# СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

# 1 ВСТУП

1.1 Мета

Мета розробки – розробка програмної системи мульти-API для 3D-рендеринга з гнучкою модульною архітектурою рушія для 3D-рендеринга з уніфікованим інтерфейсом для графічних API (DirectX, 12, Vulkan).

Метод рішення – застосування сучасних стандартів C++20, системи збирання CMake та використання патернів проектування для створення модульної архітектури з чітким розподілом відповідальності між компонентами.

Розробка програмної система спрямована на:

* вирішення проблеми фрагментації графічних технологій, зниження порогу входження для розробників;
* забезпечення високопродуктивного рендеринга незалежно від цільової платформи;
* створення доступної для новачків архітектури рушія з модульною структурою, що спрощує розуміння принципів 3D-рендеринга та уникає надмірної складності комерційних рушіїв;
* можливість легкого налаштування візуалізаційних пайплайнів, управління сценою на основі Entity Component System, та інтегрованого середовища редактора з повним набором інструментів для розробки та відлагодження.

1.2 Область застосування

Система орієнтована на використання в освітніх цілях та для розробки середньомасштабних 3D-додатків, де потрібна гнучкість та простота розуміння архітектури. Вона призначена для:

* **навчальних закладів**: як інструмент для вивчення принципів 3D-рендеринга та сучасних графічних API;
* **інді-розробників**: для створення ігор та інтерактивних додатків з низьким порогом входження;
* **прототипування**: для швидкого тестування графічних технік та експериментування з різними API;
* **дослідницьких проектів**: як база для розробки нових методів рендеринга.

Система включає інтегрований редактор з графічним інтерфейсом, що робить її доступною як для програмістів, так і для дизайнерів та художників, які працюють з 3D-контентом.

1.3 Визначення, скорочення та абревіатури

* **3D**: Three-Dimensional (тривимірний).
* **API**: Application Programming Interface (програмний інтерфейс додатку).
* **CPU**: Central Processing Unit (центральний процесор).
* **ECS**: Entity Component System (система сутність-компонент-система).
* **FPS**: Frames Per Second (кадрів на секунду).
* **glTF**: Graphics Language Transmission Format (формат передачі графічних даних).
* **GPU**: Graphics Processing Unit (графічний процесор).
* **GUI**: Graphical User Interface (графічний інтерфейс користувача).
* **HLSL**: High-Level Shading Language (високорівнева мова шейдерів).
* **PBR**: Physically-Based Rendering (фізично базований рендеринг).
* **RHI**: Render Hardware Interface (інтерфейс апаратного рендеринга).
* **SDK**: Software Development Kit (набір засобів розробки програмного забезпечення).
* **SIMD**: Single Instruction, Multiple Data (одна інструкція, множинні дані).
* **UI**: User Interface (інтерфейс користувача).

# 2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспектива продукту

Програмна система мульти-API для 3D-рендеринга розробляється як модульна освітньо-орієнтована платформа, яка виступає як навчальний інструмент та практичний рушій для розробки 3D-додатків. Система базується на багатошаровій архітектурі з абстрактним шаром Render Hardware Interface (RHI), який уніфікує операції рендеринга для різних графічних API.

Ключовою особливістю є акцент на простоті розуміння та доступності для новачків, на відміну від складних комерційних рушіїв. Система використовує сучасні стандарти C++20, систему збирання CMake та інтегрує перевірені бібліотеки (EnTT для ECS, ImGui для інтерфейсу, spdlog для логування) для забезпечення стабільності та продуктивності.

Архітектура забезпечує підтримку сучасних технік рендеринга, включаючи динамічне оновлення шейдерів, асинхронне завантаження ресурсів, різні режими візуалізації для налагодження та інтегрований редактор з інструментами для створення та модифікації 3D-сцен.

2.2 Функції продукту

**Основні функції системи рендеринга:**

* ініціалізація та управління рендеринговим контекстом через абстрактний шар RHI;
* підтримка множинних графічних API (DirectX 12, Vulkan) з уніфікованим інтерфейсом;
* система проходів рендеринга (BasePass, DebugPass, FinalPass) з підтримкою різних режимів візуалізації;
* математичні трансформації з підтримкою різних координатних систем.

**Управління ресурсами:**

* асинхронне завантаження 3D-моделей (glTF, Assimp) та текстур у фоновому потоці;
* система кешування ресурсів для запобігання повторному завантаженню;
* управління матеріалами та шейдерами з підтримкою динамічного оновлення;
* автоматичне управління пам'яттю GPU через спеціалізовані алокатори.

**Система управління сценами:**

* Entity Component System (ECS) на основі EnTT для ефективного управління об'єктами;
* cеріалізація та десеріалізація сцен у JSON формат;
* підтримка складних ієрархій об'єктів та їх взаємозв'язків.

**Інтегрований редактор:**

* графічний інтерфейс на основі ImGui з підтримкою docking-системи;
* Viewport для відображення результатів рендеринга з інтерактивними інструментами;
* інспектор об'єктів для редагування властивостей компонентів ECS;
* візуальні інструменти трансформації (gizmo) для переміщення, обертання та масштабування;
* панелі управління сценами з можливістю створення, завантаження та збереження.

**Інструменти розробки:**

* система динамічного оновлення шейдерів (hot reload) без перезапуску програми;
* інтеграція з Tracy профайлером для моніторингу CPU та GPU продуктивності;
* налагоджувальні режими візуалізації (wireframe, normal mapping, shader overdraw);
* детальна система логування з різними рівнями деталізації.

2.3 Характеристики користувачів

**Основна аудиторія:**

* **студенти та викладачі:** особи, які вивчають комп'ютерну графіку та принципи 3D-рендеринга;
* **початківці-розробники:** програмісти з базовими знаннями C++ та 3D-математики;
* **інді-розробники:** малі команди, що потребують простого та ефективного інструменту.

**Рівень знань користувачів:**

* базове розуміння програмування на C++;
* знання основ лінійної алгебри та 3D-математики;
* розуміння концепцій комп'ютерної графіки (меші, текстури, шейдери);
* бажано знайомство з системами контролю версій (Git).

**Характеристики використання:**

* переважно однокористувацький режим роботи;
* робота в середовищі розробки (Visual Studio, CLion);
* інтерактивна робота з редактором для створення та налаштування сцен;
* експериментування з різними параметрами рендеринга.

2.4 Обмеження

**Технологічні обмеження:**

* початкова підтримка лише операційної системи Windows через використання DirectX 12;
* залежність від сучасного GPU з підтримкою DirectX 12 або Vulkan;
* вимога до компілятора з підтримкою C++20 стандарту;
* обмеження архітектури на x64 платформи.

**Функціональні обмеження:**

* відсутність вбудованої підтримки мережевих функцій;
* обмежена підтримка анімації (базові трансформації);
* відсутність інтегрованої системи фізики;
* обмежена підтримка аудіо (не входить у поточну версію).

**Продуктивні обмеження:**

* орієнтація на середньостатистичне апаратне забезпечення;
* обмеження по кількості одночасно відображуваних об'єктів (до 10,000 інстансів);
* розмір текстур обмежений доступною відеопам'яттю;
* складність шейдерів обмежена цільовою частотою кадрів (60 FPS).

2.5 Припущення та залежності

**Апаратні припущення:**

* наявність сучасного GPU з підтримкою DirectX 12 або Vulkan;
* мінімум 8 GB системної пам'яті та 4 GB відеопам'яті;
* багатоядерний процесор для ефективної роботи асинхронних систем.

**Програмні залежності:**

* операційна система Windows 10 версії 1903 або новіша;
* Visual Studio 2019/2022 або еквівалентний компілятор з підтримкою C++20;
* CMake версії 3.26 або новіша для системи збирання;
* Git для роботи з зовнішніми залежностями через FetchContent.

**Припущення щодо користувачів:**

* базові знання принципів програмування та розробки ПЗ;
* доступ до сучасного апаратного забезпечення для розробки;
* готовність до навчання нових технологій та інструментів.

# 3 СПЕЦИФІЧНІ ВИМОГИ

3.1 Функціональні вимоги

3.1.1 Система абстракції графічних API

**F001 - Ініціалізація RHI:**

Система повинна забезпечувати ініціалізацію абстрактного шару Render Hardware Interface, що включає:

* створення логічного пристрою рендеринга для обраного API;
* налаштування командних черг та командних буферів;
* ініціалізацію swap chain для відображення результатів;
* створення базових ресурсів (буферів, текстур, дескрипторних куп).

**F002 - Підтримка множинних API:**

Система повинна підтримувати роботу з DirectX 12 та Vulkan через уніфікований інтерфейс:

* динамічний вибір API під час запуску через конфігураційний файл;
* ідентичність візуальних результатів незалежно від обраного API;
* автоматичне перетворення форматів даних та енумерацій між API.

**F003 - Управління ресурсами GPU:**

Система повинна надавати інтерфейси для створення та управління графічними ресурсами:

* створення та управління буферами (вершинні, індексні, константні);
* створення та управління текстурами різних форматів;
* управління шейдерами та графічними конвеєрами;
* синхронізація між CPU та GPU через fence об'єкти.

3.1.2 Система рендеринга

**F004 - Проходи рендеринга:**

Система повинна підтримувати модульну архітектуру проходів рендеринга:

* BasePass для основного рендеринга геометрії з освітленням;
* DebugPass з підтримкою різних режимів візуалізації;
* FinalPass для фінальної обробки та виведення зображення.

**F005 - Режими налагодження:**

DebugPass повинен підтримувати наступні режими візуалізації:

* Wireframe для відображення каркасної моделі;
* Normal Map Visualization для перевірки карт нормалей;
* Vertex Normal Visualization для відображення вершинних нормалей;
* Shader Overdraw для виявлення надлишкового рендеринга.

**F006 - Математичні трансформації:**

Система повинна надавати повний набір математичних операцій для 3D-рендеринга:

* трансформації координат від локального до світового простору;
* матриці виду та проекції для камер;
* підтримка різних координатних систем (лівосторонні/правосторонні).

3.1.3 Управління ресурсами

**F007 - Асинхронне завантаження:**

Система повинна підтримувати асинхронне завантаження ресурсів:

* завантаження 3D-моделей у фоновому потоці;
* Callback-сповіщення про завершення завантаження;
* індикація статусу завантаження через компоненти ECS.

**F008 - Підтримка форматів файлів:**

Система повинна підтримувати завантаження ресурсів у різних форматах:

* glTF моделі через бібліотеку Cgltf;
* додаткові 3D-формати через Assimp (OBJ, FBX та інші);
* текстури різних форматів через STB та DirectX Texture.

**F009 - Кешування ресурсів:**

Система повинна реалізувати ефективне кешування:

* запобігання повторному завантаженню ідентичних ресурсів;
* управління пам'яттю з автоматичним вивільненням невикористовуваних ресурсів;
* потокобезпечний доступ до кешу ресурсів.

3.1.4 Система управління сценами

**F010 - Entity Component System:**

Система повинна реалізувати архітектуру ECS на основі EnTT:

* створення та управління сутностями;
* додавання та видалення компонентів динамічно;
* ефективна обробка великих наборів сутностей через системи;

**F011 - Серіалізація сцен:**

Система повинна підтримувати збереження та завантаження сцен:

* серіалізація у JSON формат для читабельності;
* збереження повного стану сцени включно з компонентами;
* валідація структури даних при завантаженні.

**F012 - Управління множинними сценами:**

Система повинна надавати можливості управління кількома сценами:

* кешування завантажених сцен у пам'яті;
* швидке переключення між сценами;
* ізоляція ресурсів між різними сценами.

3.1.5 Інтегрований редактор

**F013 - Графічний інтерфейс:**

Редактор повинен надавати повнофункціональний графічний інтерфейс:

* Viewport для відображення результатів рендеринга;
* панель ієрархії сцени з можливістю вибору об'єктів;
* інспектор властивостей для редагування компонентів.

**F014 - Інструменти трансформації:**

Редактор повинен включати візуальні інструменти для редагування:

* Gizmo для переміщення, обертання та масштабування об'єктів;
* підтримка різних координатних систем (світові/локальні);
* інтеграція з системою вводу для інтерактивного редагування.

**F015 - Управління сценами в редакторі:**

Редактор повинен надавати інструменти для роботи зі сценами:

* створення нових сцен з шаблонними налаштуваннями;
* збереження та завантаження сцен через графічний інтерфейс;
* перегляд та редагування метаданих сцени.

3.1.6 Інструменти розробки

**F016 - Динамічне оновлення шейдерів:**

Система повинна підтримувати hot reload функціональність:

* автоматичне відстеження змін у файлах шейдерів;
* перекомпіляція та застосування змін без перезапуску;
* обробка помилок компіляції з відкатом до попередньої версії.

**F017 - Профілювання продуктивності:**

Система повинна інтегрувати інструменти профілювання:

* інтеграція з Tracy профайлером для CPU та GPU метрик;
* відображення метрик продуктивності в реальному часі;
* візуалізація навантаження на різні підсистеми.

**F018 - Система логування:**

Система повинна забезпечувати детальне логування:

* різні рівні деталізації (debug, info, warning, error);
* логування всіх критичних операцій рендеринга;
* інтеграція з редактором для відображення логів.

3.2 Нефункціональні вимоги

3.2.1 Вимоги до продуктивності

**P001 - Частота кадрів:**

Система повинна забезпечувати стабільну частоту кадрів не менше 60 FPS при рендерингу сцен середньої складності (до 1,000 об'єктів) на GPU середнього рівня (наприклад, NVIDIA GTX 1660 або еквівалент).

**P002 - Час завантаження:**

Час завантаження 3D-моделей не повинен перевищувати:

* 1 секунда для файлів розміром до 10 MB;
* 3 секунди для файлів розміром до 50 MB;
* 10 секунд для файлів розміром до 200 MB.

**P003 - Використання пам'яті:**

Система повинна ефективно використовувати доступну пам'ять:

* максимальне використання системної пам'яті: 2 GB для базової сцени;
* максимальне використання відеопам'яті: 1 GB для базової сцени;
* автоматичне вивільнення невикористовуваних ресурсів.

3.2.2 Вимоги до надійності

**R001 - Стабільність роботи:**

Система повинна працювати стабільно протягом тривалих сеансів:

* безперервна робота протягом 8 годин без критичних збоїв;
* корректна обробка помилок з відповідними повідомленнями;
* автоматичне відновлення після некритичних помилок.

**R002 - Управління пам'яттю:**

Система повинна забезпечувати коректне управління пам'яттю:

* відсутність витоків пам'яті при тривалій роботі;
* використання RAII принципів для автоматичного управління ресурсами;
* правильна синхронізація між CPU та GPU для уникнення гонок даних.

3.2.3 Вимоги до зручності використання

**U001 - Інтуїтивність інтерфейсу:**

Редактор повинен надавати інтуїтивний інтерфейс:

* логічне розташування панелей та інструментів;
* підтримка drag-and-drop операцій де це доречно;
* контекстні меню з релевантними опціями.

**U002 - Час навчання:**

Новий користувач з базовими знаннями 3D-графіки повинен освоїти основні функції системи протягом 4-6 годин роботи з документацією та прикладами.

**U003 - Доступність функцій:**

Всі основні функції повинні бути доступні не більше ніж за 3 кліки або за допомогою клавіатурних комбінацій.

3.2.4 Вимоги до підтримуваності

**M001 - Модульність архітектури:**

* система повинна мати модульну структуру:
* чітке розділення відповідальностей між компонентами;
* можливість заміни окремих модулів без впливу на інші;
* використання добре відомих патернів проектування.

**M002 - Документованість коду:**

Код повинен бути добре документований:

* коментарі для всіх публічних інтерфейсів;
* документація архітектурних рішень;
* приклади використання для ключових компонентів.

3.3 Обмеження проєктування

3.3.1 Технологічні обмеження

**C001 - Мова програмування:**

Система повинна бути реалізована на мові C++ зі стандартом C++20, що обмежує сумісність з застарілими компіляторами.

**C002 - Графічні API:**

Початкова підтримка обмежена DirectX 12 та Vulkan, що виключає підтримку застарілих API (DirectX 11, OpenGL).

**C003 - Платформна залежність:**

Система орієнтована на Windows платформу, що обмежує кросплатформну сумісність у початковій версії.

3.3.2 Ресурсні обмеження

**C004 - Апаратні вимоги:**

Система вимагає сучасного апаратного забезпечення:

* GPU з підтримкою DirectX 12 або Vulkan;
* мінімум 8 GB системної пам'яті;
* мінімум 4 GB відеопам'яті.

**C005 - Розмір проекту:**

Розмір кодової бази обмежений можливостями підтримки одним розробником, що впливає на масштаб функціональності.

3.4 Атрибути якості

3.4.1 Розширюваність

Система спроектована з урахуванням майбутніх розширень:

* можливість додавання підтримки нових графічних API;
* розширення функціональності редактора через плагін-систему;
* додавання нових типів компонентів ECS без модифікації ядра.

3.4.2 Тестованість

Архітектура системи забезпечує можливість тестування:

* ізоляція компонентів для unit-тестування;
* можливість mock-інг зовнішніх залежностей;
* інтеграційне тестування через автоматизовані сценарії.

3.4.3 Портативність

Система розроблена з урахуванням майбутньої портативності:

* абстракція платформно-залежного коду;
* використання кросплатформних бібліотек де можливо;
* підготовка до підтримки інших операційних систем.

# 4 ПЕРЕВІРКА ТА ВАЛІДАЦІЯ

4.1 Плани тестування

4.1.1 Функціональне тестування

**Тестування системи рендеринга:**

* перевірка коректної ініціалізації обох графічних API (DirectX 12, Vulkan);
* валідація ідентичності візуальних результатів між різними API;
* тестування всіх режимів рендеринга (базовий, налагоджувальні режими);
* перевірка коректності трансформацій координат та роботи камер.

**Тестування управління ресурсами:**

* перевірка завантаження моделей у різних форматах (glTF, OBJ, FBX);
* тестування асинхронного завантаження та callback-механізмів;
* валідація роботи системи кешування ресурсів;
* перевірка коректності вивільнення пам'яті.

**Тестування редактора:**

* функціональність всіх панелей інтерфейсу;
* робота інструментів трансформації (gizmo);
* збереження та завантаження сцен;
* перемикання між різними режимами візуалізації.

4.1.2 Тестування продуктивності

**Навантажувальне тестування:**

* рендеринг сцен з 100, 1,000 та 5,000 об'єктів;
* вимірювання FPS при різних навантаженнях;
* профілювання використання CPU та GPU ресурсів;
* тестування з різними розмірами вікна та роздільністю.

**Тестування пам'яті:**

* моніторинг використання системної та відеопам'яті;
* виявлення витоків пам'яті при тривалій роботі;
* тестування завантаження та вивантаження великих ресурсів.

4.1.3 Тестування стабільності

**Тривале тестування:**

* безперервна робота протягом 8-24 годин;
* циклічне завантаження та вивантаження ресурсів;
* стрес-тестування з максимальним навантаженням.

**Тестування обробки помилок:**

* завантаження некоректних файлів;
* імітація помилок графічних драйверів;
* тестування при недостатніх ресурсах пам'яті.

4.1.4 Інструменти тестування

**Інструменти аналізу:**

* RenderDoc для аналізу графічних команд;
* Tracy профайлер для аналізу продуктивності;
* валідаційні шари графічних API для виявлення помилок.

4.2 Критерії приймання

4.2.1 Функціональні критерії

**Система рендеринга:**

* успішна ініціалізація обох графічних API без помилок;
* ідентичність візуальних результатів при перемиканні між API (різниця менше 1%);
* коректна робота всіх режимів візуалізації;
* підтримка базових математичних трансформацій з точністю float32.

**Управління ресурсами:**

* завантаження моделей у форматах glTF та OBJ без помилок;
* асинхронне завантаження з коректними callback-сповіщеннями;
* ефективність кешування (зменшення часу повторного завантаження на 90%).

**Редактор:**

* всі панелі інтерфейсу функціонують коректно;
* збереження та завантаження сцен без втрати даних;
* інструменти трансформації працюють з точністю до 0.01 одиниці.

4.2.2 Критерії продуктивності

**Частота кадрів:**

* стабільні 60 FPS для сцени з 1,000 об'єктами на цільовому GPU;
* не менше 30 FPS для сцени з 5,000 об'єктами.

**Час завантаження:**

* моделі до 10 MB завантажуються за менше ніж 1 секунду;
* ініціалізація системи за менше ніж 5 секунд;
* переключення між сценами за менше ніж 2 секунди.

**Використання ресурсів:**

* базове споживання пам'яті менше 500 MB;
* відсутність витоків пам'яті при 8-годинному тестуванні;
* ефективне використання відеопам'яті (не більше 1 GB для базової сцени).

4.2.3 Критерії стабільності

**Надійність:**

* безперервна робота протягом 8 годин без критичних збоїв;
* коректна обробка всіх типів помилок з відповідними повідомленнями;
* автоматичне відновлення після некритичних помилок.

**Якість коду:**

* відсутність попереджень компілятора на максимальному рівні;
* успішне проходження статичного аналізу коду.

4.2.4 Критерії зручності використання

**Інтерфейс користувача:**

* інтуїтивність інтерфейсу (оцінка користувачами 4/5 або вище);
* доступність основних функцій не більше ніж за 3 кліки;
* наявність підказок та документації для всіх функцій.

**Навчання:**

* новий користувач освоює базові функції за 4-6 годин;
* успішне виконання всіх навчальних сценаріїв.

Система вважається готовою до використання лише при виконанні всіх зазначених критеріїв приймання та успішному проходженні всіх етапів тестування.