МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра комп'ютерних наук та програмної інженерії

РОЗРОБКА КРОС-ПЛАТФОРМНОГО ФРЕЙМВОРКУ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНИХ 2D ПРОГРАМ

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «бакалавр»

Виконав: студент 4 курсу 12-441 групи
Спеціальності: 121 Інженерія програмного забезпечення
Освітньо-професійної програми:
Інженерія програмного забезпечення
Бреннон Роланд Джеймс
Керівник: ст. викл. Черненко І.Є.
Рецензент: Проценко А.А., C++ developer, Pingle Studio

Херсон – Івано-Франківськ - 2024

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API ……Application Programming Interface
OpenGL……..Open Graphics Library
ECS……Entity Component System
GL……...…… Graphics Library
VBO……...…...…… Vertex Buffer Object
IBO……. Index Buffer Object
GLSL……...openGL Shading Library
RGB…….....Red, Green, Blue
sRGB…….standard Red, Green, Blue
FBO……..…..Frame Buffer Object
GUI……...…….Graphics User Interface

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА 6
1.1 Рендерер графіки 6
1.2 Рендер графіки 7
1.3 Елементи графічного рендеру 9
1.4 Render pipeline 13
1.5 Камера та матриці проекцій 16
РОЗДІЛ 2 ПРАКТИЧНА ЧАСТИНА 19
2.1 Фреймворк 19
2.2 Графічний/Користувацький Інтерфейс 21
2.3 Модуль Аудіо для виводу просторового звуку 23
2.4 Модуль Фізика 25
2.5 Скриптування та ігрова логіка 26
2.6 Оптимізація та рефактор 28
2.7 Тестування 29
ВИСНОВОК 30
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 31

ВСТУП

Актуальність теми. У сучасному інформаційно-комунікаційному середовищі зростає потреба у розробці програмного забезпечення, яке може працювати на різних платформах. З появою мобільних пристроїв та різноманітних операційних систем стає важливим розроблення програмного забезпечення, яке може легко адаптуватися до різних середовищ, а саме, створення фреймворку, який дозволить розробникам ефективно впроваджувати програми та інструменти на різних пристроях та платформах. Такий крос-платформний фреймворк може мати широкий потенціал впливу на різні сфери. Ця технологія може бути використана для:

1) Розроблення елементарних ігор та інших 2D-програм сфери розваг на будь-яких платформах.
2) Створення спеціальних розвиваючих програм та ігор для дітей та людей з обмеженими можливостями.
3) Розроблення програм та інструментів загального призначення та різних напрямків.

З огляду на вищезазначене актуальним є створення фреймворку PixiPulse, який надасть розробникам потужний інструментарій для створення інтерактивних 2D-програм з опором на мультимедійний контент. PixiPulse буде мати простий інтуїтивно зрозумілий інтерфейс, що зробить його ідеальним вибором для розробки ігор, освітніх ігор, анімацій та інших проектів, що потребують інтерактивної взаємодії та візуального контенту. Він забезпечить широкий набір функцій для роботи з графікою, аудіо та відео, дозволяючи легко інтегрувати різні мультимедійні елементи в програму.

Він може слугувати інструментом для створення доступних додатків, адаптованих до індивідуальних потреб користувачів з різними фізичними або когнітивними обмеженнями. Забезпечення можливості навчання та розваг для уразливих та залежних аудиторій може сприяти їхньому загальному

розвитку та покращенню якості життя. Таким чином, обрана тема не лише відповідає потребам розробників програмного забезпечення, але й має потенціал позитивного впливу на соціальну сферу через створення інклюзивних та освітніх інтерактивних ігор.

Мета дослідження: Метою дослідження є розроблення рушія, здатного полегшувати створення інтерактивних 2D програм.

Для досягнення поставленої мети було поставлено наступні **задачі дослідження**:

- Аналіз подібних та суміжних технологій.
- Аналіз можливостей графічного API OpenGL.
- Пошук корисних бібліотек та їх вивчення.
- Визначення структури проекту.
- Розроблення модулів проекту.

Об’єкт дослідження: Графічні API.

Предмет дослідження: Фреймворк для створення інтерактивних 2D-програм з опором на мультимедійний контент.

Структура роботи: Робота складається зі змісту, переліку умовних скорочень, вступу, двох розділів, висновків та списку використаних джерел.

РОЗДІЛ 1
ТЕОРЕТИЧНА ЧАСТИНА

1.1 Рендерер графіки

Першою сходинкою у реалізації проекту стане рендерер.

Рендерер графіки — це програмне чи апаратне забезпечення, яке відповідає за обробку та відображення графічної інформації на екрані комп'ютера чи іншого пристрою.

Для його розробки було декілька різних графічних API на вибір:

- OpenGL,
- DirectX11,
- Vulkan.

В таблиці 1 наведемо аналіз цих API.

Таблиця 1. Аналіз API

Характеристика	**OpenGL**	**DirectX11**	**Vulkan**
Тип API	Крос-платформний	Пропріетарний (Microsoft)	Крос-платформний
Платформи	Windows, macOS, Linux, Android, iOS	Windows, xbox	Windows, Linux, Android, macOS, iOS
Складність	Низька-середня	Середня	Висока
Продуктивність-ефективність	Середня	Середня	Висока
Графічні можливості	Середні	Середні	Високі
Асинхронність	Відсутня	Присутня	Присутня
API Overhead	Високий	Високий	Низький

Асинхронність: визначає можливість цих API обробляти декілька операцій паралельно або незалежно одна від одної.

API Overhead: визначає наскільки ефективно програма може використовувати графічний інтерфейс для взаємодії з апаратним обладнанням, наприклад графічною картою.

З огляду на результати аналізу було обрано саме OpenGL.

OpenGL — це кросплатформенний API для реалізації графічних програм. Він надає функціонал для взаємодії з апаратним забезпеченням і виконання різних завдань, пов'язаних з графікою, включаючи обробку подій введення, створення тривимірних об'єктів та використання шейдерів для додаткового контролю над відображенням.

Стандарт OpenGL (Open Graphics Library — відкрита графічна бібліотека) був розроблений і затверджений у 1992 році провідними фірмами в сфері індустрії програмного забезпечення як ефективний апаратно-незалежний інтерфейс, придатний для реалізації на різних платформах. Основою стандарту стала бібліотека IRIS GL, розроблена фірмою Silicon Graphics Inc.[2]

1.2 Рендер графіки

Рендер графіки — це процес створення зображення з тривимірних моделей чи сцен для подальшого відображення на екрані користувача.

Процес рендеру графіки присутній у найрізноманітніших сферах професійної діяльності: кіноіндустрія, ігрова індустрія, тощо. Це й не дивовижно, тому що з кожним роком можливості графіки так чи інакше ростуть, і таким чином її стає все складніше відрізняти від реальності.

Цей процес дозволяє досить сильно зекономити час та сили команди і підвищити ефективність роботи працівника у команді.

Рендер графіки — це один з найскладніших в технічному плані етапів у роботі з 3D графікою. Щоб пояснити цю операцію простою мовою, можна привести аналогію з роботами фотографів. Для того, щоб з’явилася готова

фотографія, фотографу потрібно пройти через деякі технічні етапи, наприклад, прояв плівки або друк на принтері. Приблизно такими ж технічними етапами обтяжені і 3d художники, які для створення підсумкового зображення проходять етап налаштування рендеру.[3]

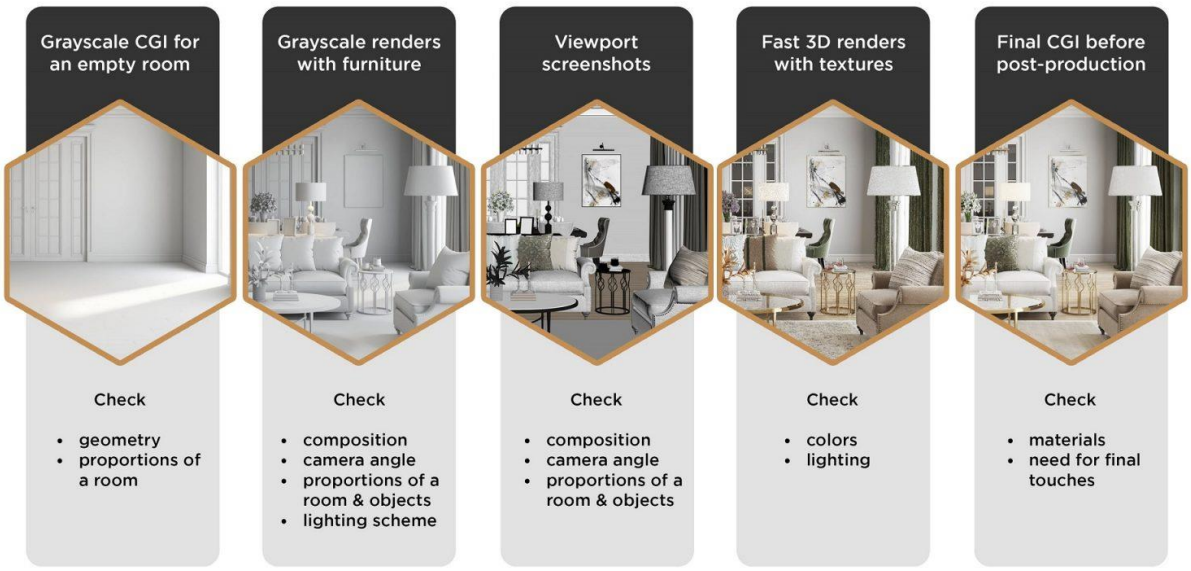

Рисунок 1.1 Приклад рендеру випадкової сцени

Процес рендеру конкретно у OpenGL може включати наступні кроки:

1. Заповнення буферів даними, такими як вершинні дані, текселі(пікселі) текстур і інші.
2. Використання вершинного та піксельного шейдерів для обробки геометрії та кольору.
3. Застосування текстур для покращення вигляду моделі.
4. Використання глибинного буфера для вирішення проблеми затінення та визначення, які об'єкти перебувають перед іншими для створення перспективи та багатомірності.
5. Виведення результатів на екран через фреймбуфер.

1.3 Елементи графічного рендеру

Елементи графічного рендеру — це компоненти та ресурси, які використовуються для створення візуальних зображень у комп'ютерній графіці.

Найпершим задіяним елементом є буфер.

Буфери (Buffer Object) у OpenGL — це об’єкти OpenGL, які зберігають масив неформатованої пам’яті, виділеної контекстом OpenGL для різних цілей. Вони можуть використовуватися для зберігання даних вершин, даних пікселів, отриманих із зображень або кадрового буфера, і багатьох інших речей. [4]

Вершинний буфер (Vertex Buffer). Це буфер, який містить геометричні дані, такі як координати вершин, нормалі, текстурні координати тощо. Він дозволяє виводити геометрію об’єкта за один єдиний виклик функцій API. Це в свою чергу дозволяє розробнику ефективно використовувати пам’ять програми, запобігати багатьох помилок та проблем.

Простими словами кажучи, це загальний термін для звичайного буферного об’єкта, коли він використовується як сховище даних масиву вершин. Він нічим не відрізняється від будь-якого іншого буферного об’єкта, просто його значення, можна використовувати як вихідні значення для масивів вершин. [5]

Буфер індексів (Index Buffer). Містить індекси вершин, що вказують порядок їх використання для створення трикутників або інших примітивів. В 3D-графіці часто вершини належать одразу декільком трикутникам. І саме для того, щоб не потрібно було багато раз використовувати одні і ті самі ж вершини у вершинному буфері була введена індексація вершин. Завдяки цьому буферу можна призначити вершинам індекси та багаторазово використовувати їх просто вказавши цей індекс.

Цей буфер зберігає список байтів, беззнакових коротких або цілих чисел, які повідомляють OpenGL, яку вершину в наразі пов’язаних VBO малювати наступною.[6]

Буферне сховище(Storage Buffer). Дозволяє зберігати інформацію для її використання іншими буферами.

Другим використовуваним елементом є шейдер.

Шейдер — це програма, яка використовується для визначення різних етапів обробки графіки, таких як обробка вершин, фрагментів(пікселів), геометрії, тощо. Шейдери виконуються на графічному процесорі і дозволяють відтворювати на екрані такі складні ефекти, як тіні, освітлення, ефекти розмиття та інші. Шейдери надають код для певних програмованих етапів конвеєра візуалізації. [7]

Існує багато видів шейдерів, але у роботі будуть розглядатися лише два.

- *Вершинний шейдер (Vertex Shader)*. Ці шейдери використовуються для обробки кожної вершини в моделі, включаючи їх позицію, текстурні координати, нормалі, колір, тощо. Вершинні шейдери використовуються для трансформації вершин у багатомірному просторі.

- *Піксельний шейдер (Pixel Shader або Fragment Shader).* Ці шейдери відповідають за обробку кожного пікселя на екрані. Вони визначають кольори та інші властивості кожного пікселя, це може бути освітлення, текстури, ефекти, тощо.

Загалом, різні шейдери комбінуються та дозволяють досягти просто неймовірних ефектів на екрані користувача.

Для роботи з шейдерами буде використовуватися мова написання шейдерів *GLSL*.

Це — мова програмування, розроблена спеціально для написання шейдерів у контексті OpenGL. Хоча завдяки розширенням OpenGL і існує кілька мов шейдерування доступних для використання, GLSL підтримуються безпосередньо OpenGL без розширень.

GLSL є унікальною завдяки своїй моделі компіляції. Її модель компіляції більше схожа на стандартну парадигму C (бо сама мова дуже схожа на C), вона контролюється кількома типами об’єктів. [8]

Вона є вбудованою мовою в стандарт OpenGL і використовується для програмування вершинних, геометричних, фрагментних(піксельних), обчислювальних, теселяційних контрольних та теселяційних оцінювальних шейдерів, тобто для усіх можливих.

PixiPulse > Shaders > FFShaders > Source > triangle.vert

```
1   #version 430 core
2
3   layout(location=0) in vec3 inPosition;
4   layout(location=1) in vec2 inUVs;
5
6   layout(location=0) out vec2 outUVs;
7
8   uniform Matrices {
9       mat4 uViewProjection;
10      mat4 uModel;
11  };
12
13  void main(void) {
14      gl_Position = uViewProjection * uModel * vec4(inPosition.xyz, 1.0);
15
16      outUVs = inUVs;
17  }
```

Рисунок 1.2 Приклад вершинного шейдеру GLSL

Згодом, шейдери об’єднуються у шейдерній програмі. Програма шейдерів містить різні типи шейдерів і використовуватиметься під час рендеру. [9]

Основна функція шейдерних програм — забезпечити розробникам контроль над тим, як графічні дані обробляються та відображаються на екрані.

Вони надають програмістам велику гнучкість роботи та контроль над графічним процесом, що дозволяє створювати складні та реалістичні візуальні ефекти.

Останнім з основних елементів є текстура.

Текстури — це дуже важливий елемент рендеру графіки. Саме завдяки ним користувач може побачити “обгортку” об’єкту на екрані. Вони використовуються для симуляції матеріалів, накладання тіней та освітлення, деталізації, накладання ефектів.

Зображення визначається як окремий масив пікселів певної розмірності (1D, 2D або 3D), певного розміру та певного формату. Текстура — це вміст одного або кількох зображень. Але текстури не зберігають довільні зображення; текстура має певні обмеження щодо зображень, які вона може містити. Існує три визначальні характеристики текстури, кожна з яких визначає частину цих обмежень: тип текстури, розмір текстури та формат зображення, який використовується для зображень у текстурі.[10]

Текселі текстури — простими словами, це пікселі текстур, їх фрагменти.

Зрозуміло, що для досягнення реалістичного відображення графіки необхідно поєднати різні елементи, включаючи текстури, з іншими технологіями, наприклад кольоровим або глибинним буферами. Не треба плутати їх з вершинними, індексними та буферами сховищ. Це зовсім різні поняття

Давайте розглянемо роль кольорового та глибинного буферів у процесі рендеру графіки більш детально.

Буфер кольору (Color Buffer). Буфери кольорів – це ті, на які користувач проектує кольори. Вони містять колірні дані RGB або sRGB, а також можуть містити альфа-значення для кожного пікселя в фреймбуфері (кадровому буфері). У ньому може бути декілька кольорових буферів.[11]

Буфер глибини (Depth Buffer). Використовується для збереження інформації про глибину пікселів для правильного відтворення перспективи. Це досягається завдяки тесту глибини, який виконується у просторі огляду (viewport) після запуску буферу індексів.

Буфер глибини містить значення глибини від 0,0 до 1,0 і порівнює свій вміст із значеннями z усіх об’єктів на сцені, які видно з боку глядача. Ці z-

значення у просторі огляду можуть бути будь-якими значеннями між ближньою та дальньою площинами усіченої проекції. [12]

Коли увімкнений режим тестування глибини, то OpenGL порівнює значення глибини фрагменту (індексу) з даними, які знаходяться у буфері глибини, і якщо цей тест проходить успішно, то фрагмент візуалізується на екрані, а буфер глибини оновлюється новим значенням. Якщо ж тест проходить невдало, то фрагмент попросту відкидається.

В цілому, використання текстур дозволяє надати об’єкту привабливий зовнішній вигляд, завгодний розробнику або користувачу, дає можливість застосувати різноманітні складні ефекти, що в свою чергу грає ключову роль у візуалізації сцени.

Рисунок 1.3 Приклад матеріалів та текстур

1.4 Render pipeline

Потік рендеру (Render pipeline) — це послідовність кроків, яку виконує OpenGL під час відтворення об’єктів. В ньому використовуються як вже перелічені елементи графічного рендеру, так і інші, більш складні.

Фреймбуфер — це об'єкт, який використовується для зберігання графічних даних перед їх відображенням на екрані. Фреймбуфер складається з різних компонентів, це можуть бути прикріплені зображення (кольорові,

глибини, трафаретні), буфери глибини, буфер трафарету (який відноситься безпосередньо до фреймбуфера та не має нічого спільного з іншими буферами), тощо. Усі ці компоненти використовуються для рендеру та подальшої обробки графічних об'єктів.

OpenGL має два типи фреймбуферів: фреймбуфер за замовчуванням, який надається контекстом OpenGL; і фреймбуфери, створені користувачем, які називаються об’єктами буфера кадрів (FBO). Буфери для фреймбуферів за замовчуванням є частиною контексту і зазвичай представляють вікно або пристрій відображення.[13]

Простими словами, це контейнер для різних буферів.

Основною перевагою є те, що фреймбуфер дозволяє одночасно працювати з кількома буферами та виконувати різні обчислення та обробки перед виведенням на екран.

Як і усі вищеназвані деталі, використання фреймбуферів також відкриває перед розробником велику кількість можливостей.

Повертаючись до поняття потоку рендеру, атрибут вершини та інші дані проходять послідовність кроків для створення остаточного зображення на екрані. У цьому конвеєрі зазвичай є 9 кроків, більшість з яких є необов’язковими, а багато з них програмуються.[14]

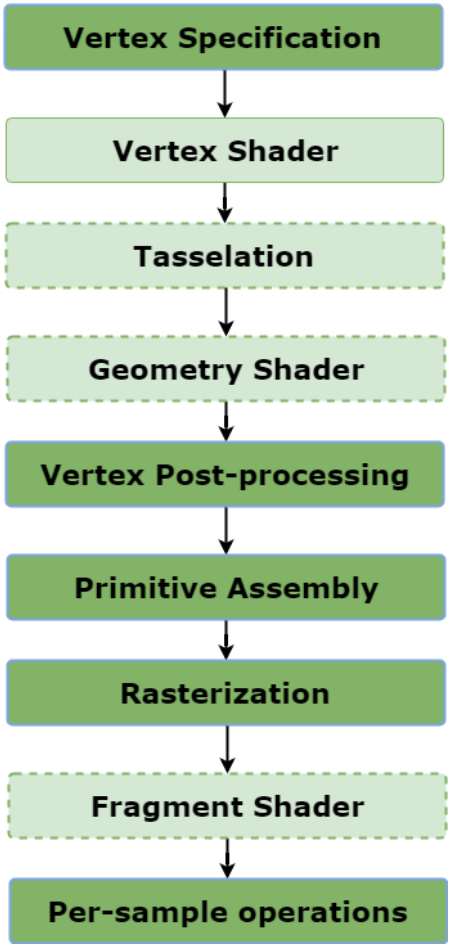

Рисунок 1.4 Render Pipeline

В проекті даного дослідження ця послідовність може виглядати таким чином:

- Одразу після ініціалізації контексту OpenGL, буфери заповнюються даними, це можуть бути вершинні дані, текселі текстур, тощо.
- Створюються та компілюються задумані розробником шейдери.
- Потім, для покращення зовнішнього вигляду моделі, накладаються текстури.
- Глибинний буфер вирішує проблеми затінення та визначення, які об'єкти перебувають перед іншими для створення перспективи.
- Створення та налаштування фреймбуферів для відображення об’єктів на екрані з використанням усіх минулих кроків.

  Згодом залучаються камера та матриці проекцій.

1.5 Камера та матриці проекцій

Камера та матриці проекцій в OpenGL використовуються для визначення того, як тривимірний світ перетворюється у двовимірне зображення на екрані. Камера визначає точку спостереження та напрямок, у якому “дивиться” спостерігач, а матриці проекцій відповідають за трансформації координат об'єктів для правильного відображення на екрані.

Але насправді камера не рухається.

Щоб створити враження, що камера рухається, програма OpenGL повинна перемістити сцену за допомогою трансформації, розмістивши її на матриці перегляду. Це зазвичай називають трансформацією перегляду.[15]

Матриця перегляду (model view projection) відповідає за переміщення об’єктів у сцені, щоб імітувати зміну положення камери, змінюючи те, що глядач зараз може бачити. [16]

Простими словами, камера — це очі, якими користувач “взаємодіє” з віртуальним світом програми. Матриці проекцій, в такому випадку, це лінзи, які маніпулюють зображенням на сітківці ока до того, як інформація передається далі. Як і інші елементи, матриці теж відрізняються між собою:

- *Ортогональна матриця проекції (Orthographic Projection Matrix)*: Це матриця, яка використовується для створення ортографічної проекції, при якій всі паралельні лінії залишаються паралельними на відображенні. Зазвичай вона використовується для рендеру 2D графіки або для відображення об'єктів без перспективи, таких як інтерфейси користувача.
- *Перспективна матриця проекції (Perspective Projection Matrix)*: Це матриця, яка використовується для створення перспективної проекції, яка відтворює ефект перспективи, де об'єкти, що знаходяться далі, здаються меншими, ніж ті, що знаходяться ближче

до спостерігача. Зазвичай вона використовується для створення реалістичних 3D сцен у графічних програмах та іграх.

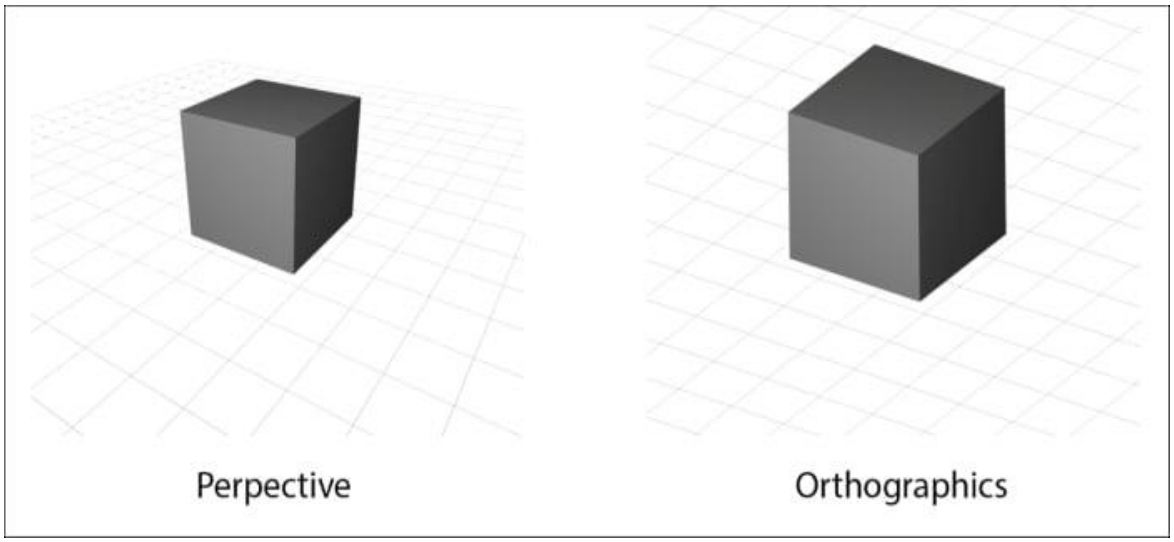

Рисунок 1.5 Порівняння перспективної та ортографічної проекцій

Кожен тип матриці проекції має свої особливості та можливості використання. Вибір підходящого типу залежить від потреб програми або ігри, а також від бажаної перспективи та відображення сцени.

Подальший процес рендеру включає створення цих матриць, об'єднання їх та передачу у вершинний шейдер для трансформації координат вершин. Результати обчислень передаються піксельному шейдеру для додаткової обробки та відображення на екрані.

Цей підхід дозволяє врахувати перспективу та кут огляду камери, оптимально відображаючи тривимірний світ на екрані.

РОЗДІЛ 2
ПРАКТИЧНА ЧАСТИНА

2.1 Фреймворк

Фреймворк (Framework) — це структурований набір програмних інструментів, бібліотек, компонентів та правил. Загалом, це структура, на якій можна створювати програмне забезпечення. Він служить основою, завдяки чому не потрібно починати все з нуля.[17]

Основна мета фреймворку — надати базову архітектуру та інструменти для розробки конкретного типу програмного продукту, забезпечуючи певний рівень абстракції та готових рішень для розв'язання типових задач.

Для виконання поставленої задачі було переглянуто декілька різноманітних фреймворків різних напрямків, це:

- рушії для розробки веб-додатків, наприклад такі як Django.
- фреймворки для розробки ігор, такі як Unity, Unreal Engine, тощо.
- рушії мобільної розробки: React Native або Flutter.
- рушії для розробки десктопних додатків, такі як .NET.

Кожен з них має свої особливості та призначення, але всі вони спрощують процес розробки програмного забезпечення шляхом надання структурованого набору інструментів та рішень.

Усі вищеназвані приклади — це складні та потужні фреймворки, над якими працюють сотні людей, тому вони можуть бути занадто потужними прикладами, щоб на них орієнтуватися.

Більш простими прикладами можуть стати:

- Raylib,
- Ogre3D,

Це вже більш прості приклади, які можна використовувати у порівнянні.

Функціонал фреймворку повинен включати в себе:

1. Рендер графіки (який буде реалізований завдяки написаному власноруч рендереру).
2. Модуль Аудіо для виводу просторового звуку. Це може бути корисно для маніпулювання звуком у створених завдяки фреймворку програмах, іграх і інструментах.
3. Модуль Фізики для симуляцій фізичних об’єктів та явищ, падіння та колізії.
4. Логіка фреймворку мусить включати в себе систему компонентів сутностей (Entity component system). Це парадигма — архітектурний шаблон, який використовується у фреймворках та інших системах для організації та керування об'єктами в грі або програмі. ECS дуже часто використовується для побудови гральних двигунів та інших програм, де важливо управляти великою кількістю об'єктів з різними функціями та характеристиками.
5. Модуль скриптування надає можливість використовувати скрипти для створення логіки гри або програми без необхідності зміни вихідного коду. Цей модуль дозволяє розробникам писати код, який виконується у віртуальному середовищі під час роботи програми, що дає можливість динамічно змінювати поведінку програми без перекомпіляції або перезавантаження програми. Скриптування буде реалізовано на мові програмування Lua.
6. Графічне або візуальне скриптування буде реалізовано у вигляді нодової системи. Ця система зараз використовується у багатьох сучасних фреймворках та рушіях. У цій системі об'єкти, які взаємодіють між собою, представлені вузлами(нодами), а їх взаємозв'язки — зв'язками між цими вузлами. Нодові системи дозволяють створювати гнучкі та розширювані програми, а також спрощують розробку, тестування та збереження візуальних рішень.

Бібліотеки-субмодулі можуть бути використані для того, щоб зекономити багато часу та ще більш зменшити поріг входження у можливості фреймворку.

Для кожного з перелічених у списку пунктів буде використовуватися своя власна бібліотека.

Це зробить розробку більш зручною та гнучкою.

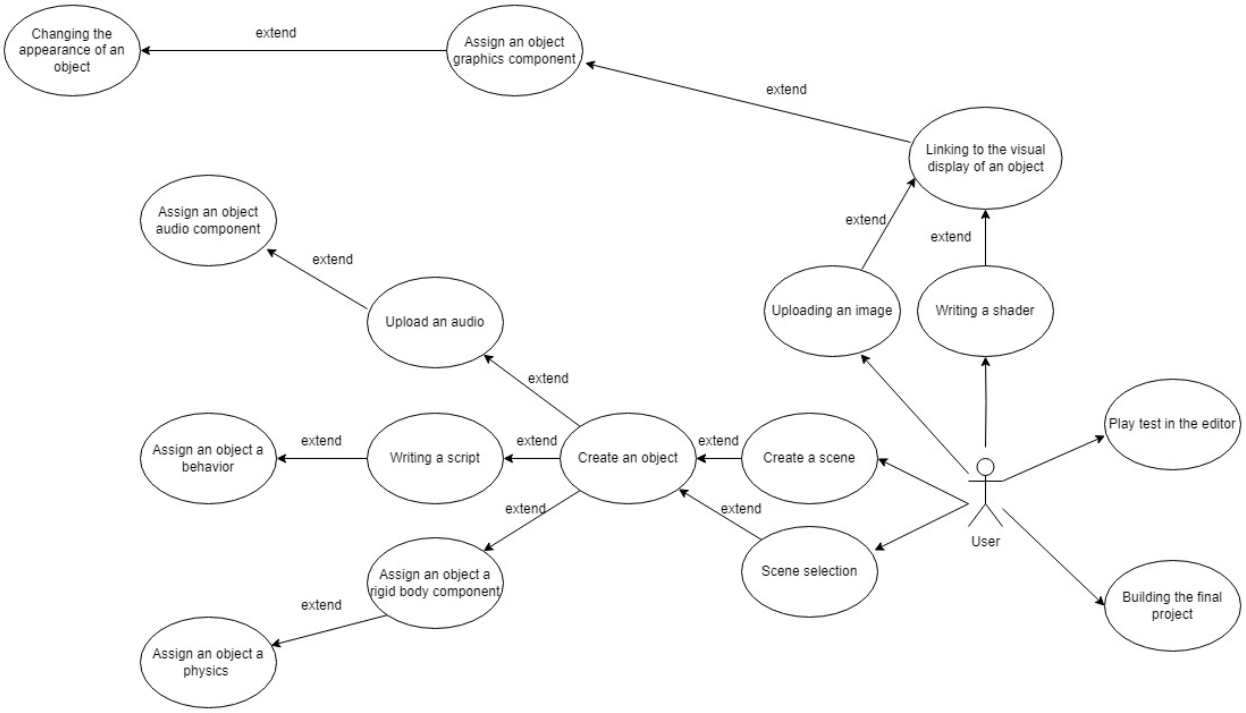

Рисунок 2.1 PixiPulse use case діаграма

2.2 Графічний/Користувацький Інтерфейс

Для створення користувацького інтерфейсу буде використовуватись бібліотека Imgui.

Dear Imgui — це проста бібліотека графічного інтерфейсу користувача для C++. Вона виводить оптимізовані буфери вершин, які можна будь-коли відобразити у програмі з підтримкою 3D-конвеєра. Вона швидка, портативна, не залежить від засобу візуалізації та автономна(без зовнішніх залежностей). [18]

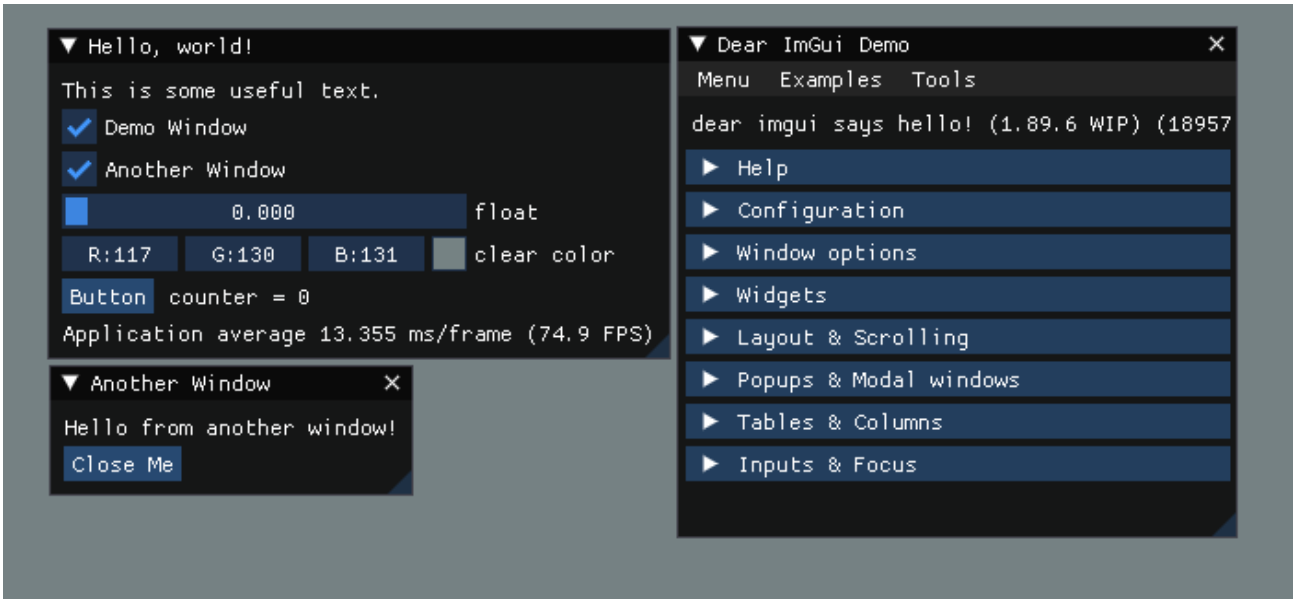

Рисунок 2.2 Типові приклади Dear ImGui

Вона працює у режимі безпосередньої взаємодії, де кожен кадр малюється окремо, а не на основі розмітки. Усі дані бібліотека зберігає у власному файлі. Таким чином, програма запам’ятовує всі зміни, які вносить користувач до інтерфейсу програми.

Ініціалізація ImGui відбувається у додатку просто вказуванням розташування вікна GUI та інших параметрів. Потім в кожному кадрі користувач викликає різні функції ImGui для створення елементів інтерфейсу, таких як кнопки, текст, поля вводу, тощо.

Однією з переваг ImGui є те, що він автоматично обробляє взаємодію користувача з інтерфейсом, включаючи натискання кнопок, переміщення миші та інше. Користувачу не потрібно писати код для цього вручну, а це в свою чергу впливає на швидкодію та гнучкість програми.

Після того, як буде побудований весь інтерфейс для поточного кадру, програма закінчує його, і ImGui відображає створений інтерфейс на екрані. Якщо стан програми змінюється, викликаються відповідні функції ImGui для оновлення інтерфейсу.

ImGui є дуже популярним та бюджетним вибором для розробки невеликого проекту у маленькій компанії або власноруч. Ця бібліотека часто

використовується для створення інструментів розробника, редакторів ресурсів та інших програм, де інтерфейс користувача є важливою складною частиною.

По завершенні роботи додатку обов’язково необхідно вивільнити ресурси, що використовує ImGui.

2.3 Модуль Аудіо для виводу просторового звуку

Miniaudio — це бібліотека для відтворення та захоплення аудіо для C/C++. Вона складається з одного вихідного файлу, не має зовнішніх залежностей та знаходиться у відкритому доступі.[19]

В першу чергу вона спрямована на просту та ефективну роботу з аудіо. Вона надає зручний інтерфейс для відтворення звуків, запису звукових потоків та інших операцій з аудіо, які дозволяють значно покращити досвід користувача.

Ініціалізація miniaudio відбувається у додатку, і зазвичай це включає створення об'єкта miniaudio(девайсу) та налаштування його параметрів, таких як формат аудіо, частота дискретизації, кількість каналів, тощо. Miniaudio надає можливість робити це просто та ефективно, не ускладнюючи код.
І вже після цього користувач може в повній мірі використовувати усі можливості цієї бібліотеки.

Miniaudio включає як низькорівневі, так і високорівневі API. Низькорівневий API буде корисним для тих, хто хоче виконувати все мікшування самостійно і потребує лише легкого інтерфейсу для основного аудіопристрою(девайсу). API високого рівня підійде ж для тих, хто має складні вимоги до мікшування та ефектів, дасть їм більше можливостей.[20]

```cpp
namespace PixiPulse {
    AudioEngine* AudioEngine::s_Instance;

    AudioEngine::AudioEngine() {
        s_Instance = this;
        ma_result result = ma_engine_init(NULL, &m_Engine);
        PP_VALIDATE(result == MA_SUCCESS);
    }

    AudioEngine::~AudioEngine() {
        ma_engine_uninit(&m_Engine);
    }


    void AudioEngine::playSound(const String& path) {
#if 0
        ma_sound* sound = getOrLoadSound(path);
        if (sound)
            ma_sound_start(sound);
#else
        ma_engine_play_sound(&m_Engine, path.c_str(), nullptr);
#endif
    }

    ma_sound* AudioEngine::getOrLoadSound(const String& path)
    {
        if (m_SoundCache.contains(path))
            return &m_SoundCache[path];

        ma_sound sound;
        ma_result result = ma_sound_init_from_file(&m_Engine, path.c_str(), 0, nullptr, nullptr, &sound);
        if (result != MA_SUCCESS)
        {
            PP_WARNING("Failed to load file {}", path);
            return nullptr;
        }

        m_SoundCache[path] = sound;
        return &m_SoundCache[path];
    }
```

Рисунок 2.3 Клас AudioEngine, що використовує високорівневий API

Однією з переваг miniaudio є те, що вона працює з різними аудіоформатами та може бути використана на різних платформах без особливих зусиль з боку розробника та користувача. Таким чином, вона може бути потужним інструментом для роботи з аудіо в вашому додатку, незалежно від його специфіки та цілей.

На завершення роботи вашого додатку, як і у випадку з ImGui, необхідно вивільнити ресурси, що використовує miniaudio, для звільнення пам'яті та інших ресурсів системи.

2.4 Модуль Фізика

Box2D — це двовимірна бібліотека моделювання фізики для ігор. Програмісти можуть використовувати його у своїх іграх, щоб змусити об’єкти рухатися реалістично та зробити ігровий світ більш інтерактивним.

Вона надає інтерфейс для створення твердих тіл, зіткнень, сил гравітації та інших фізичних ефектів та явищ.[21]

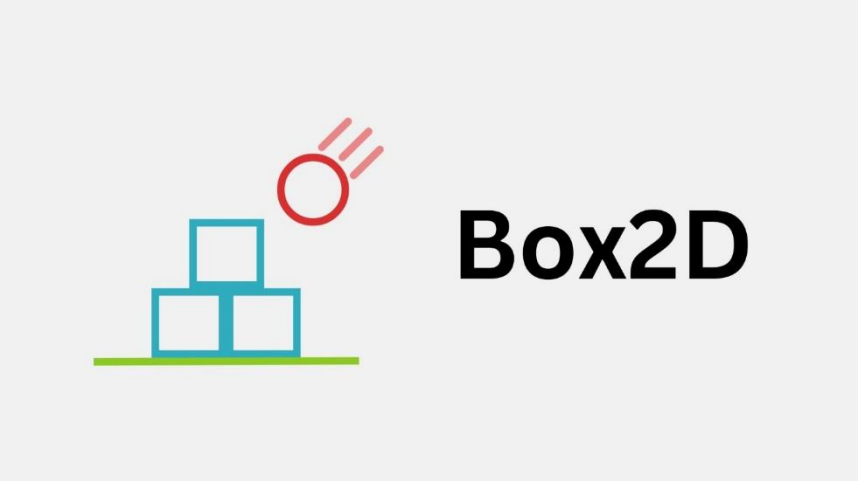

Рисунок 2.4 Бібліотека фізики Box2D

Працюючи з box2d, розробник зазвичай починає з ініціалізації світу (сцени), та вказує параметри, такі як гравітація та масштабування. Після цього він може створювати об’єкти, які представляють сутності у грі чи симуляції, і вказувати їхні властивості, такі як розміри, масу, фізичні матеріали тощо.

Box2d автоматично вирішує зіткнення між тілами, розраховує їхні реакції на сили, що діють на них, та обробляє інші аспекти фізики. Це називається колізією. Наприклад, бібліотека може виявляти зіткнення між тілами, розраховувати їхні швидкості, кутові швидкості та інше.

Однією з ключових переваг box2d є його ефективність та стабільність. Він оптимізований для роботи з великою кількістю твердих тіл у реальному часі, що дозволяє створювати складні фізичні симуляції з високою кількістю елементів.

Крім того, box2d має якісну документацію та широкий набір прикладів, що спрощує вивчення та використання бібліотеки.

2.5 Скриптування та ігрова логіка

У фреймворку буде використовуватися архітектура Entity-Component-System (ECS). І основне скриптування буде реалізовано за допомогою мови програмування Lua.

ECS — це підхід до структурування коду, де об'єкти в грі або програмі представлені як сутності (entities), які мають набір компонентів (components), що описують їхні властивості та поведінку. Він повністю виправдовує своє ім’я.

Скриптування в контексті ECS дозволяє розділити логіку гри або програми на дві частини: основну логіку, написану на мові програмування C++, та логіку, що може бути змінена або розширена, написану на мові Lua.

Один із способів реалізації скриптування у фреймворку полягає в тому, що кожна сутність та її компоненти можуть мати асоційовані скрипти Lua.

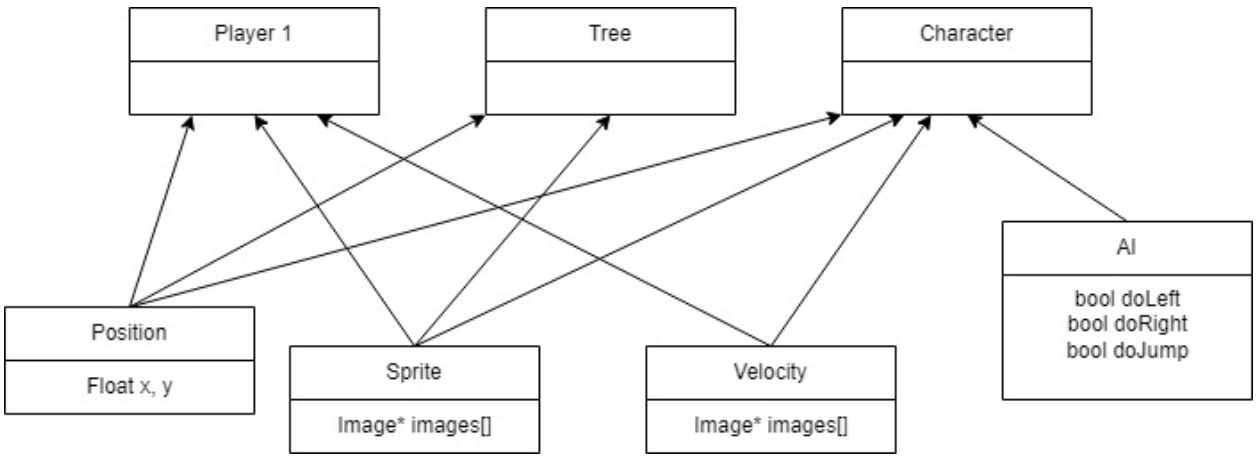

Рисунок 2.5 Простий приклад ECS

Ці скрипти визначають логіку, пов'язану з цими сутностями або компонентами. Наприклад, сутність "персонаж" може мати скрипт, що визначає його положення у просторі та штучний інтелект.

Перевагою використання Lua є його простота в інтеграції з C++ кодом, його легкість та гнучкість. Можна використовувати його для визначення складної логіки гри, наприклад штучного інтелекту сутностей, системи поведінки персонажів, обробки подій. Всі ці об’єкти можуть бути описані на Lua, при цьому зберігаючи при цьому основний код у C++.

Крім того, використання Lua дозволяє динамічно змінювати логіку гри під час виконання її програми, що полегшує тестування та розробку. Наприклад, можна змінювати параметри рівня або впливати на поведінку сутностей без перекомпіляції основного коду.

Нодова система це доволі сильне доповнення основної логіки програми. Цей підхід дозволяє розробнику створювати логіку та взаємодію об'єктів у грі або програмі шляхом з'єднання графічних вузлів, які представляють різні дії та умови.

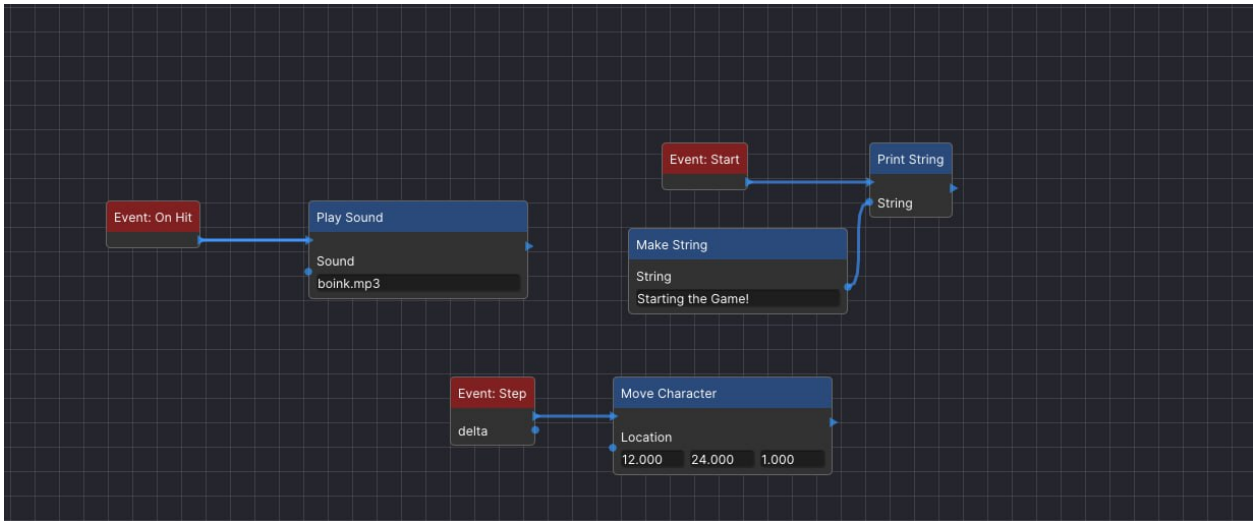

Рисунок 2.6 Зображення нодової системи у PixiPulse

Графічне скриптування може використовуватися для визначення поведінки сутностей та їх компонентів: кожен вузол представляє окрему дію або умову, яка може бути виконана або перевірена. Вузли з'єднуються між собою, утворюючи граф, який визначає порядок та умови логіки.

Наприклад, вузол "Рух вперед" може бути з'єднаний з вузлом "Перевірка колізій", і якщо колізія не виявлена, сутність буде рухатися вперед. В іншому випадку, якщо колізія виявлена, то сутність повинна зупинитися. Або може бути виконана альтернативна дія, така як зміна напрямку руху або взаємодія з іншими сутностями.

Графічне скриптування надає інтуїтивно зрозумілий спосіб визначення логіки гри чи програми, особливо для тих, хто не має глибоких знань програмування. Завдяки цьому фреймворком можуть користуватися навіть ті, кто ніяк не пов’язаний з програмуванням.

Крім того, графічне скриптування помітно полегшує візуалізацію та редагування логіки, а також сприяє швидкій розробці та тестуванню.

Однією з основних переваг графічного скриптування є його можливість інтеграції з іншими аспектами системи, такими як система візуалізації графічного інтерфейсу, система фізики чи система управління. Це дозволяє створювати повноцінні та високоефективні ігри та програми з використанням візуальних інструментів та мінімального обсягу програмування.

2.6 Оптимізація та рефактор

У міру зростання проекту відкрилося багато проблем з ефективністю та показником споживання ресурсів. Для згладжування цих проблемних моментів було реалізовано декілька модифікацій.

Однією з цих модифікацій стала імплементація так званого рендерингу пачками (batch-rendering). Його принцип напряму випливає із проблеми великої кількості викликів певних функцій, що досить сильно впливає на ефективність програми.

Під час візуалізації різних об’єктів даних найкраще організовувати дані в групи, щоб мінімізувати кількість викликів від центрального процесора до графічного процесора. Збірка, яка містить дані, які потрібно відобразити, називається пачкою (batch).[22]

Основна ідея в тому, що для різних об’єктів створюються не різні буфери, а один або декілька спільних. Насправді, це також накладає певні обмеження, але дуже сильно поліпшує продуктивність програми.

Ще однією ідеєю для оптимізації фреймворку можуть стати текстурні атласи. Вони використовуються для того, щоб не накладати на кожний об’єкт окремі текстури, а мати один спільний текстурний атлас, який складається з великої кількості текстур. Для кожного об’єкта ініціалізуються певні координати, які напряму вказують на певний фрагмент текстурного атласу, у якому і існує текстура об’єкта.

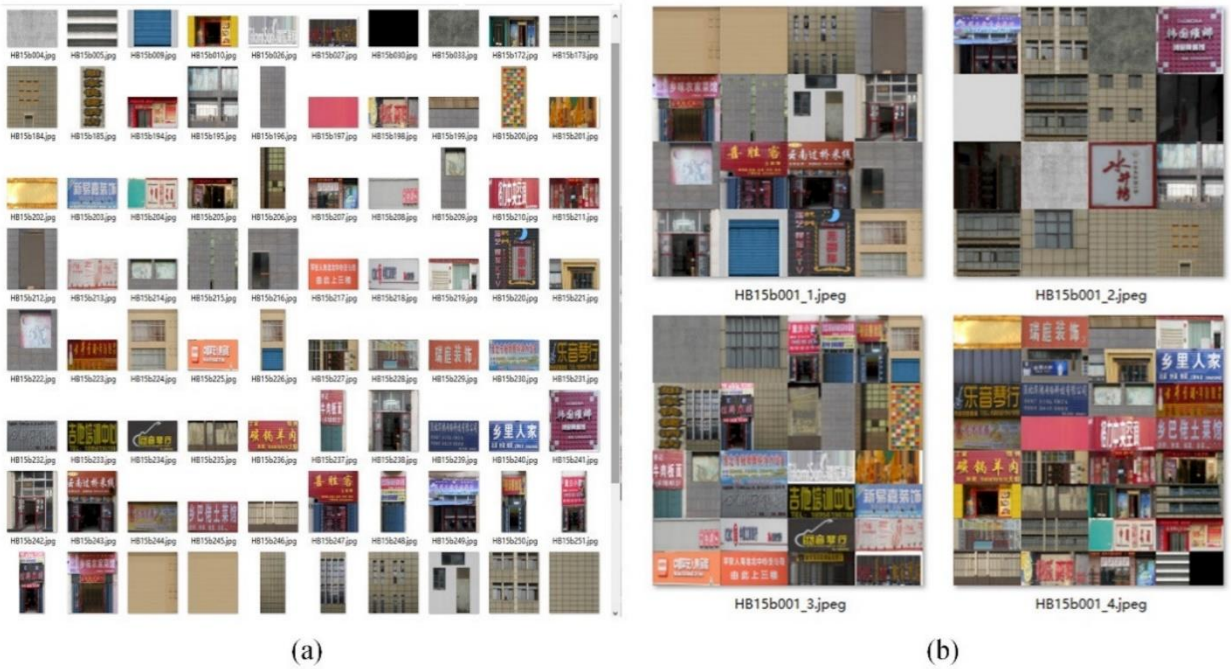

Рисунок 2.7 Приклад текстурного атласа

2.7 Тестування

Проект потребує масштабного тестування. Кожна окрема система потребує певних перевірок, які підтвердять що модуль працює саме так, як було задумано.

ВИСНОВОК

У кваліфікаційній роботі досліджено поняття фреймворку та приклади його реалізації для різних платформ та напрямків. Повністю описано процес рендеру графіки, конвеєр комп'ютерної графіки, різноманітні компоненти та елементи графічного рендеру та можливості графічного API OpenGL.

У ході роботи було розроблено простий фреймворк для створення елементарних 2D-програм, завдяки якому можна розробляти різноманітні ігри, інструменти та програми, в моменті повністю абстрагувавшись від програмування. Завдяки цьому, навіть люди, які не знають програмування на достатньому рівні зможуть реалізувати свої певні ідеї та думки.

Практичне значення результатів є вагомим. Фреймворк відповідає своїм умовам та може бути використаний для розробки ігор і програм. Можливості фреймворку максимально наближені до зазначених прикладів, зокрема Scratch.

Підсумовуючи вищевказане, можна зробити висновок, що фреймворк має потенціал для розвитку. В ньому може бути реалізовано ще багато ідей та механік. З соціальної же точки зору, він має можливість впливу на різні аудиторії через створення інклюзивного та багатогранного інструментарію розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenGL або Vulkan: найкращий вибір для графіки на мобільних пристроях? URL: https://park.zapisi.cx.ua/ukraincyam/opengl-es-abo-vulkan-naykrashhiy-vibir-dlya-grafiki-na-mobilnikh-pristroyakh.html
2. Геометричне моделювання і комп'ютерна графіка: використання OpenGL – Київський університет будівництва і архітектури. URL: https://org2.knuba.edu.ua/pluginfile.php/29517/mod_resource/content/5/OGL%20Layout.pdf
3. Рендер (Рендеринг) – що це таке у комп'ютерній графіці і 3D. URL: https://termin.in.ua/render-renderynh/
4. OpenGL Shading Language. URL: https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language
5. Buffer Object. URL: https://www.khronos.org/opengl/wiki/Buffer_Object
6. Vertex Specification. URL: https://www.khronos.org/opengl/wiki/Vertex_Specification
7. Index Buffer Object. URL: https://research.ncl.ac.uk/game/mastersdegree/graphicsforgames/indexbuffers/Tutorial%208%20-%20Index%20Buffers.pdf
8. Shader. URL: https://www.khronos.org/opengl/wiki/Shader

9. Shader Program. URL: https://yaakuro.gitbook.io/opengl-4-5/shader-program
10. Texture. URL: https://www.khronos.org/opengl/wiki/Texture#:~:text=A%20texture%20is%20an%20OpenGL,used%20as%20a%20render%20target.
11. Color buffer. URL: https://www.oreilly.com/library/view/opengl-programming-guide/9780132748445/ch04lev2sec1.html#:~:text=The%20color%20buffers%20are%20the,color%20buffers%20in%20a%20framebuffer.
12. Depth buffer. URL: https://learnopengl.com/Advanced-OpenGL/Depth-testing#:~:text=The%20depth%20buffer%20contains%20depth,frustum%27s%20near%20and%20far%20plane.
13. Framebuffer. URL: https://www.khronos.org/opengl/wiki/Framebuffer#:~:text=A%20Framebuffer%20is%20a%20collection,called%20Framebuffer%20Objects%20(FBOs).
14. OpenGL Rendering Pipeline**.** URL: https://www.geeksforgeeks.org/opengl-rendering-pipeline-overview/
15. Viewing and Transformations. URL: https://www.khronos.org/opengl/wiki/Viewing_and_Transformations#How_does_the_camera_work_in_OpenGL?
16. The model, view, and projection matrices. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/WebGL_model_view_projection
17. What is framework. URL: https://www.codecademy.com/resources/blog/what-is-a-framework/
18. Imgui. URL: https://github.com/ocornut/imgui
19. Miniaudio. URL:

https://miniaud.io/#:~:text=miniaudio%20is%20an%20audio%20playback,your%20source%20tree%20and%20go.

20. High&Low-level API for miniaudio. URL: https://miniaud.io/docs/manual/index.html
21. Box2D. URL: https://box2d.org/documentation/