Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота бакалавра

з теми: **«Дослідження сучасних технологій створення реалістичних зображень в інтерактивних середовищах»**

Виконав: здобувач вищої освіти групи KNms1-B22
спеціальності 122 Комп’ютерні науки
Гріневич Антон Сергійович
(прізвище, ім’я та по батькові здобувача вищої освіти)

Керівник: Смалько Олена Аркадіївна
(прізвище, ім’я та по батькові, науковий ступінь, вчене звання керівника)

кандидат педагогічних наук,
доцент кафедри комп’ютерних наук

Рецензент: Оптасюк Сергій Васильович
(прізвище, ім’я та по батькові, науковий ступінь, вчене звання рецензента)

кандидат фізико-математичних наук,
доцент, завідувач кафедри фізики

м. Кам’янець-Подільський – 2025 р.

АНОТАЦІЯ

У кваліфікаційній роботі проведено дослідження сучасних технологій реалізації фізично коректного рендерингу в інтерактивних середовищах

Об’єктом дослідження є процес створення реалістичних зображень в інтерактивних тривимірних середовищах, предметом дослідження виступають методи підвищення реалістичності і швидкодії алгоритмів фізично коректної візуалізації.

Мета роботи — аналіз сучасних графічних алгоритмів та створення навчального-демонстраційного застосунку, який реалізує алгоритм трасування променів.

У межах дослідження реалізовано демонстраційний застосунок, який дозволяє виконувати трасування променів у реальному часі без потреби у використанні графічних адаптерів з апаратним прискоренням трасування променів. Розроблену систему протестовано на платформах Windows, Linux та MacOS з використанням різних графічних API (DirectX 12, Vulkan, Metal). Отримані результати підтверджують працездатність і стабільність реалізації, а також придатність створеного інструменту для навчальних та дослідницьких цілей у галузі комп’ютерної графіки.

Ключові слова: комп’ютерна графіка, тривимірна графіка, глобальне освітлення, трасування променів, фізично коректний рендеринг, інтерактивне середовище, кросплатформність.

ABSTRACT

The qualification thesis explores modern technologies of physically-based rendering solutions in interactive environments.

The object of the study is a process creation of realistic images in interactive 3D environments, the subjects of the study are methods of believability approximation and optimization for physically-based visualization algorithms.

The objective — analyze modern graphical algorithms and develop an cross-platform application for demonstration and learning purposes, which implements a Ray Tracing algorithm.

The result of the work is a developed demo app, which allows to perform a real-time Ray Tracing without hardware-accelerated RT GPU. Developed system is tested on Windows, Linux and MacOS platforms with DirectX 12, Vulkan and Metal API accordingly. Results are applicable and stable, the instrument is suitable to use for studying and research purposes in Computer Graphics field.

Keywords: computer graphics, three-dimensional graphics, global illumination, ray tracing, physically-based rendering, interactive environment, crossplatform.

ЗМІСТ

ВСТУП

У сучасному світі ком’ютерна графіка знайшла застосування у багатьох сферах, від візуалізації даних для наукових досліджень до моделювання фізичних явищ. У медіа, зокрема у відеоіграх, анімації і кінематографі, комп’ютерна графіка у сучасних реаліях відіграє ключову роль. Не дивлячись на те що моделі освітлення на основі яких будуються усі сучасні технології реалізації комп’ютерної графіки існують уже пів століття, графічні акселератори які здатні обробляти алгоритми побудови фізично коректної візуалізації у реальному часі із інтерактивною швидкістю вперше були створені менше десяти років тому. Основні напрямки розвитку графіки у сучасних комп’ютерних науках — більш фізично коректна відповідність наявним моделям освітлення, що в загальному сприймається як наближення реалістичності, а також оптимізація швидкодії роботи алгоритмів утворення реалістичних зображень.

Актуальність теми. Вивчення і удосконалення сучасних технологій створення реалістичних зображень у інтерактивних середовищах є найголовнішою передумовою для подальшого розвитку усієї сфери комп’ютерної графіки. Сучасні технології, такі як трасування променів, глобальне освітлення, новітні методи згладжування відкривають нові можливості для покращення візуальної якості отриманої картинки, даючи змогу створювати все більш детальний і реалістичний користувацький досвід. Водночас розвиток апаратних технологій, основним призначенням яких є обробка тривимірної графіки, штовхає вперед і інші напрямки розвитку комп’ютерних наук, наприклад машинне навчання, обчислювальна фізика та хімія, біоінформатика і медицина.

Об’єктом дослідження є процес створення реалістичних зображень в інтерактивних тривимірних середовищах.

Предметом дослідження виступають методи підвищення реалістичності і швидкодії алгоритмів фізично коректної візуалізації.

Мета дослідження — аналіз сучасних технологій створення ефективних методів реалізації трасування променів у реальному часі та розробка навчально-демонстраційного застосунку, що реалізує трасування променів і дозволяє вивчати принципи фізично коректного рендерингу без потреби у високопродуктивній апаратній базі.

Для досягнення мети потрібно вирішити наступні **завдання**:

- Дослідити і проаналізувати сучасні методи реалізації фізично коректної візуалізації;
- Сформувати вибірку технічних рішень, придатних для реалізації у вигляді кросплатформного застосунку;
- Розробити та реалізувати інструмент, який демонструє роботу трасування променів у реальному часі;
- Провести кросплатформне тестове дослідження якості візуалізації та використання апаратних ресурсів на різних платформах.

Методи дослідження. Методологічна основа дослідження сучасних технологій створення реалістичних зображень ґрунтується на комплексному підході, що поєднує аналіз даних про застосування алгоритмів у практичних умовах. Це дозволяє оцінити, як саме використовуються наявні технології та визначити напрями їх удосконалення. Дослідження включає вивчення реальних прикладів застосування технологій генерації реалістичних зображень, що дає змогу проаналізувати практичний досвід і результати впровадження відповідних рішень. Крім того, у межах роботи приділяється увага розробці й апробації нових методів та інструментів, спрямованих на вирішення проблем у області створення реалістичних зображень.

Практичним значенням дослідження є розробка мультиплатформного інструменту, який реалізовує алгоритм трасування променів і демонструє механізм роботи алгоритму у реальному часі в інтерактивному середовищі. Інструмент може виступати в якості програмної бази для подальших досліджень

і в якості освітнього матеріалу для ентузіастів без доступу до дороговартісного обладнання.

Структура роботи. Дипломна робота складається зі вступу, трьох основних частин, висновків, списку використаних джерел та розділу з додатками.

У першому розділі проводиться огляд основоположних технологій сфери фізично коректного рендерингу, а також сучасних можливостей у цій сфері, останні технологічні досягнення — нові покоління відео адаптерів і графічні API.

В другому розділі проводиться аналіз новітніх алгоритмів покращення технологій побудови реалістичних зображень — динамічного глобального освітлення, тимчасового згладжування та знешумлення об’ємного трасування шляху.

Третій розділ описує розробку мультиплатформного навчально-демонстраційного застосунку з можливістю інтерактивної взаємодії, що реалізує трасування променів у реальному часі, включаючи впровадження сучасної графічної API у алгоритм рейтрейсингу, опис архітектури проекту, а також результати кросплатформних тестів.

Робота містить два рисунки і шість додатків.

Список використаних джерел складається з 25 пунктів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ТА ФУНДАМЕНТАЛЬНІ ТЕХНОЛОГІЇ СТВОРЕННЯ РЕАЛІСТИЧНИХ ЗОБРАЖЕНЬ

1.1. Наукове підґрунтя, базові поняття і концепції сучасних методів фізично коректного рендерингу

Найпростіші алгоритми візуалізації поверхні базуються на законі Ламберта, згідно якому яскравість ідеально розсіючої поверхні однакова по всіх напрямках. Інтенсивність відображеного світла пропорційна скалярному добутку нормалі поверхні та напрямку джерела освітлення. На основі такої моделі можна проводити симуляцію світла і його розсіювання. Це дозволяє реалістично візуалізовувати темні матові поверхні. Буі Тонг Фонг запропонував ускладнену модель розрахунку інтенсивності світла, яка припускає що джерела освітлення знаходяться далеко від об’єктів сцени, а отже не передбачає таких у самій сцені. Така модель також не ураховує глобального освітлення. Математичне зображення моделі Фонга надано у Додатку А.

У 1980 році, Тернер Віттед опублікував нову модель (див. Додаток Б) освітлення на базі моделі Фонга, опублікувавши знакову статтю “An Improved Illumination Model for Shaded Display”. У цій роботі він вперше запропонував повноцінну схему рейтрейсингу для симуляції оптичних явищ: промені випускаються з камери, перевіряють перетини з об’єктами сцени, і у випадку взаємодії з поверхнею породжують нові промені для моделювання дзеркальних відображень, заломлень через прозорі матеріали та тіней [23].

У 1984 році Роб Кук, Том Портер і Лорн Карпентер у роботі "Distributed Ray Tracing" розширили підхід Віттеда, запропонувавши стохастичне трасування променів. Вони замінили детерміноване проходження єдиного променя множиною променів, випадково вибраних за певним розподілом, що дозволило моделювати ефекти глибини різкості, м’яких тіней і розмитих відбиттів.

Цей підхід став першим кроком до використання методу Монте-Карло у трасуванні, де інтеграли освітлення обчислюються за допомогою випадкових вибірок [9].

У 1986 році Джеймс Каджія у своїй видатній роботі “The rendering equation” сформулював рівняння рендерингу, яке стало універсальним описом процесу формування зображення (див. Додаток В). У цій же роботі Каджія представив метод path tracing — трасування випадкових ланцюгів променів від камери через сцену, який дозволяє відтворювати ефекти глобального освітлення: кольорові відбиття, каустики, непряме освітлення. Трасування шляхів стало концептуальною основою для всіх подальших фізично коректних методів трасування променів, хоча через свою обчислювальну складність тривалий час залишався непридатним для застосування у реальному часі [14].

1.2. Сучасні графічні акселератори, трасування променів у реальному часі

Напротязі десятиліть растеризація залишалась основним способом рендерингу в інтерактивних програмах, тоді як трасування променів використовувалось в основному для візуалізації у кіно та анімації, а також інших сферах де не вимагалося обчислювати алгоритм трасування у реальному часі. В сучасності трасування променів у реальному часі — це переважно гібридний підхід, де основний рендеринг сцени виконується методом растеризації, а трасування променів використовується лише для обчислення вибіркових ефектів: відбиттів, заломлень, м’яких тіней, каустик або непрямого освітлення. Цей компроміс дозволяє отримувати фотореалістичну графіку в інтерактивних застосунках за прийнятної продуктивності.

У 2018 році корпорація Nvidia представила першу лінійку відеокарт на архітектурі Turing яка включала Quatro RTX — графічні акселератори робочого сегменту, та GeForce RTX — чіпи для медіа індустрії, зокрема для відеоігр.

Ключова відмінність архітектури Turing від попередніх карт Nvidia — наявність RT ядер у графічному процесорі, що дозволяє значно прискорити виконання базових операцій трасування, таких як побудова і проходження BVH-структур та перевірка перетинів променів із трикутниками сцени. Крім того, у Turing було вдосконалено тензорні ядра для обчислень, пов’язаних із машинним навчанням, зокрема для знешумлення зображень, що виникають під час трасування променів.

У 2020 році з'явилася архітектура Ampere, що включала лінійку адаптерів RTX 3000, яка суттєво підвищила продуктивність трасування променів у реальному часі. В Ampere реалізовано друге покоління RT-ядер, які ефективніше виконують паралельні обчислення променів, тіней і віддзеркалень, а також третє покоління тензорних ядер, оптимізованих під нові типи матричних обчислень — TF32 та sparse matrix operations. У поєднанні з новими можливостями API (наприклад, DirectX 12 Ultimate та Vulkan RT), це дало змогу досягти фотореалістичного рендерингу навіть на споживчих відеокартах у режимі реального часу, особливо у зв’язці з технологією DLSS (Deep Learning Super Sampling).

Подальший розвиток було закладено в архітектурі Ada Lovelace, що лягла в основу серії RTX 4000 (починаючи з 2022 року). Ada включає третє покоління RT-ядер та четверте покоління тензорних ядер, а також принципово нову технологію Shader Execution Reordering (SER) — динамічну перебудову виконання шейдерів для кращої паралелізації трасування променів. Також з’явилася підтримка DLSS 3, де генерація міжкадрових зображень (frame generation) базується на штучному інтелекті, що значно підвищує кадрову частоту навіть у важких сценах з повним трасуванням

На противагу Nvidia, AMD у 2020 році запропонували архітектуру RDNA2 в обличчі відеокарт серії Radeon RX6000, Це був перший випадок, коли графічні процесори AMD отримали апаратну підтримку трасування променів, через впровадження Ray Accelerators — спеціалізованих блоків для прискорення

обчислень, пов’язаних із перетинами променів. І хоча рішення AMD загалом демонструють нижчу продуктивність у завданнях із трасування порівняно з NVIDIA, зокрема через відсутність тензорних ядер і фреймворків типу DLSS, AMD все ж показують стабільно вищу продуктивність у растеризації, а також мають кращі показники споживання енергії.

Архітектура RDNA 3, представлена у 2022 році, принесла покращення в області рейтрейсингу, зокрема завдяки підвищенню пропускної здатності Ray Accelerators та загальній оптимізації графічного конвеєра. Вперше була реалізована модульна чиплетна архітектура (розділення обчислювальних і кеш-блоків), що забезпечує кращу масштабованість. Однак, навіть із RDNA 3, рішення AMD залишаються менш ефективними в задачах трасування променів у порівнянні з NVIDIA, особливо в умовах складного освітлення та об’ємного денойзингу, де особливо помітна відсутність апаратної підтримки AI-модулів. Це дещо компенсується наявністю у AMD власної технології яку ставлять у противагу DLSS - FidelityFX Super Resolution. FSR — це алгоритм покращення якості зображення і продуктивності програми за рахунок використання апскейлінгу. Остання версія, FSR 3.0 також впроваджує функцію генерації кадрів, тож загалом AMD пропонує весь спектр технопослуг, але не будують свої технології навколо ШІ. Також FSR — це відкрита технологія, що дозволяє розробникам інтегрувати її у рушії своїх відеоігр, тож все більше ігр на ринку підтримують цю технологію. Це також підсилюється тим фактом що FSR, на відміну від DLSS не вимагає наявності відео адаптеру конкретної моделі чи лінійки, а працює на усіх пристроях цього сегменту.

Також у контексті трасування променів у інтерактивних застосунках варто згадати WebGPU — найсучасніший графічний API, що є наступником WebGL в області веб-графіки. Головна особливість WebGPU в тому що він сумісний з усіма сучасними графічними API, як от Vulcan, Direct3D та Metal, а отже має низькорівневий доступ до ресурсів графічного адаптера. Це в свою чергу

дозволяє WebGPU мати підтримку обчислювальних шейдерів, що робить цю технологію придатною для реалізації симуляцій процесів фізики, машинного навчання, а також, в теорії використання трасування променів із веб браузера.

WebGPU має свою власну мову шейдерів — WGSL, яка була створена з урахуванням специфіки середовищ браузерів. WGSL характеризується як більш безпечна за GLSL, в тому числі за рахунок суворого синтаксису, який багато в чому бере приклад з RUST.

У використанні із іншими інструментами WebGPU цілком може підійти для розробки застосунків категорії AAA, але при цьому уникаючи типових проблем на які страждають такі релізи. До прикладу ігровий рушій Unreal Engine, навіть в останній редакцій дуже чутливий до налаштувань і оптимізації, а витоки пам’яті для нього є цілком розповсюдженою помилкою. WGSL майже повність усуває цю проблему на рівні дизайну, оскільки суворий синтаксис дозволяє відловити більшість типових помилок пов’язаних із пам’яттю в процесі аналізу коду ще до компіляції. Це також дозволяє уникати деяких помилок які можуть виникнути лише у процесі рантайму.

РОЗДІЛ 2
СУЧАСНІ ТЕХНОЛОГІЇ СТВОРЕННЯ РЕАЛІСТИЧНИХ ЗОБРАЖЕНЬ

2.1. Динамічне розсіяне глобальне освітлення

На відміну від офлайн рендерингу, методи глобального освітлення для інтерактивних застосунків, фундаментально залежні від даних освітленості, які можуть бути зчитані з просторово-кутових структур даних які зазвичай або прораховані наперед, або обмежені повільним оновленням із статичної геометрії для динамічного освітлення. Прикладами послуговують карти світла, проби радіансу та іррадіації або вокселізована репрезентація сцени чи інформації про освітлення. Кожен із методів вимушений балансувати між компактністю, гнучкістю при рантаймі, фізичною коректністю та використанням апаратних ресурсів [20].

Динамічне розсіяне глобальне освітлення (англ. Dynamic Diffuse Global Illumination, DDGI) — це сучасний підхід до розрахунку непрямого освітлення в реальному часі, що поєднує переваги традиційних технік на основі проб освітленості з можливостями апаратного трасування променів. Основна мета методу — забезпечити обчислення глобального освітлення із високою якістю та стабільністю при мінімальному впливі на продуктивність.

Серцем алгоритму є динамічні проби іррадіації — спеціальні точки у 3D-просторі, розміщені у вигляді регулярної решітки, які постійно оновлюються на кожному кадрі. Кожна проба зберігає інформацію про дифузне освітлення у своїй околиці, зокрема значення кольору іррадіації, орієнтацію поверхонь та глибину сцени. Ці дані дозволяють отримати узагальнену картину розподілу світла, яку потім можна використовувати під час шейдингу пікселів на екрані. Математичне зображення такого алгоритму надано у Додатку Д.

Оновлення проб відбувається в кілька етапів. На кожному кадрі для певної підмножини проб запускається процес генерації променів, які поширюються у

довільних напрямках і взаємодіють зі сценами. Це забезпечує отримання нової інформації про освітлення на основі поточної геометрії та джерел світла. Отримані дані агрегуються і записуються у текстури, прив’язані до проб, де зберігаються кольори іррадіації та середні глибини для обраного просторового об’єму (див. Додаток Е).

Для того щоб уникнути артефактів, пов’язаних із шумом або дискретністю проб, дані проходять через етап згладжування за допомогою фільтрів на основі моментів. Цей крок дозволяє зберегти плавність освітлення та уникнути стрибків значень при зміні геометрії або освітлення. При рендерингу сцени, під час обчислення кольору пікселя, система використовує дані проб, знаходячи найближчі точки і виконуючи три-лінійну інтерполяцію. Це дозволяє отримати значення дифузного непрямого освітлення у будь-якій точці сцени. Отримані значення комбінуються з прямим освітленням та результатами відбиттів, утворюючи повноцінну картину глобального освітлення [15].

У розширення підходу DDGI було розроблено алгоритм ресемплінгу динамічного розсіяного глобального освітлення (англ. Dynamic Diffuse Global Illumination Resampling), що дозволяє об'єднувати дані непрямого освітлення, отримані за допомогою DDGI, із прямим освітленням, а також використовувати єдиний механізм вибору кандидатів освітлення для обох компонентів. Основна мета полягає в тому, щоб сприймати всю поверхню сцени як потенційне джерело світла, незалежно від того, чи є вона емісійною поверхнею або отримує світло опосередковано через багаторазові відбиття.

Ключовими перевагами підходу є здатність до адаптації під динамічні зміни сцени: зміни освітлення, рухомі об’єкти, зміни геометрії. Оскільки оновлення проб виконується поступово, а дані зберігаються у текстурах GPU, метод забезпечує стійкий FPS навіть у складних сценах, значно перевершуючи за продуктивністю повноцінне трасування променів при збереженні високої якості результату.

Процес ресемплінгу складається з кількох ключових етапів. На першому кроці відбувається формування множини кандидатів для освітлення. Це множина, яка включає точки на емісійних поверхнях сцени, вибрані випадковим чином, а також додаткові кандидати, отримані шляхом генерації вторинних променів від поверхонь, вибраних із використанням BSDF-розподілу. Таким чином, кандидати охоплюють як пряме, так і непряме освітлення, забезпечуючи комплексне уявлення про потоки енергії у сцені. Після формування множини кандидатів застосовується метод резервуарної вибірки (англ. Reservoir Sampling) для обрання фінального кандидата. Ймовірність вибору кожного кандидата пропорційна сумі двох компонентів: випромінюваного світла та апроксимованої освітленості, отриманої з DDGI-об'єму. Це означає, що точки, які самі випромінюють світло або отримують значний внесок від непрямого освітлення, мають вищі шанси бути обраними для подальшого трасування променів.

Базовий підхід DDGI обчислює лише дифузне непряме освітлення, і якщо використовувати його як єдине джерело освітлення, фінальне зображення виходить надто темним. Це відбувається тому, що DDGI не враховує дифузне пряме освітлення, яке походить безпосередньо від джерел світла, а також компоненти відбиття від глянцевих поверхонь. Для усунення цих обмежень необхідно розширити апроксимацію DDGI, додавши відсутні зразки. Щоб отримати повніший результат наявну апроксимацію DDGI доповнюють компонентами дифузного прямого освітлення та відбиттів від глянцевих поверхонь. Важливо, що цей підхід допомагає зменшити артефакти, такі як витоки світла та відсутність контактних тіней у складних сценах [16].

2.2. Удосконалення тимчасового згладжування адаптивним трасуванням променів

У сучасних системах рендерингу реального часу важливою складовою візуальної якості є згладжування артефактів, що виникають під час відображення

динамічних сцен із використанням рендерингу з обмеженою кількістю семплів на піксель. Одним із найпоширеніших рішень є темпоральне згладжування (англ. Temporal Anti-Aliasing, TAA), що використовує дані попередніх кадрів для зменшення шуму та артефактів на поточному кадрі. Однак TAA має ряд суттєвих недоліків: поява розмиття та гоустинг-ефектів, некоректна реконструкція деталей при зміні кадру, артефакти при відкритті нових областей сцени або при швидкому русі камери. Ці обмеження суттєво знижують придатність TAA для відображення складних сцен із високими вимогами до точності рендерингу [17].

З появою апаратної підтримки трасування променів у графічних API, таких як DirectX Raytracing, та графічних акселераторів нового покоління, виникла можливість застосовувати фізично точні моделі освітлення та відображення поверхонь у режимі реального часу. Це відкриває нові підходи для усунення обмежень класичних методів згладжування, оскільки трасування променів може забезпечити коректне формування зображення навіть у складних умовах сцени.

Адаптивне темпоральне згладжування (англ. Adaptive Temporal Antialiasing, ATAA) — це гібридний підхід, що поєднує традиційний метод TAA з апаратним трасуванням променів для підвищення якості фінального зображення. Основна мета цього підходу — використовувати трасування променів вибірково, лише там, де класичне TAA не здатне впоратися із завданням реконструкції кольору пікселя. Такий підхід дозволяє зберегти продуктивність рендерингу, одночасно підвищуючи якість зображення, зменшуючи шум, зберігаючи чіткість деталей та усуваючи гоустинг-ефекти.

Основою ATAA є сегментація сцени на області, які потребують різних типів обробки. Для цього формується маска, яка для кожного пікселя вказує, чи слід застосовувати традиційне TAA, класичні методи згладжування такі як FXAA, чи трасування променів. Маска визначається за допомогою аналізу руху об'єктів, швидкості камери, векторів руху пікселів та наявності складних освітлювальних ефектів. Пікселі, де прогнозована якість TAA є недостатньою — наприклад, у

випадку появи нових об'єктів у кадрі або швидкої зміни освітлення, — позначаються для обробки трасуванням променів. Водночас стабільні області сцени обробляються за допомогою TAA або FXAA для економії ресурсів.

Для тестів алгоритм ATAA було інтегровано у рендер-пайплайн рушія Unreal Engine, використовуючи API DirectX Raytracing для виконання обчислень трасування променів. Рендер-процес побудований таким чином, що на кожному кадрі спочатку виконується формування сегментаційної маски, далі застосовується TAA для більшості пікселів, а для областей, визначених маскою, виконується трасування променів. Результати цих двох підходів об'єднуються у фінальне зображення.

Обмеження алгоритму — його продуктивність та якість безпосередньо залежать від точності сегментації сцени, яка визначає області для трасування променів. Помилки в масці можуть призвести до недостатньої обробки дрібних деталей або надмірного навантаження на GPU у складних сценах. Також метод хоч і придатний, все ж має обмеження щодо інтеграції в реальні рушії, оскільки потребує доступу до буферів руху, нормалей, освітлення та геометрії на рівні низькорівневого рендер-пайплайну. Для подальшого покращення алгоритму перспективними напрямами є автоматизація сегментації за допомогою машинного навчання, оптимізація трасування через денситі-апсемплінг, а також розробка методів динамічного регулювання щільності променів залежно від складності сцени.

2.3. Знешумлення об’ємного трасування шляху для прямого об’ємного рендерингу у реальному часі

Прямий Об’ємний Рендеринг (англ. Direct Volume Rendering, DRV) з використанням Об’ємного Трасування Шляху (англ. Volumetric Path Tracing, VPT) — це новий тренд серед алгоритмів об’ємного рендерингу, що використовує більш реалістичні моделі освітлення для фотореалістичних візуалізацій. Цей

тренд у західних практиках наукової візуалізації набув популярності під назвою Кінематографічний Рендеринг.

Моделі глобального освітлення, що застосовуються у прямому об’ємному рендерингу, базуються на рівнянні переносу випромінювання, яке є фундаментальним рівнянням для опису перенесення світла у медіа. Каджия та Вон Херзен запропонували наближене рішення цього рівняння для використання у комп’ютерній графіці. Трасування шляхів за допомогою рівняння Монте-Карло забезпечує неупереджене вирішення цього рівняння завдяки єдиній теоретичній основі, яка гарантує збіжність до точного результату. Об’ємне трасування променів формує зображення прямого об’ємного рендерингу шляхом прогресивного усереднення великої кількості вибірок освітленості, обчислених для випадково згенерованих світлових шляхів.

Основним недоліком цього підходу є те, що для отримання високоякісних зображень прямого об’ємного рендерингу потрібен значний час, а для досягнення частоти кадрів, близької до інтерактивної - надзвичайно потужне апаратне забезпечення. В інших випадках результати містять значний рівень шуму, який виникає через дисперсію, характерну для інтеграції вибірок методом Монте-Карло. М. Шіх та дослідники з Аргонської національної лабораторії запропонували паралелізований, розподілений за даними та прискорений на GPU алгоритм для об’ємного рендерингу зі складним освітленням, який реалізував м’які тіні та працював на кластері до 128 GPU [21]. Прогресивний об’ємний рендеринг методом Монте-Карло, наприклад, рендеринг експозиції або метод прогресивного обчислення тону світла, поступово уточнює кольори пікселів методом трасування шляху і може забезпечити майже безшумні зображення лише через кілька секунд. Проте навіть ці методи залишають помітний шум (див. Додаток Є), що проявляється як тимчасове мерехтіння під час зміни положення камери, модифікацій джерел світла або функції розподілу, оскільки

інтерактивний режим змушує рендеринг обмежуватись малою кількістю зразків на піксель.

Таким чином, завдання зниження шуму та забезпечення тимчасової стабільності послідовностей зображень прямого об’ємного рендерингу залишається відкритою науковою проблемою. У комп’ютерній графіці дослідження зниження шуму для об’ємного трасування променів переважно зосереджені на офлайн рендерингу. Натомість зниження шуму в реальному часі здебільшого було орієнтоване на сцени з моделями поверхонь і досі не було адаптоване для об’ємного рендерингу в реальному часі у медіа з неоднорідною структурою [11].

Інтеграл об’ємного рендерингу (див. Додаток Ж) описує перенесення світла у середовищах, що поглинають і розсіюють світло, та є ключовим рівнянням для розрахунку глобального освітлення у медіа. Його рішення у складних середовищах базується на чисельних методах, зокрема на інтеграції методом Монте-Карло. Прогресивне об’ємне трасування променів дозволяє обчислювати наближення інтегралу шляхом генерації великої кількості світлових шляхів, визначених випадковим чином.

Використання об’ємного трасування шляху є привабливим вибором для підтримки фізично коректного глобального освітлення з використанням каркасу прямого об’ємного рендерингу, оскільки VPT є загальним і покриває широкий спектр світлових ефектів. Водночас стохастична природа цього підходу призводить до значного шуму в отриманих зображеннях, особливо беручи до уваги обмежену кількості зразків на піксель. Такий шум перетворюється на тимчасове мерехтіння під час взаємодії користувача зі сценою прямого об’ємного рендерингу — наприклад, при зміні положення камери, модифікаціях джерел світла або зміні функції розподілу.

Дослідження спрямоване на фільтрацію цього шуму за допомогою методу зниження шуму в реальному часі, який використовує просторову та часову

узгодженість кольорів пікселів для покращення числової точності та візуальної якості результатів прямого об’ємного рендерингу.

У реалізації даного методу уникають використання, вже класичних, допоміжних буферів, натомість орієнтуючись на накопичення стабільних ознак за одиницю часу. Це дозволяє використовувати інформацію з попередніх кадрів для посилення просторово-часової узгодженості та забезпечення більш ефективного зниження шуму без залежності від зашумлених допоміжних даних. Таким чином дана технологія пропонує метод зниження шуму у просторі зображення для результатів прямого об’ємного рендерингу, отриманого шляхом об’ємного трасування променів. Основний підхід ґрунтується на тому, що значення яскравості кожного пікселя розглядається як функція від певного набору ознак. Цими ознаками можуть бути, наприклад, колір пікселя з попереднього кадру або інші стабільні ознаки, накопичені за одиницю часу. Для кожного пікселя будується лінійна модель, що дозволяє передбачити його колір на основі цих ознак.

Коефіцієнти лінійної моделі обчислюються адаптивно за допомогою методу рекурсивних найменших квадратів. Цей метод дозволяє постійно оновлювати модель, враховуючи різницю між прогнозованим і фактичним значенням кольору пікселя, що робить процес обчислень придатним для роботи в реальному часі.

У запропонованій системі використовується модифікація цього методу, яка передбачає обчислення ваги для кожного пікселя. Вага визначається на основі різниці між поточним кольором пікселя та накопиченою у часі ознакою. Чим більша ця різниця, тим меншою є вага вибірки. Таким чином, пікселі з високим рівнем шуму отримують менший вплив на результати фільтрації. Це дозволяє ефективно пригнічувати вплив стохастичних вибірок об’ємного трасування променів на кінцеве зображення.

Для покращення результатів також застосовується просторове згладжування. Воно виконується у вигляді двосторонньої фільтрації в локальному вікні пікселів. Такий підхід дозволяє зменшити залишковий шум без надмірного розмиття важливих деталей сцени. Поєднання просторової обробки та адаптивного вагового денойзингу забезпечує отримання зображень високої якості навіть при низьких значеннях кількості зразків на піксель [12].

РОЗДІЛ 3
РОЗРОБКА МУЛЬТИПЛАТФОРМНОГО РІШЕННЯ ДЛЯ ТРАСУВАННЯ ПРОМЕНІВ У РЕАЛЬНОМУ ЧАСІ

3.1. Впровадження трасування променів засобами сучасних кросплатформних інструментів

Аналізуючи наукові дослідження і досліджуючи сучасні алгоритми оптимізації і удосконалення трасування променів можна помітити що усі досліди проводяться у схожих інтерактивних середовищах і зі схожим апаратним забезпеченням. Зазвичай це використання певного ігрового рушія, зазвичай Unity або Unreal Engine для імплементації алгоритмів і відеокарти Nvidia у якості GPU. Таким чином навіть для базового вивчення технологій рейтрейсингу потрібно мати доволі дорогу апаратну базу, що сильно обмежує доступ до дослідження дотичних технологій. Для усунення цієї проблеми було вирішено розробити мультиплатформний навчально-демонстраційний застосунок для реалізації алгоритму трасування променів.

Незважаючи на значні успіхи у розвитку графічних API, таких як Direct3D 12 або Vulkan, їх використання часто вимагає складної та небезпечної роботи з пам'яттю, що призводить до високих вимог до досвіду програміста та підвищеного ризику помилок. У цьому контексті мова програмування Rust набирає все більшої популярності завдяки своїй здатності забезпечувати безпеку роботи з пам'яттю, високу продуктивність і підтримку багатоплатформності. Завдяки системі перевірки запозичень і статичному аналізу, Rust дозволяє уникати багатьох типових помилок, пов'язаних із керуванням ресурсами, такими як витоки пам'яті або використання звільнених об'єктів.

Ще однією важливою перевагою є використання модуля WGPU — сучасного API для роботи з графікою та обчисленнями, який базується на

сучасних стандартах, та має підтримку усіх популярних графічних API таких як Vulkan, Metal, DirectX 12 та WebGPU.

Важливо зазначити, що для реалізації проєкту у реалістичний термін було прийнято рішення обмежитися лише базовим набором функціоналу, притаманного сучасним передовим технологіям. Серед можливих розширень, які залишилися за межами цієї роботи, — підтримка складних матеріалів, текстурування, більш складні моделі освітлення, обробка каустик, ефекти заломлення, обрахунок об’ємних ефектів та підтримка великих сцен із більшою кількістю об’єктів. Проте навіть у поточній версії цей проєкт демонструє ключові аспекти побудови рендер-системи на Rust та WGPU.

У цьому проєкті розглядається архітектура системи рендерингу, включаючи модулі для управління камерою, обробки геометрії та матеріалів, створення буферів, а також використання обчислювальних та рендер-проходів на GPU. Окремо буде приділено увагу генерації випадкових чисел на GPU, обробці кадрів та акумуляції результатів для отримання якісного зображення з мінімально можливим рівнем шуму.

У результаті буде створено програмний застосунок, який демонструє практичне застосування сучасних технологій Rust та WGPU для вирішення задач фізично коректного рендерингу у реальному часі.

Для реалізації даного проєкту було обрано сучасний технологічний стек, що забезпечує безпечну роботу з пам'яттю, кросплатформенність, високу продуктивність та легку взаємодію з GPU. У центрі цього стеку знаходиться мова програмування Rust, бібліотека WGPU та мова шейдерів WGSL.

Мова програмування Rust поєднує високу продуктивність, порівнянну з C та C++, із системою безпеки пам'яті, заснованою на механізмах власності, запозичень та життєвих циклів. Ці особливості дозволяють компілятору виявляти помилки на етапі компіляції, унеможливлюючи поширені проблеми системного програмування, зокрема витоки пам'яті, доступ до звільнених областей пам'яті та

стану гонитву, коли декілька потоків одночасно звертаються до спільного ресурсу без синхронізації. Для цього проєкту Rust обрано через гарантії безпеки роботи з пам'яттю, відсутність потреби у ручному керуванні ресурсами, підтримку багатопотоковості з коробки, а також можливість використання сучасних бібліотек для GPU-програмування, серед яких WGPU є ключовою.

Бібліотека WGPU забезпечує високорівневий, безпечний і кросплатформний інтерфейс для роботи з графікою та обчисленнями на GPU. Вона є реалізацією специфікації WebGPU та використовує низькорівневі графічні API, такі як Vulkan, Metal, DirectX 12 або OpenGL, як бекенди для доступу до апаратного прискорення. При цьому WGPU приховує складність низькорівневих API, автоматично керуючи станами пайплайнів, ресурсами та синхронізацією, що дозволяє розробникам зосередитися на логіці програми, а не на деталях роботи драйверів. Бібліотека гарантує безпечне управління пам'яттю та ресурсами, виконує перевірки на етапі компіляції та виконання, а також інтегрується з ідіомами мови Rust. У даному проєкті WGPU використовується для ініціалізації GPU-контексту, створення геометрії сцени та матеріалів, підготовки буферів для передачі даних між CPU та GPU (зокрема буферів для камери, об'єктів, випадкових чисел), роботи з текстурами для зберігання проміжних та фінальних результатів рендерингу, налаштування Bind Groups для передачі ресурсів у шейдери, а також для створення обчислювального пайплайна для виконання трасування шляхів на GPU і рендер-пайплайна для виводу фінального зображення на екран.

Мова шейдерів WGSL (WebGPU Shading Language) використовується для написання програм, що виконуються на GPU у середовищі WebGPU. Вона розроблена як сучасний, зрозумілий та безпечний інструмент для реалізації обчислювальних та графічних програм. WGSL характеризується суворим синтаксисом, сильною типізацією та інтеграцією з системою ресурсів WebGPU, зокрема буферами, текстурами та об’єднаними групами об’єктів. Мова забезпечує

захист від помилок доступу до пам'яті завдяки вбудованим перевіркам валідності ресурсів. У даному проєкті WGSL використовується для написання обчислювального шейдера, що реалізує алгоритм трасування шляхів, та рендер-прохідного шейдера, який відповідає за вивід фінального зображення на екран.

Таким чином, обраний стек технологій дозволив поєднати переваги сучасних мови програмування та бібліотек для GPU-програмування, забезпечуючи високу продуктивність, безпеку роботи з пам'яттю та кросплатформність у реалізації проєкту.

3.2. Модульна архітектура програмної реалізації трасування променів

Проєкт побудовано за модульною архітектурою, що дозволяє розділити складну систему на логічно ізольовані компоненти, кожен із яких відповідає за окремий аспект роботи рендерера. Усі модулі взаємодіють між собою через чітко визначені інтерфейси, що спрощує підтримку, тестування та подальший розвиток проєкту. Основу архітектури складають модулі для управління станом GPU, опису сцени, шейдерів, буферів, обробки вводу користувача та фінального рендерингу.

Центральним елементом проєкту є модуль GpuState, який відповідає за ініціалізацію контексту GPU за допомогою бібліотеки WGPU, а також обробку вводу користувача та виклик рендер-процесу. Модуль GpuState також забезпечує підготовку вікна для рендерингу, управління конфігурацією, обробку подій (зокрема зміни розміру вікна або взаємодію з камерою) та викликає методи рендерингу через об'єкт Pipeline. Взаємодія з користувачем реалізована через перехоплення подій бібліотеки winit, що дозволяє обробляти сигнали клавіатури (пересування камери вперед, назад, вліво, вправо) та миші (обертання камери навколо осей).

Модуль Pipeline реалізує основну логіку рендерингу сцени на GPU. Він відповідає за створення всіх необхідних буферів, зокрема буфера камери, буфера об'єктів, буфера лічильника семплів, випадкової текстури для генерації шуму та текстурних буферів для зберігання проміжних результатів рендеру. У межах Pipeline також створюється обчислювальний пайплайн для виконання шейдера Path Tracing, рендер-пайплайн для відображення результату на екран, а також bind-групи для організації доступу до ресурсів у шейдерах. Архітектура Pipeline побудована таким чином, щоб забезпечити підтримку Ping-Pong буферів — механізму, який дозволяє зберігати результати рендерингу з попереднього кадру та використовувати їх під час обчислень поточного кадру для акумуляції фінального зображення.

Опис сцени, що використовується у рендері, задається безпосередньо в коді проєкту у вигляді набору об'єктів, які представляють геометричні примітиви (сфери та площини) та матеріали. Геометрія сцени описується структурами модуля Geometry, де кожен об'єкт має параметри позиції, розміру, орієнтації та типу. Матеріали задаються у модулі Material та підтримують базові моделі розсіювання світла, такі як Lambertian для матових поверхонь, Dielectric для імітації прозорого скла та Light — джерело світла. Для кожного матеріалу визначаються колір, коефіцієнти відбиття або заломлення та інші параметри, необхідні для обчислень у шейдері.

Модуль Camera відповідає за опис камери, яка визначає положення, напрямок погляду, фокусну відстань та кут розфокусування. Камера реалізована як окрема структура з методами для пересування у просторі та обертання навколо осей, що дозволяє користувачу взаємодіяти зі сценою в режимі реального часу. Дані камери передаються у GPU у вигляді спеціального буфера, який потім використовується у шейдері для генерації променів, що імітують рух світла у просторі сцени.

Ключову роль у проєкті відіграють шейдери, написані за допомогою технології WGSL. Обчислювальний шейдер реалізує основний алгоритм Path Tracing, що виконується на GPU. Він обчислює шляхи променів, взаємодію світла з геометрією та матеріалами сцени, а також накопичує результати в буферах. Шейдер працює з випадковою текстурою для генерації шуму та використовує дані попереднього кадру для поступового вдосконалення зображення. Рендер-прохідний шейдер виконує функцію пост-обробки: він бере фінальний кадр із буфера та відображає його на екрані, використовуючи квадрат вікна як геометричну основу.

Кожен цикл програми складається з кількох етапів. Спочатку обробляються події користувача, оновлюється положення камери у разі взаємодії, а також перемикаються буфери Ping-Pong для підготовки до нового кадру. Далі запускається обчислювальний шейдер Path Tracing, який генерує новий фрейм, після чого запускається рендер-прохід для виводу зображення на екран. Лічильник семплів оновлюється покадрово, що дозволяє шейдеру поступово покращувати якість фінального зображення. Завдяки такій модульній архітектурі кожен компонент системи виконує окрему задачу, а їх взаємодія забезпечує повноцінний рендеринг сцени з фізично коректним освітленням.

3.3. Інтерактивне керування камерою і цикл оновлення зображення

На старті застосунок створює об’єкт GpuState, який бере на себе всю роботу з підготовки графічного середовища. Процес можна уявити як послідовність трьох великих кроків.

Спершу GpuState звертається до драйвера WebGPU, щоб дізнатися, які відеоадаптери доступні, і вибирає той, що найкраще підходить для поточного девайса та формату відображення. Після цього вибраний адаптер виділяє логічний пристрій, канал для надсилання команд на GPU і чергу, через яку ці команди потраплятимуть у графічний процесор. Одразу після отримання

пристрою поверхню вікна конфігурують — задають роздільність, формат кольору і режим презентації кадрів.

Коли зв’язок із GPU встановлено, GpuState передає ініціалізацію модулю Pipeline. Той створює:

- буфер камери з поточними координатами та орієнтацією;
- буфер геометрії, у який серіалізуються параметри сфер, площин і джерела освітлення;
- буфер лічильника семплів, що зберігатиме кількість уже накопичених кадрів;
- дві однакові кольорові текстури- одна слугуватиме джерелом даних попереднього кадру, інша — приймачем нового по технології Ping-Pong;
- двовимірну 32-бітову випадкову текстуру тієї самої роздільності, що й кадр, — кожна її комірка містить початкове зерно для генератора випадкових чисел.

Далі модуль Pipeline компілює два WGSL-шейдери. Перший — RayTracer — налаштовується як обчислювальний пайплайн, що читатиме буфер камери, геометрію, лічильник семплів і випадкову текстуру, а результат записуватиме у «зберігаючу» Ping-Pong текстуру. Другий — Shader — налаштовують як графічний пайплайн. Він малює повноекранний квадрат і в кожному фрагменті вибирає колір із «актуальної» Ping-Pong текстури, щоб показати зображення на екрані. Усі потрібні буфери й текстури об’єднують у групи прив’язок, щоб шейдери могли отримувати до них доступ безпосередньо на GPU.

У підсумку перша фаза завершується повністю готовим набором ресурсів: контекстом пристрою, налаштованою поверхнею виводу, буферами для камери й сцени, випадковою та кольоровими текстурами, а також двома шейдерними пайплайнами, які знають, де шукати необхідні дані і куди записувати результати. Саме на цій — разовій — підготовці тримається вся подальша робота рендерера.

Після завершення ініціалізації програма переходить у головний цикл. Для демонстрації того що трасування променів відбувається у реальному часі було реалізовано простий механізм керування камерою, який забезпечує інтерактивність сцени. Усі сигнали від клавіатури й миші потрапляють із вікна через winit до методу обробки подій модуля GpuState.

Для коректної акумуляції Path Tracing зображення будь-який, навіть мінімальний, зсув або обертання камери вважається зміною сцени. На такій події лічильник семплів скидається у нуль, аби почати накопичення заново вже з нової точки спостереження.

Механізм керування камерою реалізовано наступним чином:

- При надходженні події натискання стрілки GpuState викликає відповідний метод камери: позаду цього виклику прихована проста операція — вектора позиції додається або віднімається напрямний вектор уперед-назад чи праворуч-ліворуч, масштабований сталим кроком.
- Подія руху миші обробляється так: програма запам’ятовує попередню позицію курсора, коли миша змістилася, різниця координат проєктується у відсотки від ширини й висоти вікна, після чого виконується поворот камери навколо вертикальної або горизонтальної осі на кут, пропорційний цим відсоткам.

Щойно камера змінила стан, GpuState перезаписує буфер камери у відеопам’ять, а буфер лічильника семплів одразу отримує нульове значення — це дає шейдеру зрозуміти, що з цього кадру треба починати новий цикл накопичення та ігнорувати попередній шум. Усю цю логіку приховано всередині одного методу модуля. Ззовні рендер-цикл отримує лише факт, що кадр потребує повного перерахунку, що й забезпечує інтерактивний і при цьому фізично коректний процес трасування променів.

Після обробки подій користувача програма переходить до формування чергового кадру. На самому початку модуль рендерингу міняє місцями дві

службові текстури: одну призначає джерелом даних попереднього кадру, а другу — приймачем нового. Така нескладна операція забезпечує механізм ping-pong, коли результати накопичуються без копіювання зайвих буферів.

Далі програма відкриває обчислювальний прохід і запускає шейдер Path Tracing. Він отримує дані камери, масив об’єктів сцени, лічильник семплів, початкові випадкові значення та текстуру минулого кадру. Для кожного пікселя шейдер генерує стохастичний промінь, виконує до десяти відскоків, взаємодіє з матеріалами й геометрією та зрештою обчислює новий колір. Отримане значення по формулі середнього зважується з кольором, що зберігся з попереднього кадру: кількість уже накопичених семплів виступає ваговим коефіцієнтом. Підсумок записується у текстуру, позначену як поточний приймач.

Коли обчислення завершені, рендер-прохід малює повноекранний квадрат. Вершинний шейдер передає позиції вершин та текстурні координати, а фрагментний вибирає колір із оновленої текстури й віддає його на вихід, завдяки чому користувач одразу бачить результат нового семплу.

Після того як GPU виконав обидва проходи, процесор передає кадр на поверхню відмальовування, а лічильник семплів у відповідному буфері збільшується на одиницю. Якщо камера залишалася нерухомою, з наступним кадром шейдер продовжить акумулювати маршрути світла, що поступово знижує шум і підвищує якість зображення; якщо ж камера змінює позицію, лічильник оновлюється й накопичення семплів починається з нуля.

Таким чином один повний обхід циклу послідовно виконує перемикання буферів, розрахунок нового семплу в обчислювальному шейдері, відображення текстури на екран та підготовку всієї системи до наступного кадру.

Коли всі етапи обчислень завершені, програма пакує команди виконання у буфер і надсилає їх на GPU. Після цього графічний процесор отримує доступ до фінальної текстури, яка слугує відображенням кадру на екрані. Одночасно лічильник семплів, що відображає кількість накопичених обчислень для кожного

пікселя, оновлюється на одиницю, сигналізуючи, що завершився ще один цикл накопичення. Завдяки цьому новий кадр враховує попередні результати і поступово вдосконалює зображення.

Завершення рендерингу супроводжується перемиканням текстур Ping-Pong: буфер, який щойно отримав результат, стає джерелом для обчислень наступного кадру, а буфер-джерело — приймачем. Перед завершенням циклу також оновлюється випадкова текстура, щоб забезпечити незалежні стохастичні значення для наступного кадру, гарантуючи правильне формування нових променів і варіативність результату. Завершальний крок — запит на перемальовування вікна, після якого цикл починається знову.

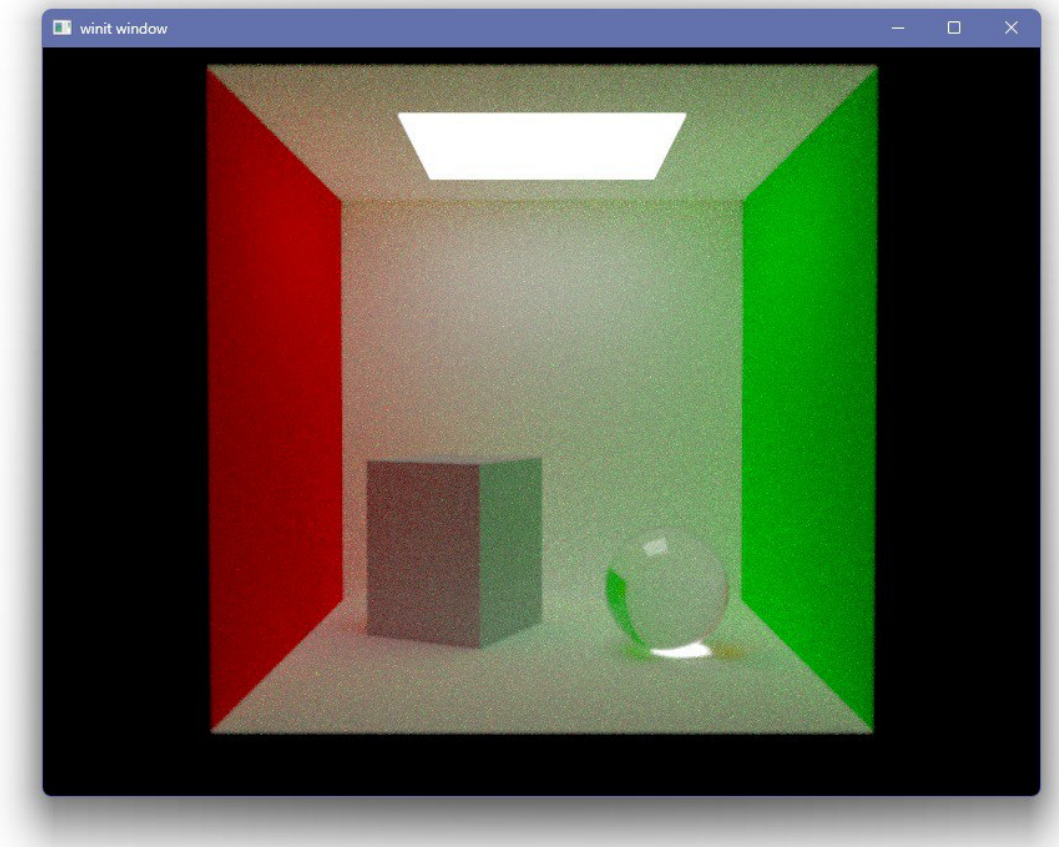

Рис. 3.1 Результат циклу рендерингу

3.4. Порівняльний аналіз якості та продуктивності трасування на Windows, Linux і MacOS

Для кросплатформних тестів використовувались наступні пристрої та операційні системи:

- Платформа Windows 11 24H2, процесор AMD Ryzen 8 5800x, відеокарта AMD Radeon 6800 XT, 32 RAM, API DirectX12;
- Платформа Ubuntu 24.04.2 LTS, процесор AMD Ryzen 8 5800x, відеокарта AMD Radeon 6800 XT, 32 RAM, API Vulkan;
- Платформа MacOS Sequoia 15.5 (24F74), процесор Apple M1, ARM, 8 RAM, API Metal;

Серед тестів — тест на запуск, візуальний огляд результатів рендерингу, та тест на продуктивність. Оскільки в нашому випадку платформи Windows та Ubuntu розділяють апаратну базу, ми можемо напряму порівняти продуктивність роботи застосунку на цих двох платформах. У випадку із девайсом Apple було б доречно порівнювати його із іншими девайсами на чіпі з архітектурою ARM, але на момент дослідження така можливість відсутня.

Результат тесту на запуск — очікувано всі три версії застосунку успішно запустились на всіх платформах. Візуально при детальному огляду важко знайти різницю між версіями для Windows та Ubuntu. Яскраве освітлення чудово падає на всі поверхні, об’єкти відкидають м’які тіні, сфера зі скла пропускає світло крізь себе збираючи його до купи і утворюючи ефект лінзи. Усі поверхні різною мірою відображають кольори видимі у сцені, витоків світла чи тіні немає.

Версія для MacOS, з іншої сторони страждає на певні графічні артефакти. Картинка загалом виглядає темнішою, тіні більш різкі, спостерігаються проблеми із буферизацією семплів (зелені точки по всій поверхні камери). Проблема із версією на MacOS може бути обумовлена як особливостями доступу до буферів у API Metal, так і може свідчити про апаратні проблеми конкретно даного девайса. У будь-якому випадку для розуміння потрібно проводити тести на інших девайсах

Apple. Так чи інакше сам факт запуску і стабільної роботи на, по суті, мобільному чіпі алгоритму трасування променів у реальному часі це уже вагоме досягнення.

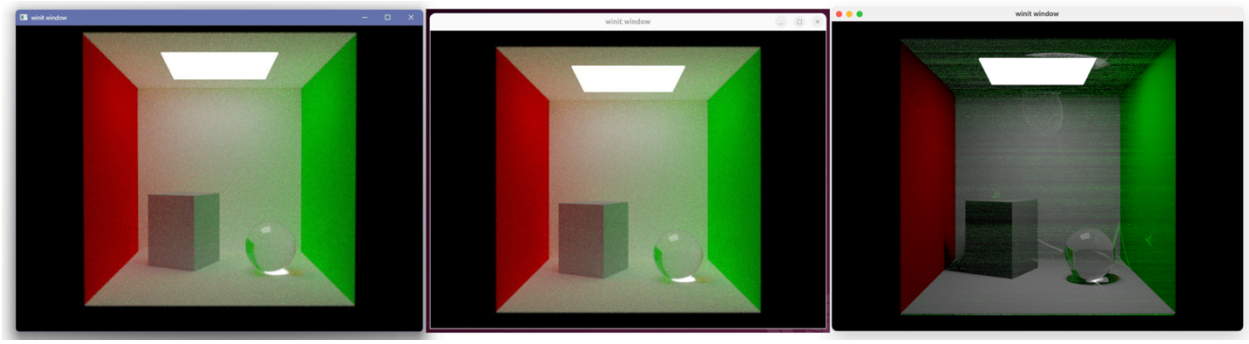

Рис. 3.2. Порівняння версій при запуску
(зліва Windows, по центру Ubuntu, справа MacOS)

Перевірка на продуктивність проводилась заміром використання апаратних ресурсів під час роботи застосунку з алгоритмом трасування променів. Для тесту на швидкодію обмежуючим фактором є дисплей, частота оновлення якого, за рахунок вертикальної синхронізації, не дозволяє застосунку працювати швидше ніж 60 кадрів на секунду. Тест продуктивності на платформах Windows і Ubuntu показав різницю у використанні ресурсу GPU у 8% в користь версії на Ubuntu (88% на Windows, 80% на Ubuntu). Цей результат є сумарним показником як швидкодії Vulkan API так і загальну меншу ресурсну вартість системи Ubuntu перед Windows. Використання ресурсу GPU на платформі MacOS сягає 96% при схожій із Windows і Ubuntu швидкодії, але із уже згаданими вище проблемами візуалізації.

Підсумовуючи результати тестів, неможливо не зазначити, що хоч розроблений застосунок і виконує поставлену задачу, він все ж не позбавлений недоліків. Звісно об’єктивно значні артефакти у роботі програми на MacBook Pro M1 2020, це найбільше розчарування, але сам факт роботи ємного алгоритму обчислення на, по суті, мобільному пристрої дивує, особливо відзначаючи швидкодію і враховуючи що APU цієї лінійки девайсів не підтримує апаратне прискорення трасування променів. Алгоритм безумовно вимагає доопрацювання, особливо не завадило б покращити швидкість перемальовування

при зміні параметрів камери. Також чудовим доповненням стали б такі QOL функції, як імпорт моделей, налаштування керування, підтримка широкого спектра матеріалів тощо. Для навчальних та демонстраційних цілей, було б доречно впровадити механізм моментальної взаємодії із алгоритмом трасування променів — інтенсивність освітлення, ширина вибірки, кількість відбиттів тощо.

ВИСНОВКИ

Не зважаючи на комплексність теми, в результаті дослідження було виконано усі завдання, поставленні на початку роботи. Аналізуючи сучасні методи реалістичної візуалізації, вдалося сформувати групу технічних рішень, які згодом стали ключовими у реалізації застосунку. Враховуючи що переважна більшість досліджень у області фізично коректного рендерингу спрямовані на оптимізацію існуючих алгоритмів, розробка мультиплатформного інструменту для демонстрації трасування променів дає шанс глянути на усю сферу з іншої сторони. Відсутність можливості кросплатформного використання даних технологій сильно обмежує доступ до вивчення цих технологій, оскільки інтерактивне трасування променів вимагає дороговартісної апаратної бази. Відповідно далеко не всі охочі ентузіасти здатні долучитися до розвитку даної області науки.

Розроблений в рамках цієї роботи програмний продукт продемонстрував ключові елементи фізично коректного рендерингу, зокрема накопичення семплів, випадкову генерацію променів та обробку матеріалів, що дозволяє вважати його повноцінним інструментом для вивчення базових принципів трасування променів. Обрані для реалізації застосунку інструменти, зокрема RUST та WGSL, чудово себе проявили як у зручності розробки, так в робочій швидкодії. Аналізатор коду відловлює більшість помилок при роботі з пам’яттю ще на етапі компіляції. Кросплатформне тестове дослідження показало оптимістичні результати. Хоча на деяких платформах спостерігаються значні вади візуалізації, більшість тестів все ж показали високий рівень якості картинки і швидкодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бербега В.О. Спосіб апаратно-програмного трасування променів у задачах тривимірної симуляції: магістерська дис. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», 2022. 103 с.
2. Грушко Ю.В. Методи трасування променів у реальному часі: магістерська дис. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», 2018. 125 с.
3. *Документація RUST*. URL: https://doc.rust-lang.org/stable/std/index.html (дата звернення: 25.05.2025).
4. *Документація WGSL*. URL: https://www.w3.org/TR/WGSL (дата звернення: 25.05.2025).
5. Романюк О.Н., Бобко О.Л., Романюк О.В. Перспективи використання технології трасування променів для створення фотореалістичних зображень у реальному часі. *Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2024. Матеріали IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 26-27 вересня 2024 р.* Одеса: ОНТУ, 2024. С.327-328.
6. Сокур О. М., Петрушина Т. І. Проблеми програмного прискорення трасування променів. *Інформатика, інформаційні системи та технології: тези доповідей двадцять другої всеукраїнської конференції студентів і молодих науковців. Одеса, 25 квітня 2025 р.* Одеса, 2025. С.225-230.
7. Фещенко І.О. Оптимізація продуктивності рендерингу методом трасування променів у реальному часі: магістерська дис. Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», 2022. 85 с.

8. Cook R. L., Carpenter L. The Reyes Image Rendering Architecture. *ACM SIGGRAPH Computer Graphics*. 1987. Vol. 21. P. 95–102. DOI: 10.1145/37402.37414.
9. Cook R. L., Porter T., Carpenter L. Distributed ray tracing. *ACM SIGGRAPH Computer Graphics*. 1984. Vol. 18. P. 137–145. DOI: 10.1145/964965.808590
10. Cook R. L., Torrance K. E. A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics*. 1982. Vol. 1. P. 7–24. DOI: 10.1145/357290.357293.
11. Verifying Volume Rendering Using Discretization Error Analysis / T. Etiene et al. *IEEE*. 2013. P. 140–154. DOI: 10.1109/TVCG.2013.90.
12. Glassner A. S. An Introduction to Ray Tracing. 3333 C oyote Hill Road Palo Alto CA 94304 USA : Xerox PARC, 2019.
13. Iglesias-Guitian J. A., Mane P., Moon B. Real-Time Denoising of Volumetric Path Tracing for Direct Volume Rendering. *IEEE*. 2020. DOI: 10.48550/arXiv.2106.08034.
14. Kajiya J. T. The rendering equation. *ACM SIGGRAPH*. 1986. Vol. 20. P. 143–150. DOI: 10.1145/15886.15902.
15. Dynamic Diffuse Global Illumination Resampling / Z. Majercik et al. *ACM SIGGRAPH*. 2021. P. 1–2. DOI: 10.1145/3450623.3464635.
16. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields / Z. Majercik et al. *Journal of Computer Graphics Techniques*. 2019. Vol. 8. URL: https://jcgt.org/published/0008/02/01/ (дата звернення: 25.05.2025).
17. Improving Temporal Antialiasing with Adaptive Ray Tracing / A. Marrs et al. *Ray Tracing Gems NVIDIA*. 2019. P. 353–370. DOI: 10.1007/978-1-4842-4427-2_22.
18. Real-Time Global Illumination using Precomputed Light Field Probes / M. McGuire et al. *ACM I3D*. 2017. DOI: 10.1145/3023368.3023378.
19. Pharr M. Guest Editor's Introduction: Special Issue on Production Rendering. *ACM Trans.*. 2018. Vol. 37. P. 1–4. DOI: 10.1145/3212511.

20. Pharr M., Jakob W., Humphreys G. *Physically Based Rendering:From Theory To Implementation*. URL: https://www.pbr-book.org/ (дата звернення: 25.05.2025).
21. Parallel distributed, GPU-accelerated, advanced lighting calculations for large-scale volume visualization / M. Shih et al. *IEEE*. 2016. DOI: 10.1109/LDAV.2016.7874309.
22. Suffern K. Ray Tracing from the Ground Up. Boca Raton, Florida, United States : A K Peters/CRC Press, 2007.
23. Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*. 1980. Vol. 23. P. 343–349. DOI: 10.1145/358876.358882.
24. Wyman C., Marrs A., Shirley P. Weighted Reservoir Sampling: Randomly Sampling Streams. *Ray Tracing Gems II. Apress*. 2021. DOI: 10.1007/978-1-4842-7185-8_22.
25. Area ReSTIR: Resampling for Real-Time Defocus and Antialiasing / S. Zhang et al. *ACM Transactions on Graphics*. 2024. Vol. 43. P. 1–13. DOI: 10.1145/3658210.

ДОДАТКИ

Додаток А

РОЗРАХУНОК ІНТЕНСИВНОСТІ СВІТЛА ЗГІДНО МОДЕЛІ БУІ ФОНГ ТОНГА

Одна із конвенційних моделей, що були розроблені для заміни алгоритмів візуалізації поверхні – це модель відображення Буі Фонг Тонга. Модель розділяє світло, що відбивається від поверхні, на три компоненти - навколишнє освітлення I_a, яке не залежить від орієнтації поверхні чи положення джерел світла, моделює загальне оточуюче світло у сцені, дифузна компонента $k_d(\vec{N} \cdot \overrightarrow{L_j})$, що, базується на законі Ламберта і відповідає за м'яке, рівномірне розсіювання світла від матових поверхонь, а також спекулярна компонента $k_s(\vec{N} \cdot \overrightarrow{L_j'})^n$, яка моделює яскраві блиски, характерні для глянцевих поверхонь. Напрям відблиску залежить від відносного розташування джерела світла, камери й нормалі поверхні.

Розрахунок інтенсивності світла згідно його моделі відбувається за формулою:

$$I = I_a + k_d \sum_{j=1}^{j=ls}(\vec{N} \cdot \overrightarrow{L_J}) + k_s \sum_{j=1}^{j=ls}(\vec{N} \cdot \overrightarrow{L_J'})^n \quad \text{(A.1)}$$

Де I — відбита інтенсивність світла;

I_a — складова освітлення, що моделює навколишнє світло;

k_d — константа дифузного відбиття;

$\vec{N}$ — одиничний вектор нормалі до поверхні у точці, що розглядається;

$\overrightarrow{L_j}$ — одиничний вектор у напрямку від точки на поверхні до j-го джерела світла;

k_s — коефіцієнт спекулярного відбиття;

L_j' — одиничний вектор, що спрямований у напрямку, який є середнім між напрямком до глядача та напрямком до j-го джерела світла;

n — експонента, що характеризує ступінь глянцевості поверхні.

Додаток Б

РОЗШИРЕНА ТЕРНЕРОМ ВІТТЕДОМ МОДЕЛЬ ОСВІТЛЕННЯ

Запропонована Віттедом модель освітлення це удосконалення класичної моделі Фонга. Удосконалена модель використовує фізично коректний спосіб розрахунку спекулярної компоненти, на основі закону відбиття. Математичне зображення моделі має наступний вигляд:

$$I = I_a + k_d \sum_{j=1}^{j=l_s} \left(\vec{N} \cdot \overrightarrow{L_J}\right) + k_s S + k_t T \qquad \text{(Б.1)}$$

Де S — інтенсивність світла, відбитого у напрямку $\vec{R}$,

T — інтенсивність світла, що проходить у напрямку $\vec{P}$,

k_t— коефіцієнт переданого світла .

Напрямок $\vec{R}$ визначається простим правилом – кут відбиття маж дорівнювати куту перетину променя із поверхнею. Водночас напрямок $\vec{P}$ переданого світла має слідувати закону відбиття. Інтенсивність світла I що надходить до спостерігача від точки на поверхні складається в остновному із компонентів дзеркального відбиття S та переданого світла T як показано на рисунку Б.1.

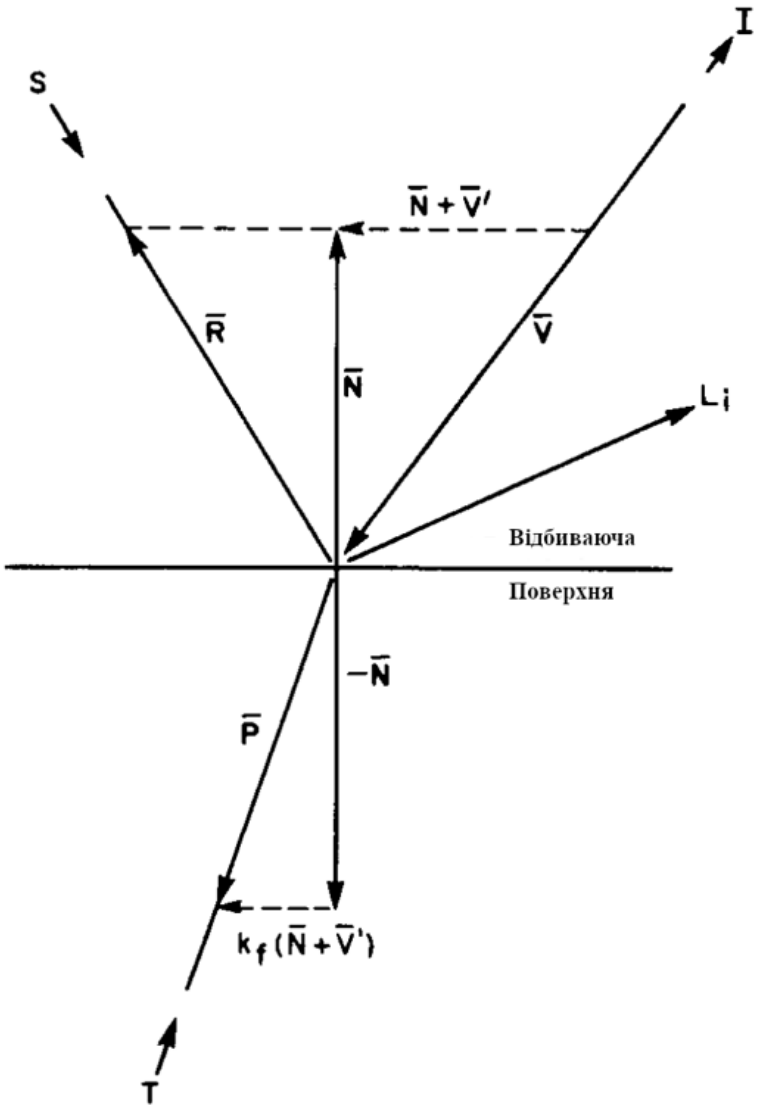

Рис Б.1 Модель освітлення, що враховує дзеркальні відбиття

Додаток В

РІВНЯННЯ РЕНДЕРИНГУ

Джеймс Каджия у 1986 р. запропонував універсальну модель освітлення що враховує пряме і непряме освітлення, дзеркальне, розсіяне та комбіноване відбиття, а також включає вторинні відбиття у глобальному освітленні. Рівняння рендерингу – це інтегральне нелінійне рівняння, на практиці розв’язати його неможливо, а у комп’ютерній графіці наближення до розв’язку зазвичай здійснюється за допомогою методу Монте-Карло. Математичне зображення такого рівняння:

$$I(x,x') = g(x,x')[e(x,x') + \int_S p(x,x',x)\mathrm{I}(\mathrm{x}',\mathrm{x})dx''] \qquad \text{(В.1)}$$

Де $I(x,x')$ — інтенсивність світла, яке передається від точки x' до точки x. Це головна величина, яку потрібно знайти: скільки світла потрапляє в точку спостереження x від поверхні в точці x';

$g(x,x')$ — функція геометричного згасання;

$e(x,x')$ — власне випромінювання в точці x у напрямку x';

$p(x,x',x'')$ — функція розсіювання, що визначає, скільки світла, що прийшло з точки x'', розсіюється в напрямку x через точку x'.

Додаток Д

МОДЕЛЬ ГЛОБАЛЬНОГО ОСВІТЛЕННЯ, ЩО ВРАХОВУЄ КОМПОНЕНТ DDGI

$$L \approx L_e + T_{fd}L_e + L_{DDGI} + T_{fg}L \quad \text{(Д.1)}$$

Де L — вихідне освітлення, тобто кількість світла, що залишає поверхню в заданому напрямку;

L_e — власне випромінювання поверхні;

$T_{fd}L_e$ — оператор розсіяного емісійного освітлення;

L_{DDGI} — апроксимація непрямого розсіяного освітлення;

$T_{fg}L$ — оператор переносу світла для глянцевого компоненту, що описує відбиття від глянцевих або дзеркальних поверхонь.

Додаток Е

РОЗМІЩЕННЯ ІРРАДІАЛЬНИХ ПРОБ ДЛЯ РОБОТИ АЛГОРИТМУ DDGI

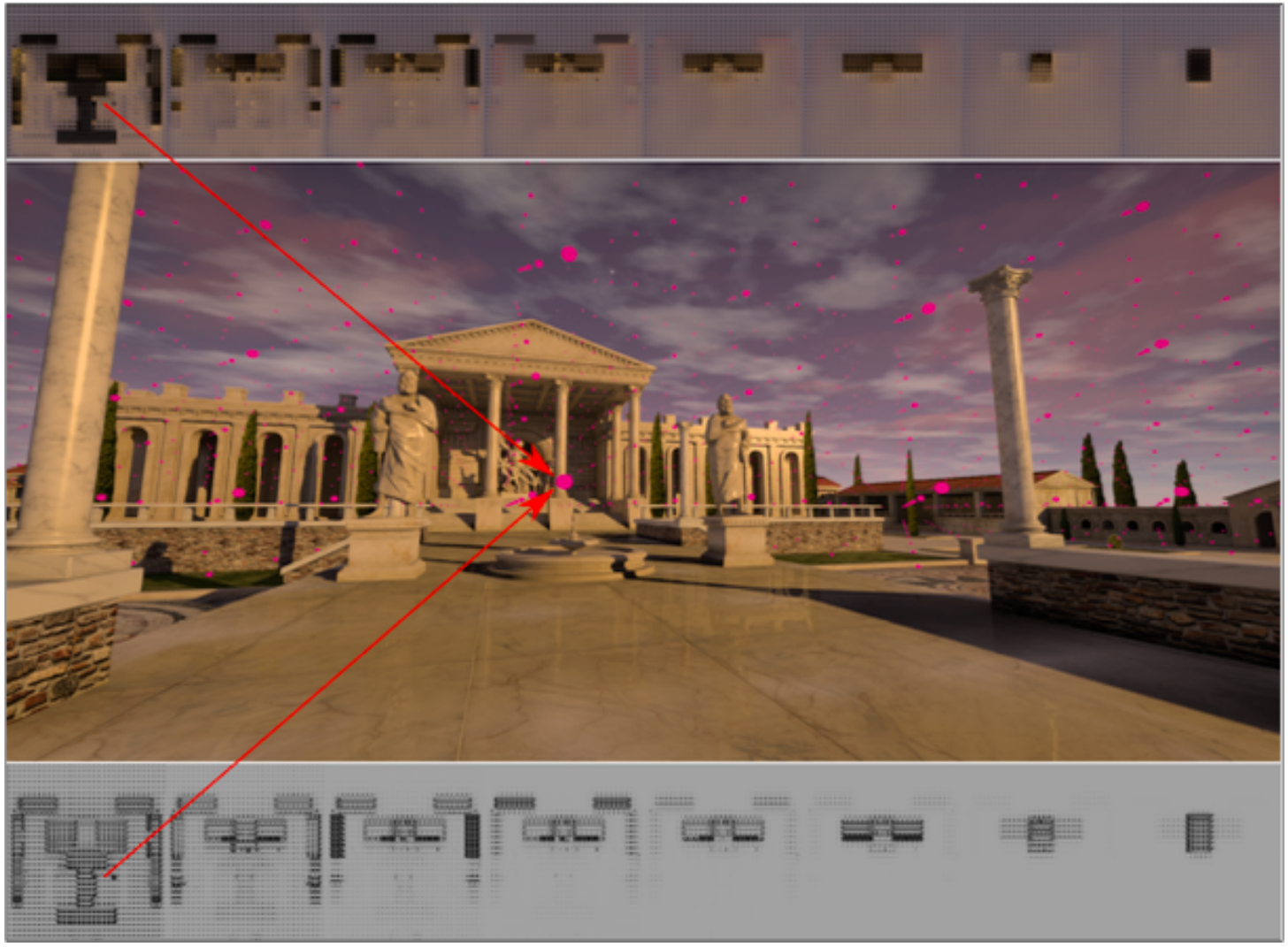

Рис Е.1Візуалізація розміщення іррадіальних проб у просторі сцени

Додаток Є

РІВЕНЬ ШУМУ ДО ЗАСТОСУВАННЯ АЛГОРИТМУ ЗНЕШУМЛЕННЯ

Рис Є.1 Застосування трасування променів для глобального освітлення без застосування знешумлення, відеогра QUAKE II RTX 2019р.

Додаток Ж

МАТЕМАТИЧНА ФОРМУЛА ІНТЕГРАЛУ ОБ’ЄМНОГО РЕНДЕРИНГУ

$$L(x,\omega) = \int_0^z T(x,y)\,[\mu_a(y)L_e(y,\omega) + \mu_s(y)L_s(y,\omega)]dy + T(x,z)L(z,\omega) \quad (Ж.1)$$

Де $L(x,\omega)$ — освітленність у точці x у напрямку ω;

$T(x,y)$ — коефіцієнт пропускання від точки y до x, тобто скільки світла дійшло без поглинання;

$\mu_a(y)$ — коефіцієнт поглинання в точці y;

$L_e(y,\omega)$ — емісія в точці у в напрямку ω;

$\mu_s(y)$ — коефіцієнт розсіювання в точці y;

$L_s(y,\omega)$— інтенсивність розсіяного світла, яке приходить через точку y і прямує в напрямку ω;

$L(z,\omega)$— інтенсивність світла, що приходить з межі середовища (наприклад, з джерела або фону);

$T(x,z)$ — ослаблення освітлення між z та x.