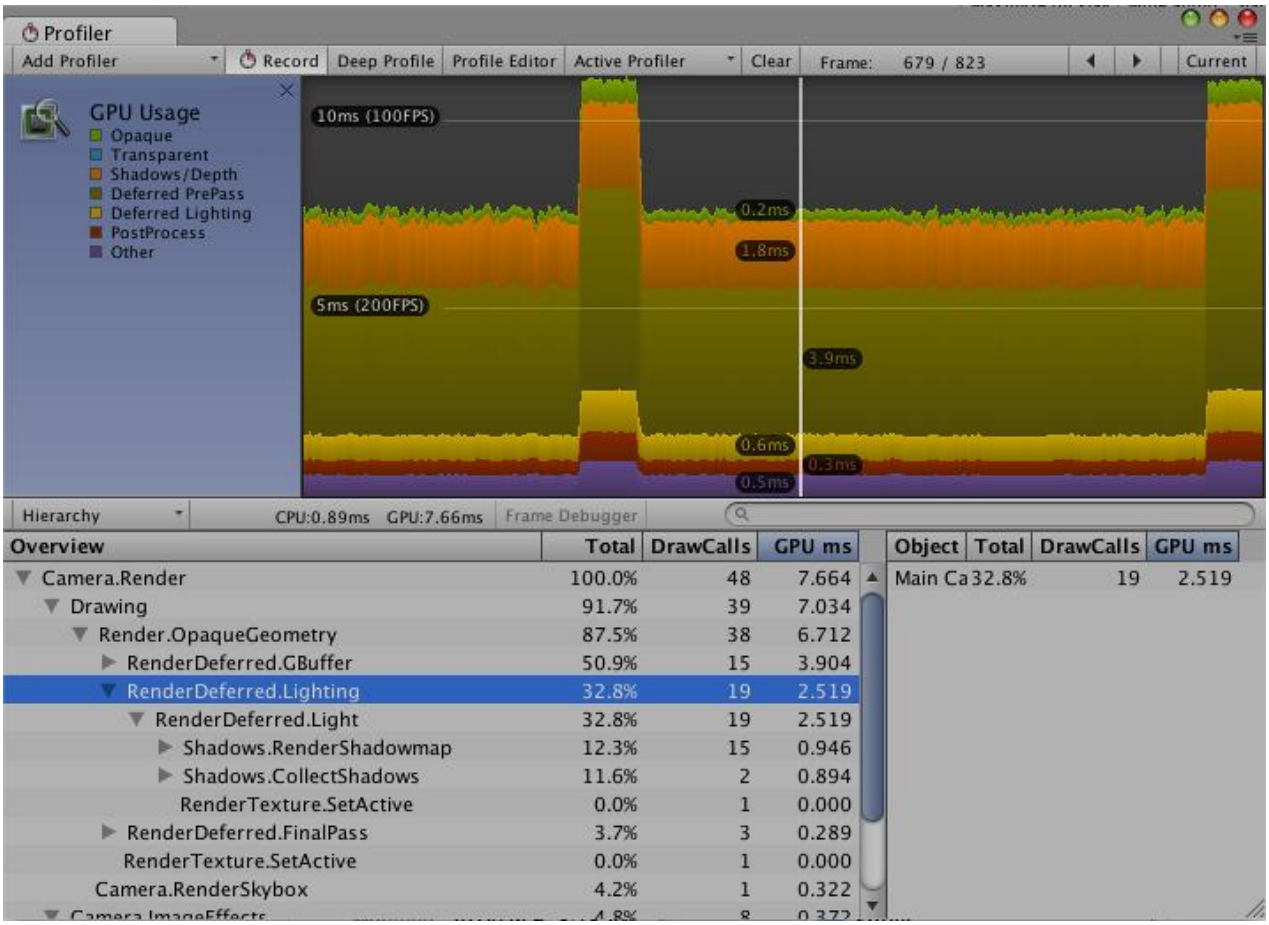


Рисунок 3.2 – Інтерфейс вікна GPU Profiler в Unity

Frame Debugger [5] - цей інструмент надає можливість «поставити на паузу» рендеринг одного кадру та покроково проаналізувати всі команди, які були відправлені на GPU для його створення (рис 3.3). Frame Debugger дозволяє побачити точну послідовність викликів рендерингу (Draw Calls), візуалізувати стан сцени на кожному етапі, перевірити, які саме шейдери та матеріали були використані для кожного об'єкта, та переглянути детальні параметри шейдера (вхідні текстури, змінні). Для розробника шейдерів це безцінний інструмент для відладки, який допомагає зрозуміти, чому ефект виглядає не так, як очікувалося, або чому рендеринг відбувається неефективно (наприклад, виконуються зайві проходи).

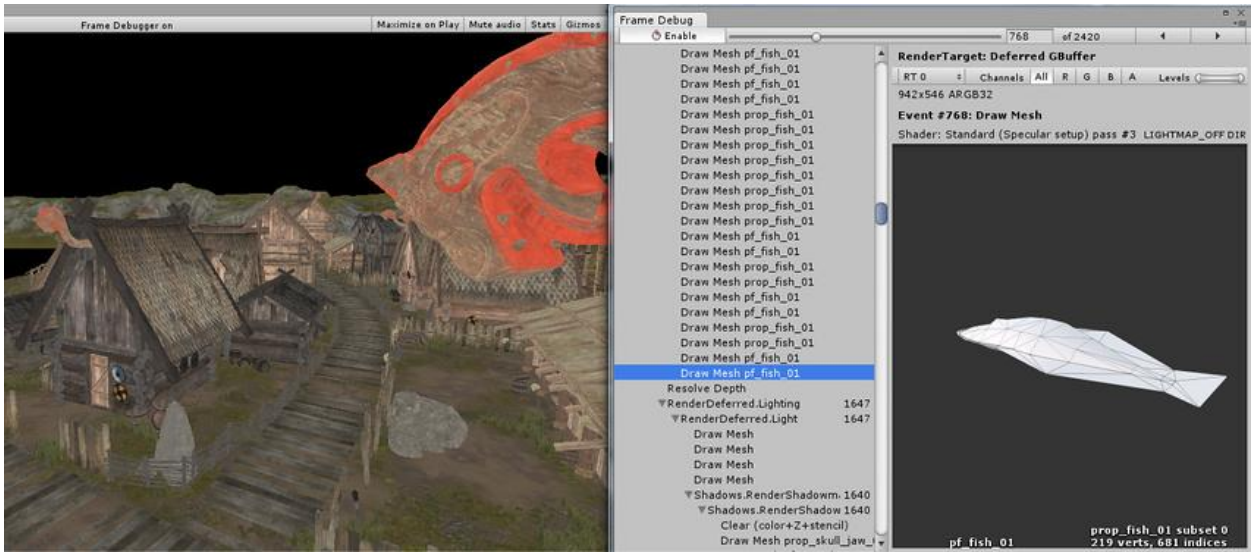


Рисунок 3.3 – Інтерфейс Frame Debugger

3.1.3. Текстовий редактор для написання коду шейдерів

Хоча писати код шейдерів можна в будь-якому текстовому редакторі, для підвищення продуктивності та зручності розробки в даній роботі використовується інтегроване середовище розробки JetBrains Rider (рис 3.4). Rider надає низку переваг при роботі з проєктами Unity, що поширюються і на розробку шейдерів:

- підсвічування синтаксису: Rider коректно розпізнає та підсвічує синтаксис як мови ShaderLab, так і вбудованих блоків HLSL, що значно покращує читабельність коду;

- автодоповнення коду та аналіз: хоча підтримка HLSL не така глибока, як для C#, Rider надає базове автодоповнення для ключових слів та вбудованих функцій, а також може виявляти синтаксичні помилки ще до компіляції в Unity;

- інтеграція з Unity: глибока інтеграція дозволяє швидко переходити між C# скриптами, що керують шейдерами, та самими .shader файлами. Rider також відображає повідомлення з консолі Unity, що спрощує процес відладки помилок компіляції шейдерів;

- зручна навігація та рефакторинг: надає потужні інструменти для навігації по кодовій базі та рефакторингу, що є корисним при роботі зі складними шейдерами або супутніми C# скриптами.

Використання професійного IDE, такого як Rider, дозволяє оптимізувати робочий процес, зменшити кількість рутинних помилок та зосередитись на алгоритмічній та творчій складовій розробки шейдерів.

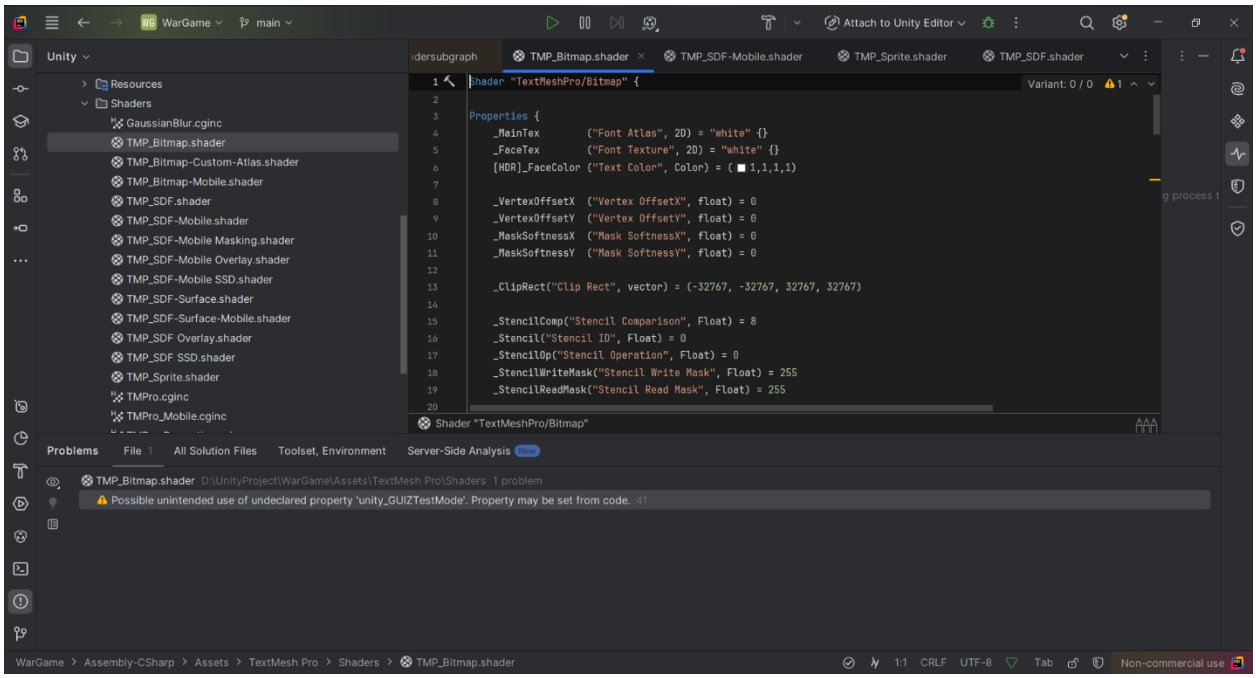


Рисунок 3.4 – IDE Rider

3.2. Вимоги до технічного та програмного забезпечення

Для забезпечення відтворюваності результатів дослідження та стабільності процесу розробки необхідно чітко визначити конфігурацію програмного та апаратного забезпечення.

3.2.1. Системні вимоги для роботи з Unity та розробки шейдерів

Розробка в Unity, особливо робота з графікою та шейдерами, висуває певні вимоги до апаратної частини робочої станції. Хоча Unity може працювати на досить широкому спектрі обладнання, для комфортної розробки та профілювання рекомендовано дотримуватися наступних або кращих характеристик(таблиця 3.1)

Таблиця 3.1 – Рекомендовані мінімальні вимоги до системи

Компонент	Вимоги
Операційна система	Windows 10/11 (64-bit), macOS, Linux
Процесор (CPU)	Багатоядерний x64 з підтримкою SSE2
Оперативна пам’ять	Мінімум 8 ГБ, рекомендовано 16 ГБ+
Відеокарта (GPU)	Дискретна, з підтримкою DirectX 11/12, OpenGL або Metal; бажано сучасна для шейдерів
Місце на диску	SSD рекомендовано; достатньо для Unity, IDE та проєкту

Дотримання цих вимог забезпечує стабільну роботу середовища розробки та дозволяє отримувати коректні дані при аналізі продуктивності.

3.2.2. Версія Unity та необхідні пакети

Для уникнення проблем із сумісністю та забезпечення стабільності всі розробки в рамках даної роботи ведуться на конкретній версії Unity та з використанням визначеного набору пакетів.

Використовується Unity 2022.3.18f1 (LTS). Вибір версії з довготривалою підтримкою (Long-Term Support) гарантує максимальну стабільність, наявність виправлень критичних помилок та мінімізує ризик виникнення

несподіваних проблем, пов'язаних з оновленнями редактора в процесі роботи над проєктом.

Проєкт конфігуровано для роботи з Universal Render Pipeline (URP) [5] . URP - це сучасний, гнучкий та оптимізований конвеєр рендерингу від Unity, який добре підходить для широкого спектру платформ, включаючи 2D-ігри. Використання URP має кілька ключових переваг для даної роботи:

- дозволяє легко кастомізувати конвеєр рендерингу, зокрема, додавати власні ефекти пост-обробки через механізм Renderer Features та ScriptableRenderPass. Це є більш сучасним та потужним підходом порівняно з методом OnRenderImage у старому, вбудованому конвеєрі (Built-in Render Pipeline).
- за замовчуванням оптимізований для ефективної роботи, зокрема, він автоматично використовує пакетну обробку (batching) для зменшення кількості викликів рендерингу.
- усі сучасні інструменти Unity, такі як Shader Graph [6] та VFX Graph, найкраще інтегруються саме з URP.

3.3. Опис програмної реалізації та оптимізації шейдерів

Програмна реалізація візуальних ефектів у даному проєкті базується на ефективному використанні та конфігуруванні вбудованих інструментів і шейдерів, що надаються конвеєром рендерингу Universal Render Pipeline (URP). Такий підхід дозволяє досягати високої візуальної якості, зберігаючи гнучкість налаштувань. Цей розділ розкриває, як саме були реалізовані ключові ефекти через систему Матеріалів та систему Об'ємів, ілюструючи принципи їхньої роботи за допомогою прикладів у Shader Graph та фрагментів коду на HLSL.

3.3.1. Програмна реалізація візуальних ефектів

3.3.1.1 Реалізація локального світіння через шейдер матеріалу

Для ефекту світіння настільної лампи було обрано підхід, що полягає у модифікації властивостей її матеріалу. Ключовим елементом є використання карти емісії (_Emission). Для ілюстрації цього принципу можна розглянути спрощену реалізацію такого шейдера в Shader Graph (рис 3.5).

Логіка такого шейдера є досить простою. Вона складається з кількох ключових вузлів (нодів):

- властивості (Properties): створюються властивості для основної текстури (_MainTex), карти емісії (_Emission), кольору емісії (EmissionColor) та її сили (EmissionForce).
- семплювання текстур: два вузли Sample Texture 2D використовуються для отримання кольору з основної текстури та значення з карти емісії.
- логіка емісії: значення з карти емісії (зазвичай, червоний канал) множиться на колір та силу емісії.
- комбінування: результат логіки емісії додається до кольору основної текстури за допомогою вузла Add.
- виведення: фінальний колір подається на вихідний вузол Fragment у канал Base Color.

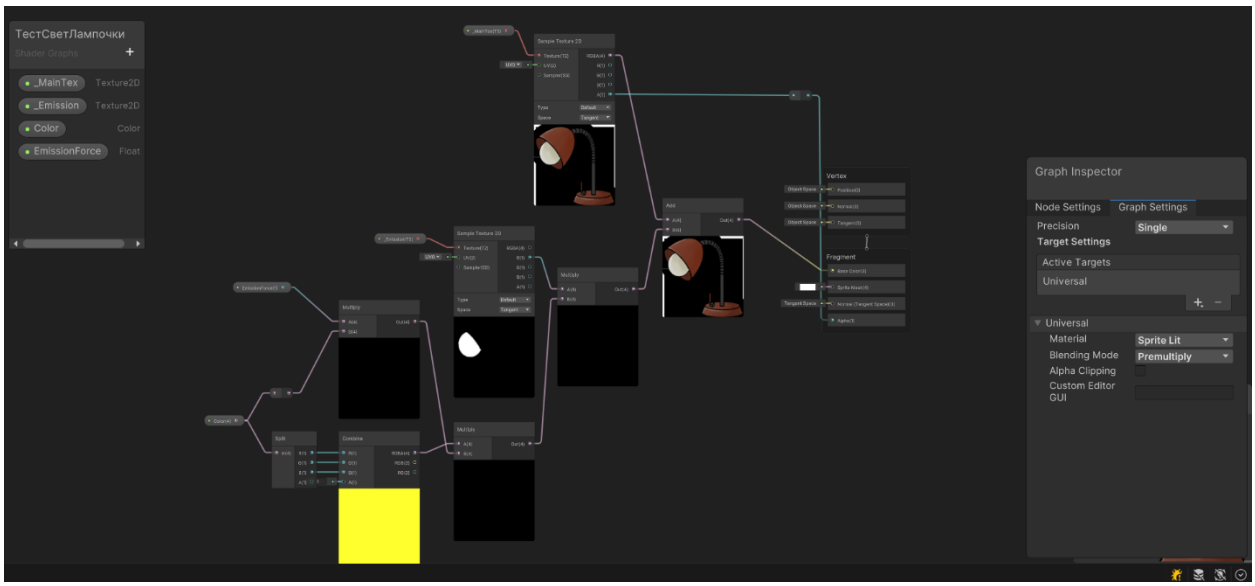


Рисунок 3.5 – Спрощена схема реалізації шейдера з емісією в Shader Graph

Така реалізація дозволяє GPU в рамках одного проходу рендерингу обчислити як базовий колір об'єкта, так і світіння, що є надзвичайно ефективним.

3.3.1.2 Реалізація пост-ефектів через систему Volumes

Глобальні ефекти, такі як корекція кольору та зернистість, реалізовані через систему Volume Framework, яка послідовно застосовує до зображення шейдери пост-обробки. Розглянемо принципи їхньої роботи.

В основі ефекту «Корекція кольору» лежить технологія Look-Up Table (LUT). LUT - це, по суті, тривимірна «таблиця» (3D-текстура), яка зберігає попередньо обчислені значення кольорів. Замість того, щоб виконувати складні математичні операції над кожним пікселем, шейдер просто використовує оригінальний колір пікселя як координату для "пошуку" нового кольору в цій таблиці.

Цей підхід є надзвичайно швидким, оскільки основна обчислювальна робота (зміна кривих) виконується один раз при створенні LUT, а в реальному часі відбувається лише одна операція вибірки з текстури (див. додаток А).

Ефект зернистості може бути реалізований двома шляхами: накладанням готової текстури шуму або його процедурною генерацією «на льоту» для кожного пікселя. Другий підхід дозволяє уникнути використання додаткових текстурних файлів. Шум генерується за допомогою простих хеш-функцій (див. додаток Б).

3.3.2. Підходи до оптимізації реалізованих ефектів

Оскільки в проєкті використовуються переважно вбудовані шейдери, оптимізація полягає у прийнятті правильних архітектурних рішень.

Ключовим архітектурним рішенням стала відмова від глобального пост-ефекту Світіння на користь локального світіння через матеріал. Глобальне Світіння є значно більш ресурсоємним, оскільки вимагає виконання серії проходів по всьому зображенню, тоді як обраний підхід з картою емісії

виконує всі обчислення в одному проході і лише для пікселів конкретного об'єкта, що значно економить ресурси GPU.

Іншим важливим методом є перцептивна оптимізація, що полягає в налаштуванні параметрів ефектів з урахуванням людського сприйняття. Налаштування інтенсивності Film Grain на низьке значення 0.27 є чудовим прикладом: ефект залишається присутнім, але його можна безболісно вимкнути на слабких системах без значної шкоди для загальної атмосфери, що демонструє важливість балансу між якістю та продуктивністю.

На рівні коду оптимізація досягається через використання типів даних з меншою точністю, як це показано у наведених прикладах шейдерів. Застосування fixed та half замість float для зберігання кольорів та проміжних обчислень дозволяє зменшити навантаження на GPU без видимої втрати якості зображення.

3.4. Керівництво з використання розроблених шейдерів

Розробка функціональних та оптимізованих шейдерів є лише частиною завдання. Не менш важливим є забезпечення можливості їх зручного та ефективного використання іншими членами команди розробки - художниками, дизайнерами рівнів чи іншими програмістами. Даний підрозділ являє собою детальне керівництво, що описує процес інтеграції розроблених шейдерів у проєкт Unity, а також надає вичерпну інформацію щодо параметрів кожного ефекту та їхнього впливу на фінальний візуальний результат.

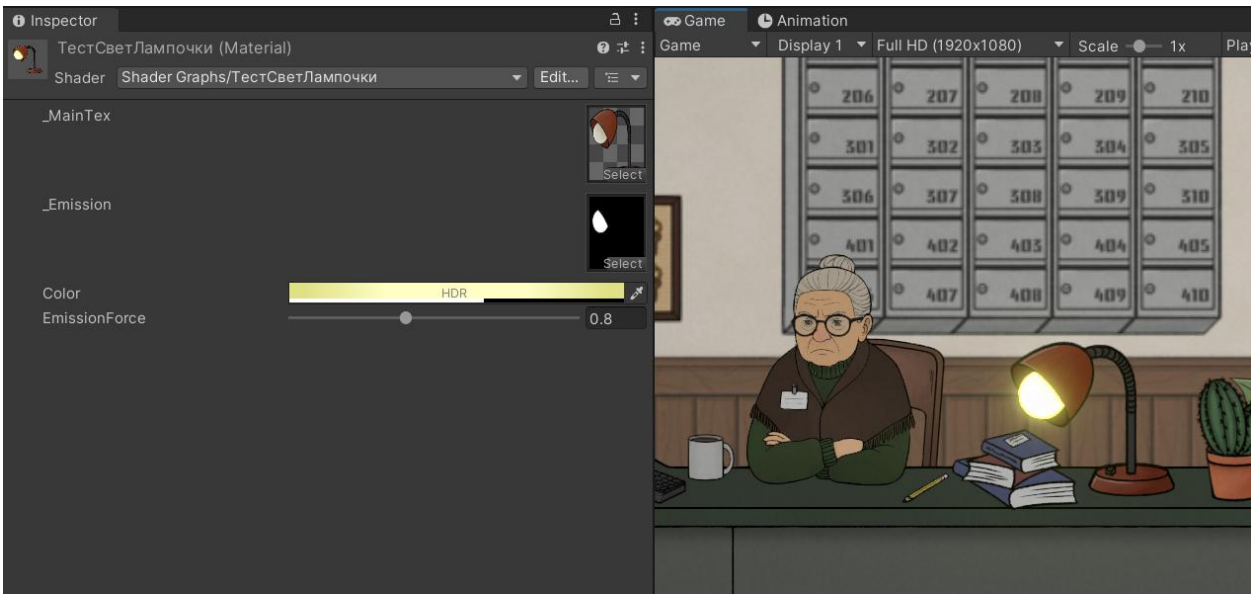
3.4.1. Опис параметрів кожного шейдера та їхній вплив на візуальний результат

Для досягнення бажаного візуального стилю в проєкті використовується набір шейдерів, параметри яких гнучко налаштовуються безпосередньо в редакторі Unity. Це дозволяє в реальному часі контролювати вигляд ефектів та адаптувати їх під художні задачі. Розглянемо ключові параметри для кожного з реалізованих ефектів.

3.4.1.1 Створення ефекту світіння матеріалу

У проєкті світіння реалізовано не як глобальний пост-ефект, а як властивість матеріалу конкретних об'єктів, що дозволяє досягти більш точкового та контрольованого результату. Яскравим прикладом є настільна лампа, світіння якої керується двома основними параметрами в шейдері (рис 3.6).

Першим і найважливішим параметром є карта емісії (_Emission). Це спеціальна чорно-біла текстура, що працює як маска: шейдер змушує світитися лише ті ділянки об'єкта, які на цій карті позначені білим кольором. Такий підхід дозволяє дуже точно визначити форму світіння.



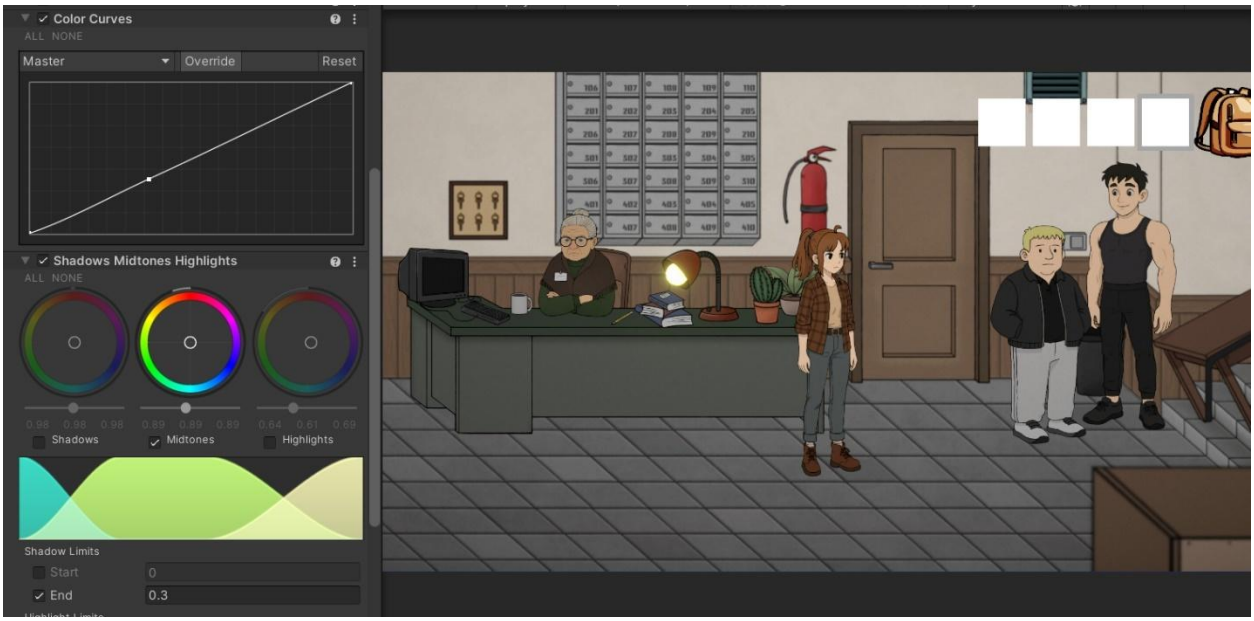
3.6 – Параметри матеріалу спрайта

Як продемонстровано на зображенні, для лампи карта емісії змушує світитися лише саму лампочку та її найближчий ореол, залишаючи решту об'єкта, як-от абажур, без змін. Другим параметром є сила емісії (EmissionForce), що діє як множник яскравості. Встановлене значення 0.8 створює м'яке, але помітне світіння, яке виглядає природно в контексті сцени та гармонійно взаємодіє з базовим освітленням від системи Light 2D, використаної для створення розсіяного світла з вікна.

3.4.1.2 Корекція кольору сцени

Загальна атмосфера та настрій сцени задаються за допомогою пост-ефектів корекції кольору, що застосовуються до всієї камери через Global Volume. У проєкті для цього використовується комбінація двох потужних інструментів. Перший - «Криві кольору» (Color Curves), який дозволяє керувати контрастом. Другий, і ключовий для створення художнього стилю, - це «Кольорові кола» (Shadows, Midtones, Highlights) (рис 3.7).

Цей інструмент дозволяє додати окремий колірний відтінок до трьох основних тональних діапазонів зображення.



3.7 – Налаштування ефектів корекції кольору

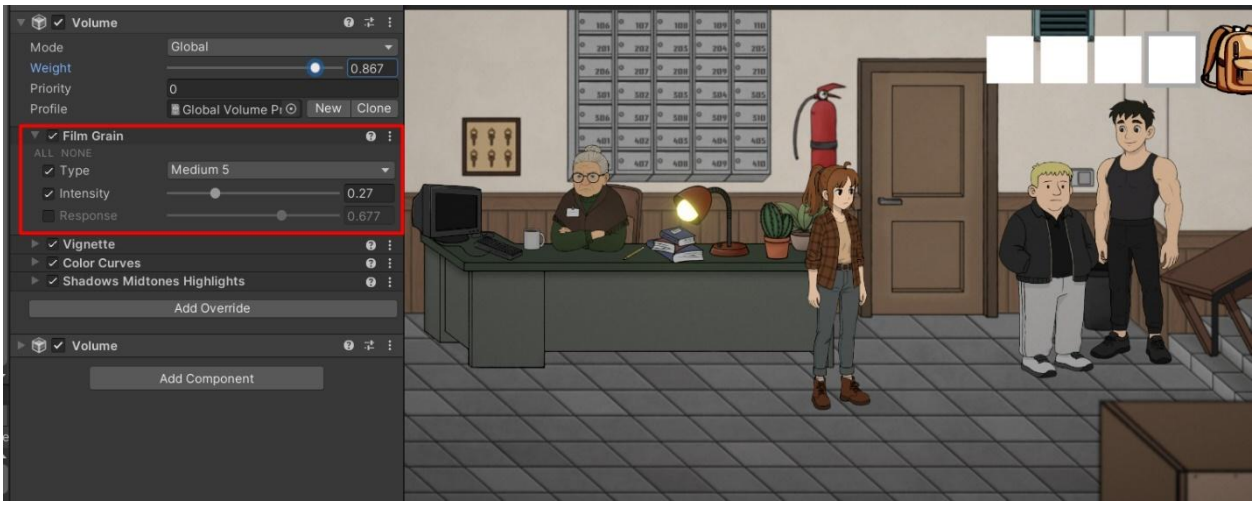
Налаштування цього ефекту, як видно на скріншоті, є вирішальним для створення атмосфери. Легке тонування тіней у холодні відтінки, а середніх тонів - у теплі, дозволяє зробити зображення більш об'ємним, глибоким та «кінематографічним», відходячи від нейтральної цифрової палітри.

3.4.1.3 Стилізація за допомогою ефекту «Зернистість»

Ефект «Зернистість» використовується для фінальної стилізації зображення. Його мета в даному проєкті - не створити агресивний ефект старої плівки, а додати ледь помітну текстуру, яка позбавляє картинку цифрової

стерильності та об'єднує всі елементи сцени в єдине ціле. Для цього використовуються два ключові параметри.

Перший - це «Інтенсивність» (Intensity), що визначає загальну видимість «зерна». Встановлене низьке значення 0.27 робить ефект дуже делікатним, він не привертає до себе уваги, але додає зображенню аналогового шарму (рис 3.8).



3.8 – Параметри ефекту «Зернистість»

Другий важливий параметр - «Відгук» (Response), який контролює розподіл зернистості по кадру. Значення 0.677 дозволяє імітувати поведінку справжньої плівки, де шум менш помітний у найсвітліших та найтемніших ділянках. Як видно на зображенні, поєднання цих точних налаштувань дозволяє досягти бажаного тонкого та художньо виправданого результату.

3.4.2. Інструкції щодо інтеграції шейдерів у Unity проєкт

Інтеграція розроблених шейдерів у проєкт на базі Universal Render Pipeline (URP) вимагає виконання кількох кроків для кожного типу ефектів. Інтеграція ефектів пост-обробки (Bloom, Color Curves, Film Grain).

Процес інтеграції пост-ефектів у URP є більш структурованим, ніж у старому конвеєрі, і виконується через Renderer Features:

1. Створення C# скрипта для ефекту: для кожного пост-ефекту створюється C# скрипт, що успадковується від ScriptableRenderPass. У цьому

скрипті описується логіка: коли саме в циклі рендерингу має виконуватися ефект, як отримати доступ до текстури сцени (_CameraColorTexture), як створити тимчасові текстури (Render Textures) та як виконати Blit з використанням матеріалу шейдера.

2. Створення C# скрипта "Renderer Feature": створюється ще один C# скрипт, що успадковується від ScriptableRendererFeature. Його завдання - створити екземпляр нашого ScriptableRenderPass та додати його до конвеєра рендерингу. Цей скрипт також виставляє публічні поля для налаштувань ефекту, які будуть відображені в інспекторі.

3. Налаштування URP Asset:

- у налаштуваннях проєкту (Project Settings -> Graphics) необхідно переконатися, що вибрано ассет UniversalRenderPipelineAsset;
- вибрати цей ассет у вікні Project, знайти його Renderer Data (або створити новий);
- в інспекторі Renderer Data натиснути кнопку «Add Renderer Feature» та вибрати наш новостворений «Renderer Feature» зі списку;
- після цього в інспекторі з'являться налаштування ефекту (наприклад, поріг яскравості для Bloom), які можна редагувати. Ефект буде автоматично застосовуватися до всіх камер, що використовують цей Renderer Data.

Інтеграція шейдера для спрайту (Sprite Distortion). Цей процес є значно простішим і відповідає стандартному робочому процесу Unity:

1. Створення Матеріалу: у вікні Project клацнути правою кнопкою миші та вибрати Create -> Material. Назвати матеріал, наприклад, DistortionMaterial.

2. Призначення шейдера: в інспекторі новоствореного матеріалу у випадаючому списку «Shader» знайти та вибрати розроблений шейдер (наприклад, Custom/SpriteDistortion).

3. Налаштування параметрів: після вибору шейдера в інспекторі матеріалу з'являться всі його властивості (_MainTex, _DistortionMap, _Strength тощо).

Необхідно перетягнути в відповідні слоти основну текстуру спрайту та карту спотворень, а також налаштувати числові параметри.

4. Застосування матеріалу до спрайту: знайти у сцені об'єкт зі компонентом Sprite Renderer, до якого потрібно застосувати ефект. У компоненті Sprite Renderer знайти поле Material та перетягнути в нього створений DistortionMaterial. Після цього спрайт буде рендеритися з використанням кастомного шейдера.

Дотримання цих інструкцій дозволяє коректно та ефективно інтегрувати розроблені шейдери в будь-який проєкт, що використовує Universal Render Pipeline, та надає можливість гнучкого керування їхніми візуальними аспектами.

Висновки до розділу

У цьому розділі визначено технічний та інструментальний фундамент роботи. Було описано ключові засоби розробки, зокрема середовище Unity версії 2022.3.18f1 (LTS) з конвеєром рендерингу URP, що є основою для інтеграції ефектів. Для об'єктивної оцінки продуктивності та відладки залучено вбудовані інструменти Unity Profiler та Frame Debugger. Також обґрунтовано вибір IDE JetBrains Rider для написання коду та сформульовано системні вимоги до програмного та апаратного забезпечення для забезпечення стабільності та відтворюваності результатів.

Друга частина розділу є практичним керівництвом для користувача, що забезпечує зручну інтеграцію розроблених рішень. Було надано вичерпний опис публічних параметрів кожного шейдера ("Світіння", "Корекція кольору", "Зернистість", "Спотворення спрайту"), з детальним поясненням їхнього впливу на фінальний візуальний результат. На завершення, сформульовано чіткі покрокові інструкції щодо інтеграції ефектів пост-обробки через механізм Renderer Features в URP та застосування спрайтових шейдерів через

систему матеріалів Unity, що робить їх доступними для використання художниками та дизайнерами.

РОЗДІЛ 4
ОЗОРОНА ПРАЦІ

Сучасний процес розробки програмного забезпечення, зокрема у сфері комп'ютерної графіки, характеризується високою інтенсивністю інтелектуальної праці та тривалим перебуванням працівника за комп'ютером. Хоча ІТ-галузь не належить до виробництв із підвищеною небезпекою, вона має низку специфічних ризиків, пов'язаних із психофізіологічними навантаженнями, особливостями робочого середовища та загальними загрозами. Тому створення безпечних та комфортних умов праці для розробника є невід'ємною складовою ефективності та успішності будь-якого проєкту.

Цей розділ присвячений аналізу питань охорони праці в контексті діяльності розробника шейдерів. У ньому розглядаються законодавчі основи регулювання безпеки праці, виявляються потенційні небезпеки, проводиться оцінка ризиків та надаються рекомендації щодо їх мінімізації.

4.1. Регулювання питань охорони праці на законодавчому рівні

Забезпечення безпечних та здорових умов праці в Україні є конституційним правом кожного громадянина та пріоритетним напрямом державної політики. Правовою основою системи управління охороною праці (СУОП) [12] є комплекс законодавчих та нормативно-правових актів, що встановлюють права, обов'язки та відповідальність усіх суб'єктів трудових відносин [8].

Основним документом, що гарантує право на безпечні умови праці, є Конституція України. Стаття 43 [13] прямо зазначає: «Кожен має право на належні, безпечні і здорові умови праці...». Ця норма є базовою і знаходить свою конкретизацію в інших законодавчих актах.

Ключовим документом у цій сфері є Закон України "Про охорону праці". Він визначає основні положення щодо реалізації конституційного права працівників, регулює відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні. Закон поширюється на всі підприємства, установи та організації незалежно від форми власності та виду діяльності.

Трудові відносини, зокрема в аспекті робочого часу, часу відпочинку та дисциплінарної відповідальності за порушення норм охорони праці, регулюються Кодексом законів про працю України (КЗпП) [8].

Для працівників ІТ-сфери, чия діяльність пов'язана з постійною роботою за комп'ютером, важливе значення мають специфічні санітарні норми [9]. Донедавна ключовим документом були «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» (ДСанПіН 3.3.2.007-98). Хоча цей документ втратив чинність, його положення щодо ергономіки робочого місця, режимів праці та відпочинку залишаються актуальними як рекомендована найкраща практика. Сучасні вимоги до параметрів мікроклімату (температура, вологість, швидкість руху повітря) в офісних приміщеннях встановлюються «Санітарними нормами мікроклімату виробничих приміщень» (ДСН 3.3.6.042-99).

Реалізація цих законодавчих положень на практиці залежить від чіткого розподілу ролей та відповідальності:

1. Роботодавець. На нього покладається головний обов'язок зі створення безпечних умов праці. Згідно зі ст. 13 Закону "Про охорону праці", роботодавець зобов'язаний:

- Створити на робочому місці умови праці відповідно до нормативно-правових актів.
- Забезпечити функціонування системи управління охороною праці.
- Організувати проведення інструктажів, навчання та перевірки знань працівників з питань охорони праці.

- Забезпечити працівників необхідними засобами індивідуального та колективного захисту (наприклад, ергономічними меблями, якісним освітленням).

- Здійснювати фінансування та контроль за виконанням заходів з охорони праці.

2. Працівник (розробник). Його відповідальність полягає у дотриманні встановлених норм. Згідно зі ст. 14 Закону, працівник зобов'язаний:

- Дбати про особисту безпеку і здоров'я, а також про безпеку оточуючих.

- Знати і виконувати вимоги нормативно-правових актів з охорони праці та правил поводження з обладнанням.

- Проходити обов'язкові медичні огляди.

- Співпрацювати з роботодавцем у справі організації безпечних умов праці та негайно повідомляти про виникнення небезпечних ситуацій.

3. Фахівець з охорони праці. На великих підприємствах (понад 50 працівників) створення такої служби є обов'язковим. Фахівець з ОП виконує контролюючу та консультативну функцію: розробляє інструкції, проводить вступні інструктажі, контролює дотримання норм, бере участь у розслідуванні нещасних випадків.

4. Державні органи. Головним органом державного нагляду є Державна служба України [14] з питань праці (Держпраці). Її повноваження включають проведення перевірок, видачу приписів про усунення порушень, накладання штрафів та призупинення робіт, що створюють загрозу життю чи здоров'ю працівників.

Таким чином, законодавство створює комплексну систему, де кожен учасник трудового процесу має свої чітко визначені права та обов'язки, що в сукупності спрямовані на збереження життя та здоров'я працівників.

4.2. Виявлення потенційних небезпек для розробника шейдерів

Робоче місце розробника шейдерів, як і будь-якого ІТ-фахівця, являє собою офісне приміщення, оснащене персональним комп'ютером (ПК) та периферійним обладнанням. Хоча таке середовище здається відносно безпечним, воно містить низку потенційних шкідливих та небезпечних факторів, які можна класифікувати на кілька груп.

Фізичні небезпечні фактори:

- Ураження електричним струмом [11] . Ця небезпека пов'язана з можливим пошкодженням ізоляції кабелів живлення комп'ютера, монітора, мережевих фільтрів, або використанням несправного обладнання [11]. Неправильне заземлення або його відсутність підвищують ризик.

- Негативний вплив мікроклімату [9] . Невідповідність температури, вологості та швидкості руху повітря санітарним нормам може призводити до перегріву або переохолодження організму, зниження імунітету та працездатності.

- Недостатнє або нераціональне освітлення. Неправильно організоване освітлення (занадто низька освітленість, наявність відблисків на екрані монітора, пульсація світла від неякісних ламп) призводить до підвищеної втоми очей та загальної втомлюваності.

- Підвищений рівень шуму. Джерелами шуму можуть бути системи охолодження ПК, офісна техніка, розмови колег. Постійний шум знижує концентрацію уваги та може викликати роздратування.

- Підвищений рівень електромагнітного випромінювання. Хоча сучасні монітори є значно безпечнішими за старі ЕПТ-моделі, ПК, Wi-Fi роутери та інше обладнання залишаються джерелами електромагнітних полів.

- Пожежна небезпека [10] . Може виникнути через коротке замикання в електромережі, перевантаження розеток або використання несправного обладнання. Скупчення паперу та пилу може сприяти швидкому поширенню вогню [10].

Психофізіологічні небезпечні фактори:

- Перенапруження зорового аналізатора [12] . Це ключовий ризик для розробника, пов'язаний із тривалою роботою з монітором, необхідністю розрізняти дрібні символи, працювати з кодом та графічними редакторами. Наслідками є головний біль, сухість та різь в очах, прогресуюче погіршення зору.

- Розумове та емоційне перенапруження. Розробка шейдерів вимагає високої концентрації, вирішення складних алгоритмічних задач та суворого дотримання термінів (дедлайнів). Це призводить до стресу, нервового виснаження та ризику професійного вигорання.

- Гіподинамія. Сидячий, малорухливий спосіб життя призводить до порушень у роботі опорно-рухового апарату (остеохондроз, сколіоз), серцево-судинної системи та до набору зайвої ваги.

- Монотонність праці. Виконання однотипних завдань протягом тривалого часу може знижувати інтерес до роботи та викликати апатію.

- Нераціональна організація робочого місця. Неергономічні меблі (стіл, крісло), неправильне розташування монітора та клавіатури призводять до вимушеної незручної пози, що викликає біль у спині, шиї та руках (тунельний синдром).

Загальні небезпеки:

- Військова загроза. В умовах воєнного стану в Україні існує постійна загроза ракетних обстрілів та атак БПЛА. Це створює пряму небезпеку для життя та здоров'я працівників, а також викликає сильний психологічний стрес.

- Біологічні небезпеки. Поширення інфекційних захворювань, що передаються повітряно-крапельним шляхом (грип, COVID-19), в умовах офісного простору.

Виявлення цих небезпек є першим кроком до побудови ефективної системи управління ризиками на робочому місці [13] .

4.3. Дослідження ризику реалізації небезпек та заходи щодо їх мінімізації

Після ідентифікації потенційних небезпек наступним кроком є оцінка ризику, тобто визначення ймовірності їх виникнення та тяжкості можливих наслідків. Оцінка ризику дозволяє пріоритезувати небезпеки та розробити адекватні заходи для їх контролю.

4.3.1 Теоретичні основи оцінки ризику

Оцінка ризику — це систематичний процес, що дозволяє підприємствам вживати заходів для захисту своїх працівників. Її основні задачі:

- ідентифікувати небезпеки та оцінити пов'язані з ними ризики;
- визначити, чи є існуючі заходи контролю достатніми;
- розробити план дій для усунення або зменшення ризиків до прийнятного рівня;
- пріоритезувати заходи на основі рівня ризику.

Для оцінки ризику часто використовується матричний метод, що полягає у визначенні рівня ризику на перетині двох показників: ймовірності (частоти) виникнення небезпечної події та тяжкості наслідків.

4.3.2 Проведення оцінки ризику для розробника

Проведемо оцінку ризику для двох характерних небезпек, виявлених у попередньому підрозділі, використовуючи матрицю 3x3.

1. Небезпека 1: Перенапруження зорового аналізатора.
2. Небезпека 2: Ураження електричним струмом.

Таблиця 4.1 - Матриця оцінки ризику

Ймовірність →Тяжкість ↓	Малоймовірно (1)	Ймовірно (2)	Дуже ймовірно (3)
Незначна шкода (1)	Низький (1)	Низький (2)	Середній (3)
Середня шкода (2)	Низький (2)	Середній (4)	Високий (6)
Велика шкода (3)	Середній (3)	Високий (6)	Неприйнятний (9)

Небезпека 1: Перенапруження зорового аналізатора

- Тяжкість наслідків: Середня шкода (2). Можливі наслідки включають головний біль, тимчасове погіршення зору, розвиток синдрому "сухого ока", що потребує лікування, але не становить прямої загрози життю.
- Ймовірність виникнення: Дуже ймовірно (3). Ця небезпека реалізується щодня, оскільки робота розробника безпосередньо і нерозривно пов'язана з використанням монітора протягом усього робочого дня.
- Рівень ризику: 2 * 3 = 6. Відповідно до матриці, це Високий ризик.

Небезпека 2: Ураження електричним струмом

- Тяжкість наслідків: Велика шкода (3). Наслідки можуть варіюватися від сильних опіків та травм до летального випадку.
- Ймовірність виникнення: Малоймовірно (1). За умови використання сертифікованого сучасного обладнання, регулярних перевірок та дотримання правил експлуатації, ймовірність такої події є низькою.
- Рівень ризику: 3 * 1 = 3. Відповідно до матриці, це Середній ризик.

4.3.3 Рекомендації щодо мінімізації ризиків

Для зниження високого ризику, пов'язаного з перенапруженням зорового аналізатора, необхідно впровадити комплексний підхід, що охоплює організаційні, технічні та медичні аспекти. На організаційному рівні ключову роль відіграє встановлення чітко регламентованих перерв у роботі. Згідно з найкращими практиками, після кожної години інтенсивної роботи за комп'ютером рекомендується робити короткі паузи тривалістю 5-10 хвилин, щоб надати очам можливість відпочити. Це доповнюється проведенням регулярних інструктажів для працівників, метою яких є підвищення їхньої обізнаності щодо важливості дотримання режимів праці та відпочинку.

Не менш значущими є технічні та ергономічні заходи. Роботодавець повинен забезпечити робочі місця якісними сучасними моніторами з високою роздільною здатністю, які підтримують технології захисту очей, такі як Flicker-Free (усунення мерехтіння) та Low Blue Light (зниження синього

випромінювання). Важливим є і правильне налаштування освітлення: загальне світло в приміщенні має бути розсіяним, а розташування світильників та робочого місця повинно виключати появу відблисків на екрані. Крім того, монітор слід розміщувати на оптимальній відстані 60-70 сантиметрів від очей, причому його центр має бути трохи нижче їхнього рівня. Завершує цей комплекс заходів медико-профілактичний напрям, що включає організацію періодичних медичних оглядів для своєчасного виявлення можливих проблем із зором та надання працівникам рекомендацій щодо виконання спеціальної гімнастики для очей під час робочих перерв.

Щодо контролю середнього ризику ураження електричним струмом, заходи зосереджені насамперед на технічній справності обладнання та організаційній дисципліні. З технічної точки зору, фундаментальною вимогою є використання виключно сертифікованого комп'ютерного та офісного обладнання, що відповідає стандартам безпеки. Це має доповнюватися регулярним технічним оглядом стану всієї електромережі, включно з розетками та кабелями живлення, для виявлення та усунення потенційних пошкоджень. Обов'язковою умовою є використання належного заземлення для всього обладнання, а також застосування мережевих фільтрів для захисту від раптових перепадів напруги.

Заходи щодо загальних небезпек:

- Пожежна безпека: Забезпечення приміщення вогнегасниками, наявність плану евакуації та проведення протипожежних інструктажів.
- Військова загроза: Наявність чіткого алгоритму дій під час повітряної тривоги, забезпечення доступу до укриття (власного або найближчого), можливість переходу на дистанційний режим роботи.

Впровадження цих заходів дозволить значно знизити рівень професійних ризиків та створити безпечне та продуктивне робоче середовище для розробника.

Висновки до розділу

У даному розділі було проведено комплексний аналіз питань охорони праці та безпеки в контексті професійної діяльності розробника шейдерів.

Було розглянуто законодавчу базу України, що регулює сферу охорони праці. Проаналізовано ключові нормативні документи, такі як Конституція України та Закон "Про охорону праці", та визначено розподіл обов'язків і відповідальності між основними суб'єктами трудових відносин: роботодавцем, працівником та державними контролюючими органами.

На основі аналізу умов праці було ідентифіковано та класифіковано потенційні небезпеки, що загрожують розробнику. До них належать фізичні фактори (електричний струм, мікроклімат), специфічні психофізіологічні ризики (перенапруження зору, стрес, гіподинамія), а також загальні загрози, зокрема пожежна та військова небезпека.

За допомогою матричного методу було проведено оцінку ризику для двох характерних небезпек - перенапруження зорового аналізатора та ураження електричним струмом. Встановлено, що ризик, пов'язаний із навантаженням на зір, є високим, тоді як ризик ураження струмом - середнім. На основі цієї оцінки було розроблено та запропоновано конкретний перелік організаційних, технічних та медико-профілактичних заходів. Рекомендовано впровадження регламентованих перерв, використання ергономічного обладнання, регулярні перевірки техніки та проведення інструктажів, а також дотримання правил безпеки під час надзвичайних ситуацій.

Реалізація запропонованих заходів дозволить створити безпечне, здорове та сприятливе для продуктивної інтелектуальної праці робоче середовище.

ВИСНОВКИ

У рамках даної кваліфікаційної роботи було проведено комплексне дослідження, що охопило теоретичні, практичні та організаційні аспекти розробки та оптимізації візуальних ефектів для 2D-ігор на платформі Unity. Мета роботи, яка полягала у створенні набору оптимізованих шейдерів для реалізації ключових візуальних ефектів, була повністю досягнута.

За результатами виконаної роботи були отримані наступні висновки та результати:

1. Проведено глибокий аналіз предметної області, який показав, що візуальні ефекти є невід'ємним інструментом сучасних 2D-ігор, що виконує функції створення атмосфери, забезпечення зворотного зв'язку та формування унікального візуального стилю. Встановлено, що шейдери є основним технічним засобом для реалізації цих ефектів, що дозволяє гнучко керувати графічним конвеєром.

2. Розроблено та реалізовано набір функціональних шейдерів, що включає ефекти пост-обробки («Світіння», «Корекція кольору», «Зернистість») та ефект спотворення для 2D-спрайтів. Для кожного шейдера було детально проаналізовано інформаційні потоки, розроблено математичні моделі та алгоритми, що лежать в основі їхньої роботи. Зокрема, для ефекту «Світіння» застосовано багатопрохідний алгоритм з розмиттям, для «Корекції кольору» — ефективний метод на основі LUT, а для «Зернистості» — процедурний підхід.

3. Практична реалізація проєкту здійснена у сучасному середовищі розробки Unity 2022.3.18f1 з використанням Universal Render Pipeline (URP). Це дозволило застосувати сучасні підходи до інтеграції ефектів через механізм Renderer Features. Було створено демонстраційну сцену, що наочно показує роботу кожного ефекту в релевантному ігровому контексті.

4. Важливою складовою роботи стало створення детального керівництва з використання розроблених шейдерів. У ньому описано процес інтеграції

ефектів у проєкт Unity та надано вичерпну інформацію щодо публічних параметрів кожного шейдера. Це робить розроблені рішення доступними для використання не лише програмістами, але й художниками та дизайнерами, значно підвищуючи їхню практичну цінність.

5. У рамках роботи було розглянуто питання охорони праці та безпеки для розробника. Проведено аналіз законодавчої бази, виявлено специфічні для IT-сфери професійні ризики, такі як психофізіологічне та зорове перенапруження. За допомогою матричного методу було проведено оцінку цих ризиків та розроблено конкретні рекомендації щодо їх мінімізації, що підкреслює комплексний підхід до процесу розробки.

Наукова та практична значущість роботи полягає у систематизації знань про розробку шейдерів для 2D-ігор та створенні готового до використання, оптимізованого набору графічних рішень. Результати роботи можуть бути використані як для покращення візуальної складової комерційних та незалежних ігрових проєктів, так і в освітніх цілях — як навчальний посібник з програмування комп'ютерної графіки в Unity.

Таким чином, дана кваліфікаційна робота є завершеним дослідженням, що успішно поєднує теоретичні основи, алгоритмічну розробку, практичну реалізацію та аналіз безпеки праці, демонструючи повний цикл створення сучасного програмного продукту в галузі комп'ютерної графіки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lengyel, E. (2019). Foundations of Game Engine Development, Volume 2: Rendering. Terathon Software LLC.

2. Akenine-Möller, T., Haines, E., & Hoffman, N. (2018). Real-Time Rendering (4th ed.). A K Peters/CRC Press.

3. Catlike Coding. Unity C# and Shader Tutorials. [Електронний ресурс]. Режим доступу: https://catlikecoding.com/unity/tutorials/

4. HLSL documentation - MSDN - Microsoft. [Електронний ресурс]. Режим доступу: https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl

5. Unity Technologies. Universal Render Pipeline (URP). [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@latest

6. Unity Technologies. Shader Graph. [Електронний ресурс]. Режим доступу: https://docs.unity3d.com/Packages/com.unity.shadergraph@latest

7. LearnOpenGL. Textures. [Електронний ресурс]. Режим доступу: https://learnopengl.com/Getting-started/Textures

8. Кодекс законів про працю України: Закон України від 10.12.1971 р. № 322-VIII. URL: https://zakon.rada.gov.ua/laws/show/322-08 (дата звернення: 23.06.2025).

9. Санітарні норми мікроклімату виробничих приміщень: ДСН 3.3.6.042-99: затв. Постановою Головного державного санітарного лікаря України від 01.12.1999 р. № 42.

10. Правила пожежної безпеки в Україні: затв. наказом МВС України від 30.12.2014 р. № 1417. URL: https://zakon.rada.gov.ua/laws/show/z0252-15 (дата звернення: 23.06.2025).

11. Правила безпечної експлуатації електроустановок споживачів: НПАОП 40.1-1.21-98: затв. наказом Держнаглядохоронпраці від 09.01.1998 р. № 4.

12. Катренко Л. А., Кіт Ю. В. Охорона праці в галузі інформаційних технологій: навчальний посібник. Львів: «Новий Світ – 2000», 2011. 368 с.

13. Жидецький Ю. Ц. Основи охорони праці: підручник. Львів: УАД, 2006. 336 с.

14. Офіційний сайт Державної служби України з питань праці. URL: https://dsp.gov.ua/ (дата звернення: 23.06.2025).

15. The Book of Shaders by Patricio Gonzalez Vivo & Jen Lowe. [Електронний ресурс]. Режим доступу: https://thebookofshaders.com/

16. Fernando, R., & Kilgard, M. J. (2003). The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics. Addison-Wesley Professional.

17. UPenn CIS 5610. GPU Programming and Architecture. [Електронний ресурс]. Режим доступу: https://www.cis.upenn.edu/~cis565/

18. Wikipedia. Bloom (shader effect). [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Bloom_(shader_effect)

19. Jimenez, J. (2014). Next-Gen Post-Processing in Call of Duty: Advanced Warfare. [Електронний ресурс]. GDC Vault. Режим доступу: https://www.gdcvault.com/play/1021771/Next-Gen-Post-Processing-in

20. Selan, J. (2005). "Chapter 24. Color Correction Using 3D Textures". У книзі: GPU Gems 2. Addison-Wesley Professional. [Електронний ресурс]. Режим доступу: https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-24-color-correction-using-3d-textures

21. ARM. (2020). Mali GPU Best Practices. [Електронний ресурс]. ARM Developer. Режим доступу: https://developer.arm.com/documentation/101897/latest/

22. Жерновий М.О., Баталов С.Д., Братерська Н. М. Кіберспорт у вищих навчальних закладах: розвиток та можливості // Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації - 2023 / Матеріали III Всеукраїнської науково-технічної конферен-ції молодих вчених, аспірантів і студентів, Одеса, 28-29 жовтня 2023 р. - Одеса, Видавництво ОНТУ, 2023 р. – с.47-49 – URL: https://ontu.edu.ua/download/konfi/2023/Abstracts-Computer-

games-and-multimedia-as-an-innovative-approach-to-electronic-communication-23.pdf. (дата звернення: 13.05.2025).

ДОДАТКИ

Додаток А

Реалізація шейдеру для корекції кольору

```
// Назва шейдера, яка буде відображатися в інспекторі матеріалів Unity.
Shader "Diploma/PostProcessing/ColorCorrectionLUT"
{
  // Властивості, доступні для налаштування в інспекторі.
  Properties
  {
    // Основна текстура сцени (заповнюється автоматично через C#).
    _MainTex ("Texture", 2D) = "white" {}
    // 3D-текстура (LUT), яку завантажує користувач.
    _LUT_Texture ("LUT Texture", 3D) = "" {}
    // Інтенсивність ефекту (слайдер від 0 до 1).
    _Intensity ("Intensity", Range(0.0, 1.0)) = 1.0
  }
  SubShader
  {
    // Налаштування для пост-ефекту: вимкнути запис у z-буфер,
    // вимкнути відсікання геометрії, оскільки рендериться повноекранний квад.
    Cull Off ZWrite Off ZTest Always

    Pass
    {
      HLSLPROGRAM
      #pragma vertex vert
      #pragma fragment frag

      #include "UnityCG.cginc" // Включаємо стандартні хелпери Unity

      // Структура даних, що передаються з CPU у вершинний шейдер.
      struct appdata
      {
        float4 vertex : POSITION;
        float2 uv : TEXCOORD0;
      };

      // Структура даних, що передаються з вершинного у фрагментний шейдер.
      struct v2f
      {
        float2 uv : TEXCOORD0;
        float4 vertex : SV_POSITION;
      };

      // Вершинний шейдер. Його задача - просто передати координати далі.
      v2f vert (appdata v)
      {
        v2f o;
        o.vertex = UnityObjectToClipPos(v.vertex);
        o.uv = v.uv;
        return o;
      }

      // Оголошуємо змінні, які ми визначили у блоці Properties.
      sampler2D _MainTex;
      sampler3D _LUT_Texture;
      float _Intensity;

      // Фрагментний шейдер - тут відбувається вся магія.
      fixed4 frag (v2f i) : SV_Target
      {
        // 1. Отримуємо оригінальний колір пікселя з текстури сцени.
```

```
            fixed4 originalColor = tex2D(_MainTex, i.uv);

            // 2. Використовуємо RGB-значення як 3D-координати для пошуку
            //    нового кольору в 3D-текстурі (LUT).
            fixed3 lutColor = tex3D(_LUT_Texture, originalColor.rgb).rgb;

            // 3. Змішуємо оригінальний колір та колір з LUT
            //    відповідно до параметра інтенсивності.
            //    lerp (linear interpolation) - функція лінійної інтерполяції.
            fixed3 finalColor = lerp(originalColor.rgb, lutColor, _Intensity);

            // 4. Повертаємо фінальний колір, зберігши оригінальну прозорість.
            return fixed4(finalColor, originalColor.a);
        }
        ENDHLSL
    }
  }
}
```

Додаток Б

Реалізація шейдеру для процедурної зарнистості

```
Shader "Diploma/PostProcessing/ProceduralFilmGrain"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
        _Intensity ("Intensity", Range(0.0, 1.0)) = 0.1
        _Scale ("Scale", Range(1.0, 200.0)) = 100.0
    }
    SubShader
    {
        Cull Off ZWrite Off ZTest Always

        Pass
        {
            HLSLPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            struct appdata
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };

            struct v2f
            {
                float2 uv : TEXCOORD0;
                float4 vertex : SV_POSITION;
            };

            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                o.uv = v.uv;
                return o;
            }

            sampler2D _MainTex;
            float _Intensity;
            float _Scale;

            // Проста хеш-функція для генерації псевдовипадкового значення.
            // Приймає 2D-координати і повертає число від 0 до 1.
            float random(float2 p)
            {
                return frac(sin(dot(p.xy, float2(12.9898, 78.233))) * 43758.5453);
            }

            fixed4 frag (v2f i) : SV_Target
            {
                // 1. Отримуємо оригінальний колір сцени.
                fixed4 originalColor = tex2D(_MainTex, i.uv);

                // 2. Готуємо координати для генерації шуму.
                //    - Множимо на _Scale, щоб контролювати розмір "зерна".
                //    - Додаємо час (_Time.y), щоб шум був динамічним (анімованим).
```

```
            float2 noiseUV = i.uv * _Scale;
            noiseUV.y += _Time.y;

            // 3. Генеруємо значення шуму (0..1) і зміщуємо його
            //    в діапазон [-0.5..0.5], щоб він міг як освітлювати, так і затемнювати піксель.
            float noise = random(noiseUV) - 0.5;

            // 4. Додаємо шум до оригінального кольору, множачи його на інтенсивність.
            fixed3 finalColor = originalColor.rgb + noise * _Intensity;

            return fixed4(finalColor, originalColor.a);
        }
        ENDHLSL
    }
  }
}
```

Додаток В

Апробація результатів роботи

Міністерство освіти і науки України
Одеський національний технологічний університет
Вінницький національний технічний університет
Інститут комп`ютерної інженерії, автоматизації, робототехніки та програмування ім.П.Н.Платонова

ПРОГРАМА

ІІІ ВСЕУКРАЇНСЬКОЇ НАУКОВО – ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ МОЛОДИХ ВЧЕНИХ, АСПІРАНТІВ ТА СТУДЕНТІВ

«КОМП'ЮТЕРНІ ІГРИ І МУЛЬТИМЕДІА ЯК ІННОВАЦІЙНИЙ ПІДХІД ДО КОМУНІКАЦІЇ - 2023»

28-29 вересня 2023 р.
ОДЕСА

УДК 004.42+37.06

КІБЕРСПОРТ У ВИЩИХ НАВЧАЛЬНИХ ЗАКЛАДАХ: РОЗВИТОК ТА МОЖЛИВОСТІ

ЖЕРНОВИЙ М.О., БАТАЛОВ С.Д., БРАТЕРСЬКА Н.М.
(mykyta.zhernovyi@kname.edu.ua, serhii.batalov@kname.edu.ua, nataliia.braterska@kname.edu.ua)
Харківський національний університет міського господарства імені О. М. Бекетова

Навчання в XXI столітті є невід'ємною частиною шляху становлення пристойної людини, але конкретно система навчання бере початок ще за часів "Київської Русі". Вона не була переробена, а тільки доповнювалася, тому в наші дні пересічний навчальний заклад має такі ж методи навчання, як і тисячоліття тому. Але "середня температура по лікарні" не відображає усього спектру можливостей нашого часу, бо існують винятки, тенденції та новаторські ідеї, причиною яких є загальний розвиток цивілізації. Одним з багатьох новаторств є кіберспорт та його застосування в "закостенілій" системі освіти.

За весь час розвитку геймингу було створено багато ігор, деякі з них мають у своєму корені систему командної гри, в якій 2 або більше команд гравців змагаються один з одним. Саме такі ігри можуть входити до кіберспортивних дисциплін. Для прикладу, найпопулярнішими дисциплінами є:

- League of Legends – гра жанру MOBA(Multiplayer Online Battle Arena), в якій ціль кожної з команд полягає в контролю над картою та базою суперника.
- Dota 2 – така ж сама MOBA, як і League of Legends, ціль гри домінування на карті та знищення бази суперника
- Counter-Strike: Global Offensive – командна гра, в якій дві команди виконують ролі терористів та анти-терористів. Є шутером, в якому задача полягає в знищенні команди суперника, встановленні бомби або порятунку заручників.

Для участі у змаганнях з кіберспорту гравцю необхідно досконало знати особливості гри та вміти працювати в команді.

Кіберспорт як вид змагань серед гравців бере свій початок це з 1970-х років. Тоді спортивна зацікавленість була у отриманні та утриманні рекордів, але це не був саме кіберспорт, в сьогоденному розумінні. Те саме поняття «кіберспорт» з'явилося в США ще 1997 року внаслідок створення CPL (The Cyberathlete Professional League) – професійної Ліги з кіберспорту, призначенням якої була організація перших турнірів з Quake.

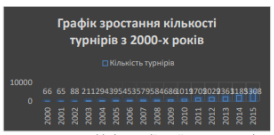


Рисунок 1 – Графік створено на основі інформації з сайту esportsearnings.com

Це дало потужний поштовх у потужний поштовх у розвитку геймінгу та кіберспорту в цілому. Трохи пізніше Counter Strike стане ключовою дисципліною кіберспортивного геймінгу, а починаючи з 2000-х почнуть з'являтися професійні ліги та турніри.

На даний момент кіберспорт є великою індустрією та охоплює велику кількість ігрових дисциплін,мільйони глядачів. Команди, які приймають участь у змаганнях мають с розвиток технологій та інтернет дав змогу проводити турніри, за якими спостерігають спонсорські контракти з великими компаніями та визнання у геймерському просторі.

Набуваючи все більшу популярність у світі кіберспорт почав привертати увагу освітян-практиків та дослідників. Тому кількість освітніх закладів, які впроваджують в кіберспорт як дисципліну зростає з кожним роком. Наприклад, у 2019 році кількість середніх шкіл, які беруть