МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ТЕХНОЛОГІЯ РЕНДЕРИНГУ 3D-СЦЕН ІЗ ВИКОРИСТАННЯМ КАРТ ГЛИБИНИ ТА ГЕНЕРАТИВНИХ НЕЙРОННИХ МЕРЕЖ

Виконав: студент групи 2П-22
Спеціальності
121 Інженерія програмного забезпечення
Владислав КУЦИЙ
Керівник:
Станіслав МАРЧЕНКО

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувачка циклової комісії КІ та ІТ

_____ Наталя ФАЛЬЧЕНКО
(підпис)
«_____» _____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Куцому Владиславу Володимировичу

1. Тема кваліфікаційної роботи Технологія рендерингу 3D-сцен із використанням карт глибини та генеративних нейронних мереж

Керівник роботи Марченко Станіслав Віталійович, викладач першої категорії

затверджені наказом закладу фахової передвищої освіти від «06» жовтня 2025 року № 84/1-у.

2. Строк подання студентом кваліфікаційної роботи 02.06.2026

3. Вихідні дані до кваліфікаційної роботи: середовище 3D-моделювання Blender; мова програмування Python; Blender Python API; вебфреймворк FastAPI; хмарна платформа Google Cloud Platform; сервіс Google Cloud Run; Google Gemini API для роботи з мультимодальними генеративними моделями; Google OAuth 2.0 для автентифікації користувачів; FastSpring для інтеграції платіжної системи; SonarCloud для статичного аналізу якості та безпеки коду; GitHub як система керування версіями та розміщення репозиторію; технології REST API, JSON, JWT, CI/CD; засоби генерації та обробки карт глибини, Mist Pass, Smart Points, Inpaint Editor і Smart Restore.

4. Зміст кваліфікаційної роботи: аналіз предметної області та методів інтелектуальної візуалізації 3D-сцен (характеристика сучасних технологій візуалізації; еволюція методів нейронного рендерингу; аналіз наявних програмних рішень та їх технологічних обмежень; обґрунтування вибору технологій та постановка задачі»); проєктування та реалізація програмного

забезпечення (концептуальне моделювання та специфікація вимог до системи; попередня архітектура програмного забезпечення; імплементація клієнтської частини та інтеграція рушія рендерингу); тестування та розгортання програмного продукту (статичний аналіз та аудит безпеки коду (SonarCloud); автоматизація процесів постачання програмного забезпечення»; організація та проведення закритого бета-тестування; апробація результатів розроблення).

5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	Розділ 1. Аналіз предметної області та методів інтелектуальної візуалізації 3D-сцен	09.12.2025	
3	Розділ 2. Проєктування та реалізація програмного забезпечення	10.03.2026	
4	Розділ 3. Тестування та розгортання програмного продукту	28.04.2026	
5	Висновки	12.05.2026	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2026	
8	Подання кваліфікаційної роботи на затвердження завідувачці циклової комісії (за 7 днів до захисту)	10.06.2026	

Студент _____ Владислав КУЦИЙ
(підпис)

Керівник роботи _____ Станіслав МАРЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційну роботу присвячено розв’язанню критичної проблеми обчислювальної складності традиційного 3D-рендерінгу, що створює високий апаратний та економічний бар’єр для широкого кола користувачів. Головним результатом роботи є розробка та впровадження інтелектуального програмного комплексу «Nano Banana Render (Nanode)» для середовища Blender.

У роботі реалізовано гібридний метод нейронного рендерингу, який поєднує точність структурних даних 3D-сцени із генеративною потужністю хмарної мультимодальної моделі Google Gemini. Для забезпечення високої швидкості та нативної якості візуалізації спроєктовано надійну мікросервісну клієнт-серверну архітектуру на базі FastAPI та Google Cloud Run, що забезпечує автоматичне масштабування та нівелює залежність від локальної відеопам’яті користувача.

Програмний модуль інтегровано як повноцінний нативний рушій рендерингу Blender з інноваційним алгоритмом випереджального захоплення кадру, що гарантує стабільність інтерфейсу під час асинхронних мережевих запитів. Функціонал включає: алгоритм Smart Points для точного просторового керування генерацією об’єктів; редактор для локального редагування текстур безпосередньо у вікні перегляду та систему управління історією.

Особливу увагу приділено розробці комплексної інфраструктури: впроваджено безшовну автентифікацію Google OAuth 2.0, інтеграцію платіжної системи FastSpring (Merchant of Record) та механізм захисту API-квот через блокування за апаратними ідентифікаторами (HWID). Якість і безпека коду верифіковані статичним аналізом SonarCloud з найвищою оцінкою "A".

Проведений аудит підтвердив можливість генерації фотореалістичних зображень у роздільній здатності Native 4K лише за 30 секунд. Результати роботи мають високу практичну цінність, підтверджену експертним визнанням та отриманням інфраструктурного гранту Google for Startups Cloud Program.

Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, РЕНДЕРИНГ, BLENDER, PYTHON, ГЕНЕРАТИВНІ МОДЕЛІ, COMPUTER GRAPHICS, API, MIST PASS.

ANNOTATION

This thesis addresses the critical issue of computational complexity in traditional 3D rendering, which creates significant hardware and economic barriers for a wide range of users. The primary outcome of this research is the development and implementation of an intelligent software suite, "Nano Banana Render (Nanode)," designed for the Blender environment.

The project introduces a hybrid neural rendering method that merges the precision of 3D scene structural data with the generative capabilities of the cloud-based Google Gemini multimodal model. To ensure high speed and native render quality, a robust microservice client-server architecture was engineered using FastAPI and Google Cloud Run. This architecture provides automatic scaling and completely eliminates the reliance on the user’s local video memory.

The software module is integrated as a fully fledged native Blender rendering engine. It utilizes an innovative pre-emptive frame capture algorithm, ensuring interface stability during asynchronous network requests. The core features include the Smart Points algorithm for precise spatial control over object generation, an Editor for local texture adjustments directly within the viewport, and a comprehensive history management system.

Significant emphasis was placed on developing a complete infrastructure. This includes the implementation of seamless Google OAuth 2.0 authentication, the integration of the FastSpring payment gateway (Merchant of Record), and an API quota protection mechanism utilizing Hardware ID (HWID) blocking. The code’s quality and security have been validated through SonarCloud static analysis, achieving the highest "A" rating.

The conducted audit confirmed the system’s ability to generate photorealistic images in Native 4K resolution in just 30 seconds. The project’s results demonstrate high practical value, validated by expert recognition and the award of an infrastructure grant from the Google for Startups Cloud Program.

Keywords: ARTIFICIAL INTELLIGENCE, RENDERING, BLENDER, PYTHON, GENERATIVE MODELS, COMPUTER GRAPHICS, API, MIST PASS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

API	Application Programming Interface (Програмний інтерфейс застосунку)
bpy	Blender Python API (Програмний інтерфейс Blender на Python)
CAGR	Compound Annual Growth Rate (Середньорічний темп приросту)
CGI	Computer-Generated Imagery (Комп’ютерна графіка)
CI/CD	Continuous Integration / Continuous Deployment (Безперервна інтеграція та доставка)
FastAPI	Високопродуктивний асинхронний веб-фреймворк
GCP	Google Cloud Platform (Хмарна платформа Google)
HWID	Hardware ID (Апаратний ідентифікатор)
JSON	JavaScript Object Notation (Формат обміну даними)
JWT	JSON Web Token (Захищений маркер доступу)
LDM	Latent Diffusion Models (Латентні дифузійні моделі)
LLM	Large Language Model (Велика мовна модель)
PBR	Physically Based Rendering (Фізично коректний рендеринг)
RAM	Random Access Memory (Оперативна пам’ять)
REST API	Representational State Transfer Application Programming Interface
SDK	Software Development Kit (Комплект для розробки програмного забезпечення)
SOA	Service-Oriented Architecture (Сервіс-орієнтована архітектура)
SPA	Single-Page Application (Односторінковий застосунок)
UML	Unified Modeling Language (Уніфікована мова моделювання)
UX	User Experience (Користувацький досвід)
VAE	Variational Autoencoder (Варіаційний автоенкодер)
ViT	Vision Transformer (Архітектура візуального трансформера)
VRAM	Video Random Access Memory (Відеопам’ять)

ВСТУП

Сучасна індустрія комп’ютерної графіки та створення цифрового контенту переживає фундаментальні зміни, пов’язані з інтеграцією методів штучного інтелекту. Традиційні алгоритми рендерингу, такі як трасування шляхів, що використовуються у промислових стандартах (Cycles, V-Ray, Arnold), забезпечують високу фізичну точність, проте характеризуються експоненційною обчислювальною складністю. Згідно з останніми аналітичними звітами [1; 2], ринок 3D-рендерінгу оцінюється у 4,5-5,6 млрд доларів США і демонструє середньорічний темп приросту (CAGR) на рівні 18–20%, що зумовлено попитом у сферах архітектури, ігор та кіновиробництва. При цьому час, необхідний для налаштування сцени та обчислення фінального зображення, залишається критичним «вузьким місцем», поглинаючи до 40-50% часового бюджету проєктів.

Поява генеративних нейронних мереж відкрила нові можливості для прискорення візуалізації. Проте, наявні рішення на базі дифузійних моделей (Stable Diffusion) часто мають високі вимоги до апаратного забезпечення (зокрема, обсягу відеопам’яті) та страждають від проблеми «галюцинацій», коли згенероване зображення не відповідає геометрії сцени. Це створює нагальну потребу в розробці гібридних програмних інструментів, які поєднують точність 3D-геометрії з генеративною потужністю сучасних мультимодальних моделей (таких як Gemini 3 Pro), забезпечуючи високу швидкість та контроль результату [3-7].

Об’єктом дослідження є процеси генерації фотореалістичних зображень у системах тривимірного моделювання.

Предметом дослідження виступають методи, технології та інструментальні засоби програмної реалізації гібридного рендерингу з використанням мультимодальних нейронних мереж та карт глибини.

Метою роботи є підвищення ефективності та швидкодії процесу візуалізації тривимірних сцен шляхом розробки програмного забезпечення для

середовища Blender, що реалізує метод нейронного рендерингу на основі карт глибини. Для досягнення мети необхідно вирішити такі завдання:

1) здійснити системний аналіз наявних методів рендерингу та алгоритмів генеративного штучного інтелекту, визначити їхні переваги та недоліки;

2) розробити мікросервісну архітектуру на базі FastAPI та Google Cloud Run [8; 9], імплементувавши розширення (add-on) як нативний рушій рендерингу з алгоритмом випереджального захоплення кадру (pre-capture) для неблокуючої взаємодії;

3) створити алгоритми обробки структурних даних 3D-сцени (Mist Pass) та розробити механізм просторового керування Smart Points для точного позиціонування об’єктів і забезпечення пріоритету геометричної структури;

4) імплементувати інструменти інтерактивного пост-оброблення, включаючи Inpaint Editor для локального редагування та систему управління історією Smart Restore для гнучкого контролю версій;

5) розробити комплексну систему безпеки та адміністрування: безшовну автентифікацію (Google OAuth 2.0 [10]), захист API-квот через HWID-блокування та інтеграцію платіжної системи FastSpring [11];

6) провести верифікацію якості коду через статичний аналіз та імплементувати асинхронну систему збору відгуків користувачів (RLHF-телеметрію) для оптимізації параметрів генерації.

У роботі дістало подальшого розвитку застосування мультимодальних трансформерних моделей у пайплайні 3D-рендерингу, що, на відміну від традиційних дифузійних підходів, дозволяє використовувати семантичне розуміння сцени для більш точного текстурування без необхідності локального розгортання великовагових нейромереж. Розроблюваний програмний продукт «Nano Banana Pro Render» надаватиме змогу зменшити час отримання ескізних візуалізацій з десятків хвилин до секунд, знижуючи поріг входження для новачків та вимоги до апаратного забезпечення.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ІНТЕЛЕКТУАЛЬНОЇ ВІЗУАЛІЗАЦІЇ 3D-СЦЕН

1.1 Характеристика сучасних технологій візуалізації

Сучасна індустрія комп'ютерної графіки (Computer-Generated Imagery, CGI) перебуває на етапі фундаментальної технологічної трансформації, яка характеризується переходом від детермінованих алгоритмів розрахунку освітлення до імовірнісних методів генеративного штучного інтелекту. Традиційні методи візуалізації, що формували галузь протягом останніх трьох десятиліть, досягли певної межі ефективності, де подальше підвищення якості зображення вимагає експоненційного зростання обчислювальних потужностей.

Згідно з аналітичним звітом Grand View Research [1], глобальний ринок програмного забезпечення для 3D-рендерингу оцінюється у 4,21 млрд доларів США. За прогнозами аналітиків, цей показник досягне 19,82 млрд доларів США до 2033 року, демонструючи вражаючий середньорічний темп приросту на рівні 19,2%. Таке експоненційне зростання зумовлене експансією 3D-контенту в сфери архітектурної візуалізації, ігрової індустрії (GameDev), кіновиробництва (VFX) та віртуальної реальності (VR/AR). Наочно динаміку зростання ринку та попиту на технології візуалізації представлено на рис. 1.1. Однак, незважаючи на фінансове зростання, технічний бік процесу залишається обтяженим низкою критичних проблем.

Основним стандартом фотореалістичної візуалізації сьогодні є метод трасування шляхів (Path Tracing), який використовується у таких рушіях як Cycles (Blender), Arnold (Autodesk) та V-Ray (Chaos Group). Документація Blender визначає Cycles як фізично коректний трасувальник шляхів для production-рендерингу [12]. Математично цей метод базується на чисельному розв'язанні рівняння рендерингу (Rendering Equation), сформульованого Джеймсом Каджією (James Kajiya) у 1986 році [13].

Рівняння описує закон збереження енергії при поширенні світла:

$$L_o(x, w_0) = L_e(x, w_o) + \int_{\Omega} f_r(x, w_i, w_o) L_i(x, w_i)(w_i \cdot n) dw_i, \quad (1)$$

де L_o – вихідна енергетична яскравість у напрямку спостерігача, L_e – власне випромінювання поверхні (для джерел світла); Ω – півсфера напрямків падіння світла; f_r – двонапрямлена функція розподілу відбиття (BRDF). Фізична коректність цього рівняння гарантує високий реалізм, проте обчислювальна складність алгоритму зростає лінійно зі збільшенням кількості джерел світла та геометричної складності сцени. Для отримання зображення, вільного від стохастичного шуму, необхідно розрахувати тисячі світлових шляхів (семплів) для кожного пікселя. Це необхідно для коректного моделювання складних ефектів глобального освітлення, таких як м’які тіні, каустика та підповерхневе розсіювання (Subsurface Scattering).

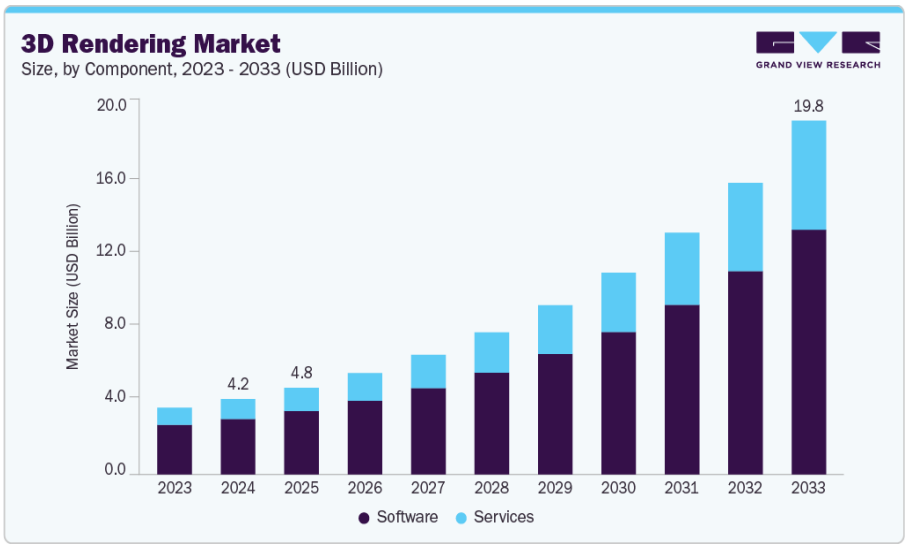


Рисунок 1.1 – Прогноз зростання ринку 3D-рендерінгу до 2033 року за даними Grand View Research [1]

Емпіричні дослідження продуктивності сучасних рендер-рушіїв свідчать, що рендеринг одного кадру у роздільній здатності 4K (3840x2160) на споживчому обладнанні може тривати від 20 хвилин до кількох годин. Особливо критичним цей фактор стає у контексті анімації, де одна секунда відео зазвичай складається з 24-60 кадрів. У таких умовах сумарні витрати часу на рендеринг

навіть короткого ролика зростають експоненційно, перетворюючись на тижні машинного часу.

Згідно зі звітом Grand View Research [1], оптимізація часу рендерингу та зниження витрат на обчислювальні потужності є ключовим технічним викликом для 68% великих профільних підприємств. Узагальнюючи викладене, у класичному підході до візуалізації можна виділити комплекс фундаментальних проблем, що суттєво стримують розвиток індустрії. Насамперед, критичним фактором є часова складність алгоритмів. Надмірно високі вимоги до часу обчислень фактично блокують можливість швидких ітерацій дизайну. Художник змушений перебувати в режимі очікування результату рендерингу після кожної, навіть незначної зміни параметрів сцени, що створює ефект так званого «вузького місця» (bottleneck) у виробничому пайплайні.

Окрім часових обмежень, існує високий кваліфікаційний бар’єр. Процес налаштування фізично коректних матеріалів (PBR shading) вимагає від користувача не лише художнього смаку, а й глибокої технічної експертизи у фізиці поширення світла. Зокрема, необхідним є точне оперування такими параметрами, як коефіцієнт заломлення (Index of Refraction) та підповерхневе розсіювання, що значно ускладнює вхід у професію для дизайнерів-початківців та вимагає тривалого навчання.

Якісний рендеринг традиційними методами (Path Tracing) висуває жорсткі, часто надмірні вимоги до апаратного забезпечення користувача. Намагання досягти фотореалізму змушує художників використовувати текстури надвисокої роздільної здатності (4K/8K UDIMs), складну геометрію з мікродеталізацією та важкі об’ємні ефекти (volumetrics).

Серед усіх апаратних ресурсів найбільш критичним є відеопам’ять та швидкість доступу до даних.

Проблема обсягу текстурних даних: паралелі з ігровою індустрією Масштаб проблеми можна проілюструвати на прикладі сучасної ігрової індустрії. Середній розмір AAA-ігор в останні роки, наприклад, Call of Duty, Monster Hunter Wilds, Final Fantasy VII Rebirth часто перевищує 150 ГБ [14].

Левову частку цього обсягу займають саме текстури високої роздільної здатності.

Однак, у сфері продакшн-рендерингу (Blender Cycles) ситуація є значно складнішою. Якщо в іграх використовуються алгоритми компресії (BC7/DDS), то рендер-рушії часто завантажують текстури у відеопам’ять у розпакованому вигляді для уникнення артефактів та збереження точності фізичних властивостей матеріалів [15]. Обсяг відеопам’яті, необхідний для збереження текстурних даних, зростає експоненційно залежно від їхньої роздільної здатності. Емпіричні дані свідчать, що один набір текстур формату 4K (який включає карти кольору, шорсткості, нормалей та зміщення) займає близько 256 МБ відеопам’яті. Використання ж формату 8K збільшує цей показник до 1 ГБ лише для одного матеріалу. Наочно залежність споживання відеопам’яті від роздільної здатності текстур наведено в таблиці 1.1. Як видно з таблиці, використання 8K текстур (що є стандартом для великих планів) дозволяє завантажити у пам’ять популярної відеокарти (8 ГБ) лише 6-7 матеріалів. Будь-яка сцена складніша за «тестовий куб» миттєво вичерпує цей ліміт.

Таблиця 1.1 – Орієнтовне споживання відеопам’яті одним сетом текстур (PBR Material) [15]

Роздільна здатність	Кількість пікселів	Розмір у VRAM (1 PBR сет)	Кількість матеріалів до переповнення 8 ГБ VRAM
2K (2048x2048)	4 млн	~64 МБ	~100–120
4K (4096x4096)	16 млн	~256 МБ	~25–30
8K (8192x8192)	64 млн	~1024 МБ (1 ГБ)	6–7

Коли ліміт відеопам’яті перевищено, рендер-рушій змушений переходити в режим Out-of-core rendering. У цьому режимі дані вивантажуються у системну оперативну пам’ять (RAM). Хоча обсяг RAM може бути великим (64–128 ГБ), її пропускна здатність критично поступається відеопам’яті. Порівняльний аналіз пропускної здатності пам’яті демонструє критичну різницю між спеціалізованою відеопам’яттю та системною оперативною пам’яттю. Зокрема, швидкість обміну

даними VRAM стандарту GDDR6X може досягати 1008 ГБ/с, тоді як пропускна здатність сучасної системної пам’яті DDR5 становить лише близько 60–80 ГБ/с. Така різниця у швидкості (у 12-15 разів) призводить до суттєвого падіння продуктивності при виході за межі доступного обсягу відеопам’яті. Це падіння швидкості доступу до даних призводить до того, що час рендерингу збільшується експоненційно, а стабільність роботи драйвера знижується [16].

Окрім технічних обмежень, у 2025-2026 роках виник новий критичний бар’єр – різке зростання вартості пам’яті через бурхливий розвиток індустрії штучного інтелекту. Навчання великих мовних моделей (LLM) потребує колосальних обсягів швидкої пам’яті. Через це провідні виробники чіпів (Samsung, SK Hynix, Micron) масово перепрофілювали виробничі лінії з випуску споживчої пам’яті на серверну пам’ять для дата-центрів. Це спричинило штучний дефіцит на ринку споживчої електроніки. Згідно з аналітикою Cybernews та галузеві огляди 2025 року (рис. 1.2) фіксують суттєве подорожчання модулів пам’яті та SSD на тлі високого попиту з боку інфраструктури ШІ [17].

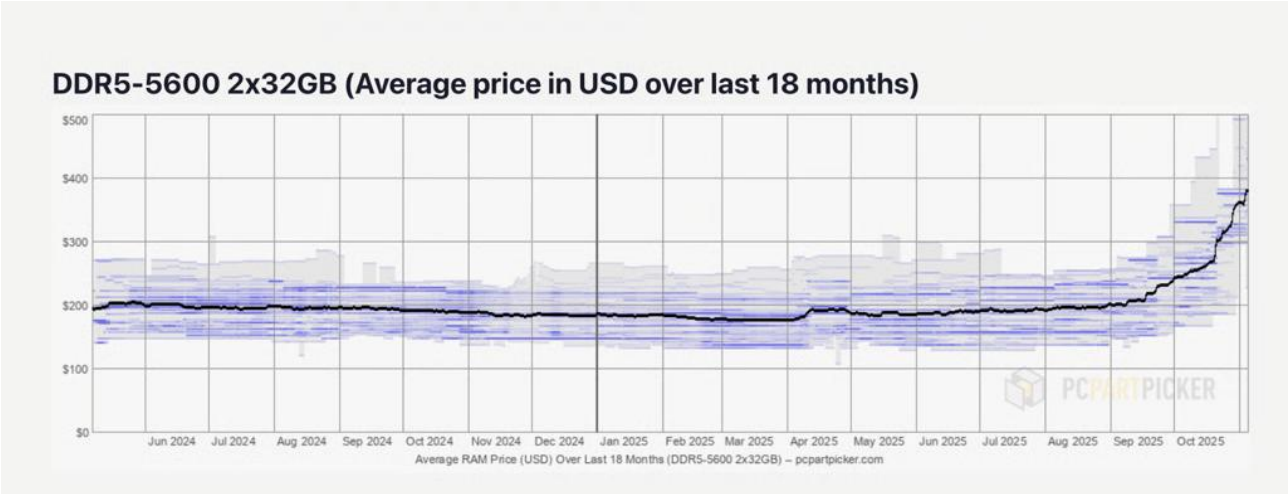


Рисунок 1.2 – Динаміка зростання цін на модулі пам’яті DDR5 у 2024-2025 роках через дефіцит виробничих потужностей [17]

Узагальнюючи аналіз апаратних та економічних факторів, можна констатувати формування глибокої «технологічної прірви» в індустрії.

Складається парадоксальна ситуація замкненого кола. З одного боку, відбувається постійне зростання вимог, сучасні стандарти якості, такі як нативна 4K роздільна здатність, фотореалізм та мікродеталізація (micro-displacement), вимагають обчислювальних ресурсів класу Hi-End, зокрема відеокарт з обсягом пам'яті понад 24 ГБ.

З іншого боку, виникає непереборний економічний бар'єр. Через глобальний дефіцит напівпровідників та ажіотажний попит з боку сфери штучного інтелекту, вартість необхідного обладнання зросла до рівня, недосяжного для більшості студентів та незалежних художників (бюджет робочої станції від $3000). Це призводить до штучного обмеження творчості: молоді спеціалісти змушені витрачати до 60% робочого часу не на творчий пошук, а на технічну оптимізацію сцени – зменшення роздільної здатності текстур та спрощення геометрії, аби відповідати жорстким апаратним лімітам наявного бюджетного обладнання. Це призводить до того, що традиційний метод трасування шляхів (Path Tracing) поступово перетворюється на елітарну технологію, повноцінний доступ до якої мають лише великі студії з власними рендер-фермами. Для масового користувача локальний рендеринг стає технологічним тупиком.

Єдиним ефективним вирішенням цієї проблеми є зміна парадигми, перехід від локальних обчислень (які жорстко обмежені бюджетом користувача) до хмарних генеративних технологій. Це дозволяє перекласти ресурсомісткі обчислення на сервери провайдера (в межах даної роботи – Google Cloud), демократизуючи доступ до високоякісної візуалізації. Саме такий підхід покладено в основу розробки програмного модуля «Nano Banana Render (Nanode)».

1.2 Еволюція методів нейронного рендерингу

Поняття «нейронний рендеринг» охоплює широкий клас методів, що використовують глибоке навчання для синтезу або покращення зображень. Історично цей напрямок розвивався від допоміжних інструментів пост-обробки

до повноцінних генеративних систем, здатних створювати контент «з нуля». Для розуміння архітектурних переваг проєктованої системи необхідно розглянути три ключові етапи розвитку технології.

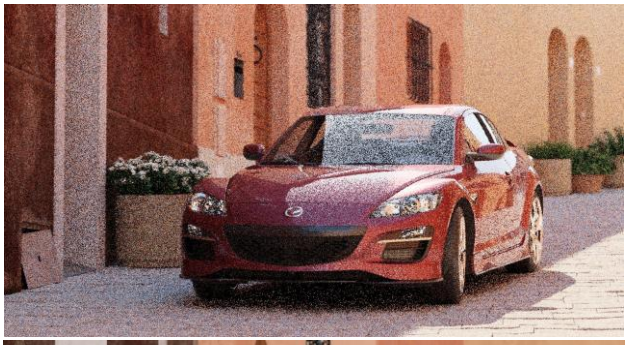
Першим успішним застосуванням нейромереж у продакшн-пайплайнах стала технологія інтелектуального знешумлення (AI-Accelerated Denoising). Вона вирішує фундаментальну проблему методу трасування шляхів (Path Tracing) – стохастичний шум, який виникає через недостатню кількість світлових семплів на піксель. Принцип роботи базується на використанні згорткових автоенкодерів (Convolutional Autoencoders), навчених на парах зображень «шумне-чисте». Прикладом промислового стандарту є NVIDIA OptiX AI Denoiser та Intel Open Image Denoise (OIDN) [18]. Характерною особливістю сучасних алгоритмів знешумлення є використання так званих допоміжних геометричних буферів (G-Buffers), які дозволяють нейромережі відокремлювати корисний сигнал від шуму. Зокрема, критично важливою є карта кольору матеріалів (Albedo Pass) – зображення, що містить інформацію про дифузний колір поверхонь без впливу тіней та освітлення.

Другим компонентом виступає карта нормалей (Normal Pass), яка описує векторний рельєф поверхні у кожній точці. Отримуючи ці дані разом із зашумленим рендером, нейромережа здатна ідентифікувати, де зміна кольору пікселя є наслідком геометричної деталізації, а де – випадковим шумом, що дозволяє зберегти чіткість країв об’єктів при фільтрації. Нейромережа отримує на вхід шумний рендер + карту нормалей і «розуміє», де шум є випадковим, а де – частиною текстури, зберігаючи чіткість країв об’єктів (рис. 1.3). Хоча цей метод прискорює візуалізацію у 2-4 рази, він є лише методом реконструкції і не здатний генерувати нові деталі, яких не було у геометрії сцени [3].

Ще одним, і на сьогодні найбільш масовим етапом еволюції ШІ-графіки, стала поява архітектури латентних дифузійних моделей (Latent Diffusion Models, LDM). Фундаментальна робота [3] змінила підхід до генерації: замість роботи з пікселями (як це робили GAN або звичайні дифузійні моделі), обчислення були перенесені у стиснутий векторний простір. Саме на цій технології базуються такі

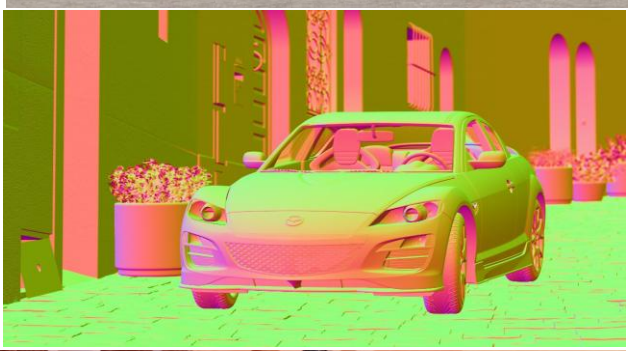
популярні рішення, як Stable Diffusion 1.5, SDXL та відповідні доповнення для Blender (Dream Textures).

а) зашумлене зображення (Вхід)



б) Albedo (дані)

в) Normal (дані)



г) чистий результат (вихід)

Рисунок 1.3 – Принцип роботи ШІ-знешумлення з використанням допоміжних карт (Albedo/Normal)

В основі методу лежить імовірнісний процес, який навчається звертати назад поступову деградацію даних. Якщо до будь-якого зображення поступово додавати гаусів шум, зрештою воно перетвориться на хаос. Нейромережа

навчається виконувати зворотну операцію (Reverse Diffusion Process): передбачати і віднімати шум крок за кроком, відновлюючи змістовне зображення з випадкового шуму.

З архітектурної точки зору, латентні дифузійні моделі (LDM) являють собою складну композицію трьох спеціалізованих нейромереж, що працюють у синергії (рис. 1.4). Першою ланкою є перцептивний автоенкодер (VAE – Variational Autoencoder), функція якого полягає у компресії вхідного зображення з піксельного простору у компактний латентний простір. Таке перетворення зменшує розмірність даних (типово у 48 разів), що уможливлює виконання обчислень на споживчих відеокартах, хоча і є процесом стиснення із втратами дрібних високочастотних деталей.

Центральним обчислювальним вузлом системи виступає мережа U-Net – глибока згорткова нейромережа, оснащена механізмом уваги. Її завданням є ітеративне передбачення шумової складової на кожному кроці генерації. Третім компонентом є текстовий енкодер (наприклад, CLIP або OpenCLIP), що відповідає за семантичний аналіз запиту. Він трансформує промпт користувача у набір числових векторів (embeddings), які надалі використовуються шарами Cross-Attention всередині U-Net для просторового керування генерацією.

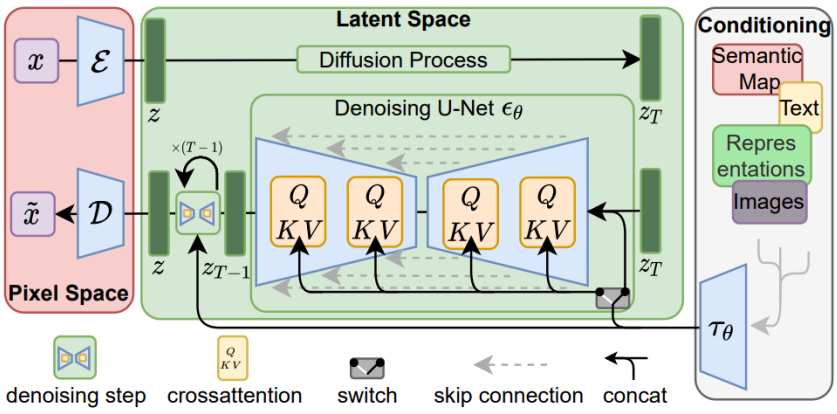


Рисунок 1.4 – Архітектура моделі Latent Diffusion (LDM): процес стиснення даних у латентний простір та механізм керування генерацією через Cross-Attention [3]

Попри революційність у галузі 2D-ілюстрації, при інтеграції у професійні 3D-пайплайни виявляються фундаментальні архітектурні обмеження LDM. По-перше, енкодер CLIP, що використовується у базових версіях, має жорстке технічне обмеження на довжину вхідної послідовності – 77 токенів. Це унеможливлює передачу комплексного технічного завдання, оскільки модель автоматично відсікає частину довгого опису.

По-друге, спостерігається проблема відсутності просторового розуміння (Spatial Agnosia). Оскільки навчання моделі відбувалося на парах «зображення-підпис», вона не сформувала глибинного розуміння тривимірних відношень. Інструкції з позиціонування об’єктів (наприклад, «ліворуч від», «на задньому плані») часто інтерпретуються некоректно через проблему зв’язування атрибутів (Attribute Binding). По-третє, існує обмеження фіксованої роздільної здатності навчального набору (512x512 або 1024x1024 пікселів). Спроба згенерувати зображення нестандартного співвідношення сторін або високої роздільної здатності (4K) призводить до дублювання об’єктів та появи артефактів, що вимагає використання додаткових інструментів апскейлінгу.

Третім, і на сьогодні найбільш передовим етапом розвитку технологій нейронного рендерингу, який активно формується в останні роки, стала фундаментальна зміна базової архітектури генеративних систем. Індустрія поступово відходить від використання спеціалізованих згорткових мереж (таких як U-Net у моделях Stable Diffusion) на користь мультимодальних трансформерів (Multimodal Transformers, рис. 1.5). Цей клас моделей, яскравими представниками якого є сучасні мультимодальні моделі на зразок Google Gemini, GPT-4o та Claude Sonnet, реалізує концепцію «уніфікованого сприйняття» (Unified Perception). Суть цієї парадигми полягає у тому, що різнорідні типи даних – текстові інструкції, програмний код, двовимірні зображення, карти глибини та відеопотоки – обробляються в єдиному векторному просторі без необхідності використання окремих адаптерів для кожної модальності [19; 20].

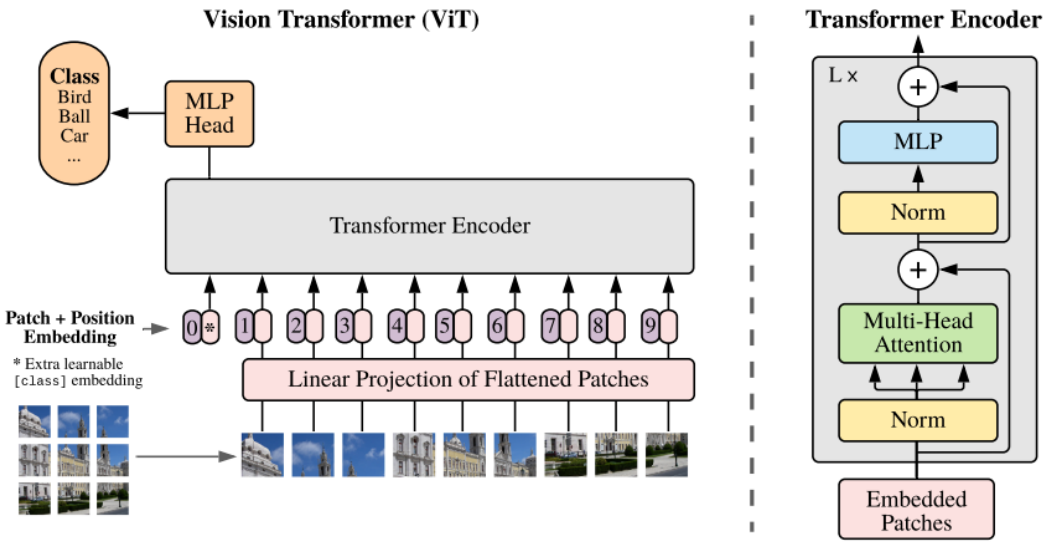


Рисунок 1.5 – Архітектура Vision Transformer (ViT): процес розбиття зображення на патчі та їх обробка механізмом уваги [20]

Фундаментальна відмінність мультимодальних трансформерів від дифузійних моделей попереднього покоління полягає у методі обробки вхідної інформації. Традиційні архітектури (Latent Diffusion Models) працюють із двома ізольованими потоками даних: текстовий запит обробляється енкодером CLIP, а зображення – варіаційним автоенкодером (VAE). Натомість, сучасні трансформери використовують архітектуру єдиного потоку (Single Stream), де візуальна інформація розглядається як своєрідна «іноземна мова».

Процес обробки починається з процедури візуальної токенізації. Вхідне зображення або карта глибини (mist pass) не стискається у латентний простір, як це відбувається у VAE, а розбивається на сітку фіксованих фрагментів, які називаються патчами. Розмір такого патча зазвичай становить 14×14 або 16×16 пікселів. Далі кожен патч проходить через шар лінійної проєкції (Linear Projection), де піксельні дані трансформуються у одновимірний вектор. До цього вектора додається позиційне кодування (Positional Embedding), яке дозволяє нейромережі зберегти інформацію про те, в якій частині зображення знаходився конкретний фрагмент. У результаті цієї операції зображення перетворюється на

послідовність «візуальних токенів», які для архітектури трансформера є математично еквівалентними текстовим токенам (словам).

Ключовим обчислювальним ядром системи виступає механізм глобальної самоуваги. На відміну від згорткових мереж, які аналізують зображення локально (ковзним вікном), механізм уваги дозволяє кожному токену «бачити» всі інші токени одночасно. Це дозволяє моделі встановлювати складні семантичні та причинно-наслідкові залежності між будь-якими елементами послідовності. Наприклад, система здатна ідентифікувати, що слово «відображення» у текстовому промпті семантично пов’язане з конкретною групою пікселів дзеркальної поверхні на зображенні, навіть якщо у вхідній послідовності вони знаходяться на значній відстані один від одного.

Застосування мультимодальних трансформерів дає змогу подолати низку критичних обмежень, властивих дифузійним моделям, та забезпечує нову якість генерації. Першою і найвагомішою перевагою є наявність просторового інтелекту (Spatial Intelligence). Традиційні моделі часто сприймають зображення як сукупність текстурних ознак («мішок слів»), не розуміючи тривимірної структури сцени. Мультимодальні трансформери, які навчалися на величезних масивах 3D-даних та відеопослідовностях, розвинули емерджентну властивість просторового мислення. Модель здатна виконувати логічні операції з геометрією: вона коректно інтерпретує градієнти яскравості на карті глибини як метричну інформацію про відстань до об’єкта. Це дозволяє генерувати складні оптичні ефекти, такі як атмосферна перспектива (туман, що густішає з відстанню) або глибина різкості (Depth of Field), фізично коректно. Крім того, модель безпомилково обробляє явища оклюзії, розуміючи, що об’єкт переднього плану повинен перекривати фон, а не змішуватися з ним .

Другою суттєвою перевагою є здатність до побудови ланцюжка міркувань (Chain-of-Thought Rendering). Завдяки інтеграції з потужним мовним ядром LLM, система не просто генерує пікселі на основі статистики, а фактично планує структуру майбутнього зображення. Цей процес, відомий у науковій літературі як Visual Chain-of-Thought, дозволяє системі вибудовувати логічні

зв’язки освітлення ще до початку візуалізації. Наприклад, отримавши завдання на рендеринг інтер’єру зі скляним столом, модель внутрішньо формує послідовність логічних тверджень: «джерело світла знаходиться зліва», «стіл є прозорим», «отже, промені мають пройти крізь стільницю і утворити каустику на підлозі». Традиційні дифузійні моделі позбавлені цього логічного шару, тому часто припускаються грубих фізичних помилок (наприклад, тіні, що падають у різні боки), тоді як трансформер забезпечує логічну консистентність сцени .

Третьою технічною перевагою є нативна масштабованість (Resolution Independence). Архітектура U-Net має жорстку прив’язку до роздільної здатності навчального набору (зазвичай 512×512 або 1024×1024 пікселів), оскільки використовує фіксовані ядра згорток. Трансформери ж оперують послідовностями токенів довільної довжини. Збільшення роздільної здатності вихідного зображення (наприклад, до стандарту 4K) для такої архітектури означає лише лінійне збільшення кількості токенів у послідовності. Це дозволяє реалізувати концепцію «нативного 4K рендерінгу» без використання додаткових інструментів апскейлінгу, які часто призводять до втрати деталізації та появи артефактів.

Сукупність вищезазначених архітектурних переваг – глибокого просторового інтелекту, здатності до логічного планування сцени та незалежності від роздільної здатності – визначає мультимодальні трансформери як оптимальну технологічну основу для реалізації програмного модуля «Nano Banana Pro Render». Саме цей підхід дозволяє вирішити ключову проблему, сформульовану в аналізі предметної області: забезпечення високої якості візуалізації (нативний 4K) на споживчому апаратному забезпеченні, нівелюючи необхідність у дороговартісних відеокартах з великим обсягом пам’яті.

1.3 Аналіз наявних програмних рішень та їх технологічних обмежень

На ринку інструментів для інтеграції генеративного штучного інтелекту в середовище 3D-моделювання Blender наразі спостерігається активна фаза експериментальних розробок. Аналіз відкритих репозиторіїв на платформі

GitHub дозволяє класифікувати існуючі рішення на дві групи: локальні клієнти (Local Execution), що використовують апаратні ресурси користувача, та хмарні плагіни (Cloud Connectors).

Для об’єктивної оцінки було проаналізовано первинний код та технічну документацію найбільш популярних проєктів. Доповнення Dream Textures (офіційний репозиторій carson-katri/dream-textures [21]) є найбільш відомим Open Source рішенням, що інтегрує бібліотеку diffusers та моделі Stable Diffusion в інтерфейс Blender. Аналіз технічної документації репозиторію та журналу помилок (Issue Tracker) дав можливість виявити критичні архітектурні вади.

Згідно із системними вимогами, зазначеними в документації розробника, для роботи з моделями SDXL необхідна відеокарта з підтримкою CUDA та мінімум 12-14 ГБ VRAM для стабільної генерації без артефактів [21]. Це створює вузьке місце для більшості користувачів. На відеокартах середнього сегменту (NVIDIA RTX 3050/4060 з 4–6 ГБ) доовнення змушене використовувати механізм System Memory Fallback, вивантажуючи шари нейромережі в повільну оперативну пам’ять. Це призводить до падіння швидкості генерації з 15 секунд до 10–15 хвилин на одне зображення та робить інструмент непридатним для ітеративного дизайну.

Аналіз скриптів інсталяції показує, що інструмент намагається розгорнути власне віртуальне середовище Python всередині Blender, завантажуючи важковагові бібліотеки PyTorch, Transformers та Diffusers. Це є джерелом постійних конфліктів. У розділі «Issues» на GitHub зафіксовано понад 300 звернень, пов’язаних із помилками інсталяції:

– несумісність версій Python, вбудованого у нові версії Blender (4.0+);
– конфлікти драйверів CUDA та бібліотек xformers;
– помилки при спробі завантажити ваги моделей (Checkpoints) обсягом 6-10 ГБ при нестабільному з’єднанні.

Оскільки Dream Textures базується на класичних дифузійних моделях, він успадковує їхні архітектурні обмеження: ліміт промпту у 77 токенів (через

енкодер CLIP) та неможливість нативної генерації 4K без використання апскейлерів, які розмивають текстури.

Офіційний плагін Stability for Blender (репозиторій Stability-AI/stability-blender-addon-public [22]) реалізує модель «тонкого клієнта», відправляючи запити на API Stability AI. Аналіз кодової бази клієнта (client.py) показує, що плагін використовує застарілі ендпоінти API (Legacy DreamStudio), які не підтримують новітні методи контролю композиції, доступні у моделях 2025 року. Функціональність Inpainting реалізовано на базовому рівні, часто залишаючи видимі шви на текстурах.

Використання плагіна жорстко прив’язане до кредитної системи (API Credits). Це створює бар’єр для студентів та незалежних художників. На відміну від цього, розроблюваний у даній роботі модуль буде використовувати безкоштовні квоти Google Gemini API, що робить його більш доступним.

Плагін обробляє карту глибини (Depth Map) як звичайне 2D-зображення. Він не використовує просторовий контекст сцени так, як це роблять мультимодальні трансформери (Gemini). Це призводить до помилок у перспективі та освітленні при спробі згенерувати складні об’ємні ефекти (наприклад, туман або відображення). [5; 6; 7]

Підсумовуючи порівняльний аналіз технічної документації та первинного коду аналогів, можна виокремити такі проблеми: локальні рішення (Dream Textures) недоступні для масового користувача через високі вимоги до VRAM та складність налаштування середовища, а хмарні плагіни є технологічно обмеженими та економічно невигідними. Це обґрунтовує необхідність розробки власного рішення «Nano Banana Pro Render», яке використовує хмарну архітектуру Multimodal LLM для подолання апаратних обмежень та забезпечення глибокого розуміння 3D-сцени.

1.4 Обґрунтування вибору технологій та постановка задачі

На основі проведеного аналізу можна зробити висновок, що для створення сучасного інструменту візуалізації, доступного широкому колу користувачів,

використання локальних дифузійних моделей є архітектурним тупиком через високі апаратні вимоги та обмежену якість.

Стратегія розробки проектованого модуля «Nano Banana Pro Render» полягає у використанні хмарних обчислень та найновіших мультимодальних моделей (Gemini 3 Pro). Це дозволяє винести ресурсомісткі задачі на сервери Google, забезпечуючи кінцевому користувачеві можливість отримувати 4K-рендери навіть на слабких ноутбуках. Порівняльна характеристика проектованого рішення з існуючими аналогами наведена в табл. 1.1 [5-7].

Таблиця 1.1 – Порівняльний аналіз систем ШІ-рендерингу для Blender

Параметр порівняння	Dream Textures (Аналог)	Stability for Blender (Аналог)	Nano Banana Pro (Пропоноване)
Архітектура моделі	Stable Diffusion 1.5 / SDXL	Stable Diffusion XL	Google Gemini 3 Pro (Multimodal)
Місце обчислень	Локальний ПК (Local GPU)	Хмара Stability AI	Хмара Google Cloud
Вимоги до VRAM	Мінімум 8-12 ГБ (Критично)	Низькі	Мінімальні (будь-який GPU)
Нативна роздільна здатність	512px / 1024px	1024px	Native 4K (4096px)
Метод отримання 4K	AI Upscaling (втрата якості)	Upscaling	Нативна генерація (без втрат)
Час генерації (4K)	~3-5 хвилин (з апскейлом)	~1-2 хвилини	~30 секунд
Розуміння контексту	Низьке (CLIP обмеження)	Середнє	Високе (Advanced LLM)
Доступність для новачків	Низька (складна інсталяція)	Середня	Висока (Plug & Play)

Загалом, за результатами ґрунтовного системного аналізу предметної області було виявлено критичні обмеження традиційного рендерингу методом трасування шляхів (Path Tracing). Визначено, що цей підхід є архітектурним тупиком через експоненційну обчислювальну складність, надмірні вимоги до відеопам’яті (VRAM) та високий економічний бар’єр для масового користувача.

Аналіз наявних AI-аналогів на базі дифузійних моделей (Stable Diffusion) також показав їхню вразливість перед проблемою втрати просторового

розуміння та нездатність генерувати зображення у нативній роздільній здатності 4K без артефактів апскейлінгу.

Обґрунтовано стратегічне рішення щодо розробки програмного модуля «Nano Banana Render (Nanode)», що використовує хмарну архітектуру мультимодальних трансформерів (Google Gemini), що дозволяє забезпечити нативну якість 4K та високу швидкість візуалізації, повністю нівелюючи залежність від локального апаратного забезпечення [5-7].

На основі цього аналізу сформульовано технічні задачі роботи: спроєктувати мікросервісну клієнт-серверну архітектуру (Blender-FastAPI-Gemini), реалізувати алгоритм випереджального захоплення кадру (Pre-capture), створити інструменти просторового керування Smart Points та Inpaint Editor для художнього контролю, а також впровадити комплексну систему безпеки та адміністрування.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Концептуальне моделювання та специфікація вимог до системи

Перехід від концептуального опису предметної області до безпосереднього програмування вимагає строгої формалізації вимог та побудови поведінкових моделей системи. На цьому етапі визначаються ключові суб’єкти взаємодії (актори) та прецеденти, що описують логіку функціонування розроблюваного програмного комплексу «Nano Banana Render».

Основними користувачами системи є 3D-художник (дизайнер), який безпосередньо взаємодіє з інтерфейсом Blender, та Адміністратор платформи, який здійснює моніторинг інфраструктури через вебпанель. Роль 3D-художника є центральною, оскільки саме цей користувач ініціює основний сценарій роботи системи: підготовку сцени, запуск хмарного рендерингу, уточнення результату за допомогою локального редагування та повторне використання попередніх генерацій. Для нього критичними є безперервність творчого процесу, мінімальна кількість ручних операцій і відсутність блокування графічного інтерфейсу Blender під час обробки запиту.

Адміністратор платформи не бере участі у творчому процесі безпосередньо, однак забезпечує контроль працездатності інфраструктури. Його функції пов’язані з переглядом стану сервісів, моніторингом запитів, контролем помилок, аналізом навантаження та перевіркою коректності роботи серверної частини. Такий розподіл ролей дає змогу відокремити користувацькі сценарії створення зображення від інфраструктурних завдань супроводу системи.

На відміну від класичного локального рендерингу, у якому всі обчислення виконуються на робочій станції користувача, запропонований комплекс передбачає винесення частини генеративної обробки в хмарне середовище. Тому вимоги мають охоплювати не лише стандартні дії користувача в інтерфейсі, а й приховані системні процеси: підготовку сцени до передавання, формування

допоміжних даних, обробку відповіді від API, кешування результатів, повторне використання параметрів генерації та контроль стабільності сервісу.

Діаграма прецедентів (рис. 2.1) використовується для узагальненого подання функціональних меж програмного комплексу. Вона відображає, які дії доступні кожному актору та які функції повинні бути реалізовані для забезпечення повного циклу роботи з системою: від запуску генерації до перегляду результатів і контролю стану платформи. Така модель не деталізує внутрішні алгоритми, проте фіксує зовнішню поведінку системи, тобто те, як її функціональність сприймається користувачами.

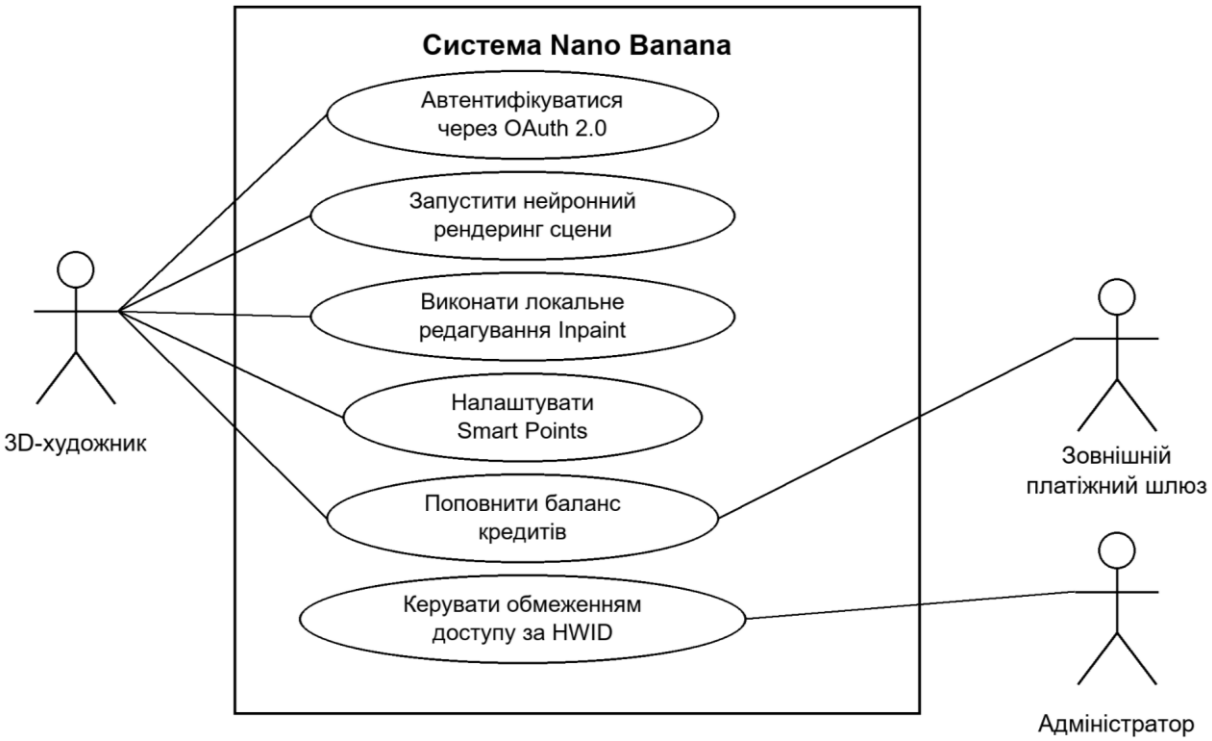


Рисунок 2.1 – Діаграма прецедентів для програмного забезпечення

У межах діаграми доречно розглядати Blender не лише як середовище моделювання, а й як клієнтський інтерфейс до хмарного ШІ-рендерингу. Саме через нього 3D-художник передає сцену, промпт, маску або просторові маркери. Вебпанель адміністратора, навпаки, виконує роль інструмента спостереження за серверною частиною та не втручається у процес творчої генерації.

Наведена діаграма демонструє, що основний функціональний контур системи зосереджений навколо 3D-художника. До нього належать запуск хмарного рендерингу, передавання структурних даних сцени, редагування окремих фрагментів зображення, робота зі Smart Points та управління історією генерацій. Ці прецеденти формують цілісний сценарій інтерактивного створення зображення, у якому користувач не просто отримує результат від нейромережі, а поступово керує ним через геометричні, текстові та візуальні обмеження.

Окрему групу становлять адміністративні прецеденти, пов’язані з контролем серверної інфраструктури. Їх наявність обґрунтована тим, що хмарна генерація залежить від доступності зовнішнього API, коректності авторизації, стабільності черги запитів і достатності обчислювальних ресурсів. У разі збільшення кількості користувачів саме адміністративний контур забезпечує можливість оперативного виявлення помилок, аналізу навантаження та прийняття рішень щодо масштабування.

Для деталізації технічного завдання та забезпечення вимірюваності критеріїв приймання програмного продукту, нижче згруповано та специфіковано ключові функціональні (FR, табл. 2.1) та нефункціональні вимоги (NFR, табл. 2.2) до системи.

Вимоги FR-1-FR-5 охоплюють повний цикл роботи 3D-художника з програмним комплексом. Спочатку користувач запускає хмарну генерацію фінального зображення, після чого система передає допоміжні структурні дані сцени, необхідні для збереження композиції. Далі користувач може локально уточнювати результат за допомогою маски, керувати просторовими орієнтирами через Smart Points і повертатися до попередніх варіантів завдяки історії генерацій. Таким чином, функціональні вимоги орієнтовані не на одноразову генерацію кадру, а на ітеративний творчий процес.

Специфіковані функціональні вимоги демонструють, що ключовою особливістю комплексу є поєднання генеративного ШІ з керованістю результату. Вимога FR-1 визначає базову функцію системи — отримання зображення через віддалене API, однак сама по собі вона не гарантує точного відтворення

композиції сцени. Саме тому FR-2 передбачає передавання структурних даних, зокрема карти глибини та метаданих об’єктів. Це дає змогу зменшити ризик того, що нейромережа проігнорує просторові пропорції, взаємне розміщення об’єктів або загальну композиційну логіку сцени.

Таблиця 2.1 – Специфікація функціональних вимог до програмного комплексу

ID	Найменування вимоги	Опис функціонального контексту	Критерій приймання
FR-1	Хмарний ШІ-рендеринг	Генерація фінального зображення тривимірної сцени через віддалене API.	Користувач запускає рендер з інтерфейсу Blender і отримує результат без блокування GUI.
FR-2	Передавання структурних даних сцени	Експорт карти глибини (Mist Pass) та текстових метаданих об’єктів.	Дані містять повну геометричну структуру сцени для збереження композиції.
FR-3	Локальне редагування (Inpaint)	Перемальовування виділених фрагментів зображення за допомогою растрової маски.	Модифікація пікселів відбувається лише у межах маски без перегенерації всього кадру.
FR-4	Просторова прив'язка Smart Points	Керування позиціонуванням об’єктів за допомогою інтерактивних 2D/3D маркерів.	Координати точок зчитуються та враховуються нейромережею під час генерації деталей.
FR-5	Управління історією генерацій	Локальне кешування та відновлення параметрів попередніх рендерів.	Користувач може переглядати галерею, порівнювати версії та відновлювати промпти/моделі.

Вимога FR-3 розширює можливості системи після отримання первинного результату. Локальне редагування через Inpaint дозволяє виправляти окремі ділянки зображення без повторної генерації всього кадру, що економить час користувача і зменшує витрати на хмарні обчислення. Вимога FR-4 пов’язана з підвищенням керованості генерації через Smart Points: такі маркери дають змогу задавати просторові акценти, уточнювати розташування деталей і передавати нейромережі додаткові обмеження. Вимога FR-5 забезпечує повторюваність творчого процесу, оскільки історія генерацій дозволяє аналізувати проміжні результати, порівнювати варіанти й повертатися до попередніх параметрів.

У табл. 2.2 нефункціональні вимоги подано через технічне обмеження та метрику або інструмент валідації, що дає змогу одразу пов’язати кожну вимогу з конкретним способом перевірки. Нефункціональна вимога NFR-1 є критичною через характер даних, з якими працює система. Під час взаємодії з хмарним API можуть передаватися службові токени, параметри генерації, метадані сцени та результати рендерингу. Використання HTTPS, OAuth 2.0 / JWT та HMAC SHA-256 зменшує ризики несанкціонованого доступу, підміни запитів або витоку конфіденційної інформації. У контексті інтеграції з Blender це особливо важливо, оскільки користувач очікує, що процес генерації буде прихованим від сторонніх осіб і не вимагатиме ручного керування чутливими ключами доступу.

Таблиця 2.2 – Специфікація нефункціональних вимог, атрибутів якості

ID	Атрибут якості	Опис технічного обмеження	Метрика / Інструмент валідації
NFR-1	Інформаційна безпека	Захист конфіденційних даних, токенів та верифікація транзакцій.	Обмін даними через HTTPS, авторизація OAuth 2.0 / JWT, підписи HMAC SHA-256.
NFR-2	Масштабованість	Автоматичне регулювання обчислювальних потужностей сервера під навантаженням.	Розгортання у безсерверному середовищі Cloud Run із підтримкою тактики Scale-to-Zero.
NFR-3	Супроводжуваність коду	Забезпечення чистоти, модульності та низького рівня технічного боргу.	Обов’язковий статичний аналіз SonarCloud із цільовою оцінкою надійності класу «A».

Вимога NFR-2 пов’язана з непередбачуваністю навантаження на сервіс генерації. Запити на ШІ-рендеринг можуть бути нерівномірними: у певні періоди система може майже не використовуватися, а в інші – отримувати велику кількість паралельних звернень. Розгортання в середовищі Cloud Run із підтримкою Scale-to-Zero дозволяє оптимізувати витрати на інфраструктуру та автоматично збільшувати кількість екземплярів сервісу в разі зростання навантаження. Це робить архітектуру придатною як для одиничного використання, так і для подальшого масштабування.

Вимога NFR-3 визначає якість внутрішньої структури програмного коду. Оскільки комплекс поєднує клієнтське розширення Blender, серверну логіку,

інтеграцію з API, обробку даних сцени та вебмоніторинг, без належної модульності код швидко стане складним для супроводу. Використання SonarCloud [23] як інструмента статичного аналізу дає змогу контролювати надійність, потенційні помилки, дублювання коду та технічний борг.

У сукупності функціональні та нефункціональні вимоги формують основу для подальшого проєктування архітектури програмного комплексу. Функціональні вимоги визначають, які можливості повинна надати система користувачу, тоді як нефункціональні вимоги задають обмеження щодо способу реалізації цих можливостей. Такий поділ дає змогу перейти від загального опису ідеї «Nano Banana Render» до конкретних архітектурних рішень, вибору технологій і сценаріїв тестування.

2.2 Попередня архітектура програмного забезпечення

Архітектура розробленого програмного комплексу Nano Banana Render (Nanode) базується на сервіс-орієнтованій архітектурі (SOA) та відповідає багаторівневій моделі абстракції. На найвищому рівні система забезпечує взаємодію між користувачем (художником), внутрішніми інфраструктурними сервісами (API Gateway) та зовнішніми провайдерами хмарних обчислень. Головною метою проєктування була відмова від монолітної структури, яка б змушувала локальну обчислювальну машину виконувати надмірні математичні розрахунки.

Використання традиційної монолітної архітектури в контексті програм для моделювання накладає критичні обмеження на локальні апаратні ресурси. Сучасні нейронні мережі вимагають значного обсягу відеопам’яті (VRAM) графічного адаптера (GPU) та високої обчислювальної потужності. Запуск подібних математичних моделей безпосередньо на комп’ютері користувача у вигляді локального процесу призводить до суттєвого падіння продуктивності відображення у робочій області (Viewport), а в більшості випадків зупиняє роботу через помилку вичерпання доступної пам’яті.

Враховуючи ці проблеми, було ухвалено рішення про проєктування хмарної інфраструктури з чітким розподілом зон відповідальності (Separation of Concerns). Система логічно розділена на три незалежні вузли, зв’язок між якими здійснюється виключно через мережеві протоколи REST API. Клієнтська частина (Blender Add-on) буде реалізована мовою Python з використанням бібліотеки bpy [8; 9]. Вона відповідає за збір просторових даних сцени, створення базового зображення (Pre-pass) та формування JSON Payload для відправки.

Серверна частина (API Gateway) побудована на базі асинхронного фреймворку FastAPI та забезпечує маршрутизацію мережевих запитів, приховування секретних ключів, перевірку JWT-токенів авторизації та управління базою даних SQLite. Клієнтський вебінтерфейс реалізується односторінковий застосунок на React призначений для реєстрації користувачів, обробки платежів та візуалізації статистики.

Для забезпечення високої продуктивності, масштабованості та безпеки обробки даних, архітектура програмного комплексу «Nano Banana Pro Render» побудована за сучасним мікросервісним принципом із розгортанням у безсерверному середовищі (Serverless). Запропонована декомпозиція дозволяє незалежно масштабувати кожен компонент. Зокрема, розгортання бекенду на Google Cloud Run забезпечує автоматичне масштабування від нуля до тисяч екземплярів контейнерів під час пікових навантажень (Scale-to-Zero), що оптимізує фінансові витрати на інфраструктуру. Нижче наведено організаційну схему клієнт-серверної взаємодії (рис. 2.2).

Для деталізації основного сценарію взаємодії Blender-клієнта з серверною частиною розглянуто процес REST API-запиту генерації. Діаграма активності відображає підготовку payload на клієнті, перевірку токена, HWID та кредитного балансу на сервері, звернення до Gemini API і повернення результату користувачу.

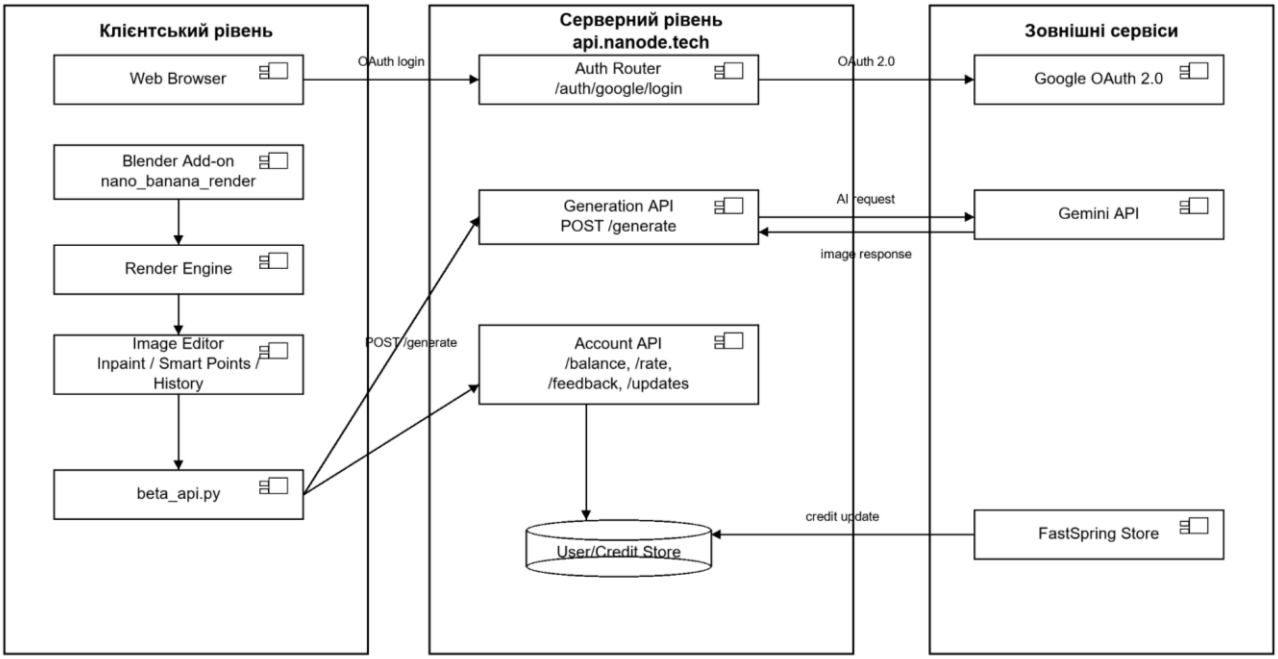


Рисунок 2.2 – Організаційна схема клієнт-серверної взаємодії системи

Фізичне розділення архітектури відображено також на рівні системи контролю версій. Як показано на рис. 2.2, первинний код клієнтського плагіна та вебпорталу зберігається у виділених сховищах. Ізоляція кодових баз дозволятиме розгортати застосунки незалежно один від одного через власні автоматизовані конвеєри CI/CD. Такий розподіл відповідальності забезпечує високу стійкість системи до відмов.

2.3 Імплементація клієнтської частини та інтеграція рушія рендерингу

Клієнтський рівень містить вебпортал nanode.tech (на базі React/Vite/Tailwind) та клієнтський додаток для Blender. Процес автентифікації користувача побудований на глибокій інтеграції з сервісом Google OAuth 2.0 [10]. Процес реєстрації та логіну (рис. 2.3) ініціюється безпосередньо в інтерфейсі аддону Blender. При натисканні кнопки логіну, аддон відкриває вебброузер і перенаправляє користувача на захищену сторінку авторизації Google. Для забезпечення цього потоку, мною було створено OAuth 2.0 Client ID у Google Cloud Console, налаштовано списки дозволених доменів (Authorized JavaScript origins) та URI перенаправлення (Redirect URIs).

Після успішного підтвердження особи на стороні Google, відбувається автоматичний редирект на спеціальний ендпоінт бекенду: https://api.nanode.tech/auth/google/callback. Цей маршрут обробляє код авторизації, генерує JWT-токен для сесії та повертає вебсторінку, яка автоматично запитує перехід назад у Blender, використовуючи механізм глибоких посилань (Deep Linking) або спеціальні скрипти. Завдяки цьому, користувачу не потрібно вручну копіювати API-ключ. Токен автоматично вставляється у налаштування аддону Blender, забезпечуючи максимально безшовний користувацький досвід.

Рисунок 2.3 – UML-діаграма активності REST API-запиту генерації

Клієнтська частина програмного комплексу інтегрується в середовище моделювання Blender не як допоміжний скрипт або макрос, а як повноцінний нативний рушій рендерингу (Render Engine). Основною ідеєю проєкту було органічне вбудовування генеративного конвеєра у звичний робочий процес художника. Для досягнення цього, головний клас NanoBananaRenderEngine був успадкований безпосередньо від базового класу bpy.types.RenderEngine. Це архітектурне рішення дозволило зареєструвати рушій у внутрішньому реєстрі Blender, завдяки чому він з'являється у стандартному випадаючому списку поруч із класичними рушіями (Cycles та Eevee).

Алгоритмічна складність модулю render_engine.py зумовлена специфікою архітектури програми. Блокування головного обчислювального потоку (Main Thread) під час виклику стандартного оператора bpy.ops.render.render() призводило до аварійного завершення роботи застосунку при спробі отримати графічний буфер робочої області (Viewport) через OpenGL. Ця проблема відома як Render-lock. Для подолання цього технологічного бар'єру було розроблено алгоритм випереджального захоплення кадру (Pre-capture, рис. 2.4). Логіка алгоритму випереджального захоплення складається з таких кроків:

1) користувач ініціює рендер натисканням кнопки у створеному кастомному інтерфейсі (або через гарячу клавішу); замість стандартного оператора викликається BananaOTRender;

2) оператор тимчасово перехоплює управління і виконує захоплення кадру з робочої області (Viewport) у режимі Solid або Material Preview за допомогою методів bpy.ops.render.opengl(write_still=True);

3) збережений референсний кадр тимчасово записується у файлову систему (наприклад, у директорію %TEMP% або /tmp), а шлях до нього кешується у глобальному словнику модуля _pre_capture;

4) після успішного запису оператор викликає стандартний процес рендерингу; на цьому етапі управління переходить до методів update() та render() класу NanoBananaRenderEngine.

5) метод render() витягує шлях до референсного зображення зі словника _pre_capture, серіалізує налаштування з UI (текстовий промпт, обрану модель, референсне зображення) у формат JSON та ініціює блокуючий HTTP POST запит до сервера через бібліотеку requests;

6) після отримання згенерованого зображення (у форматі байтів) від програмного інтерфейсу, плагін використовує методи render_result. load_from_file() для відображення результату безпосередньо у вікні Image Editor.

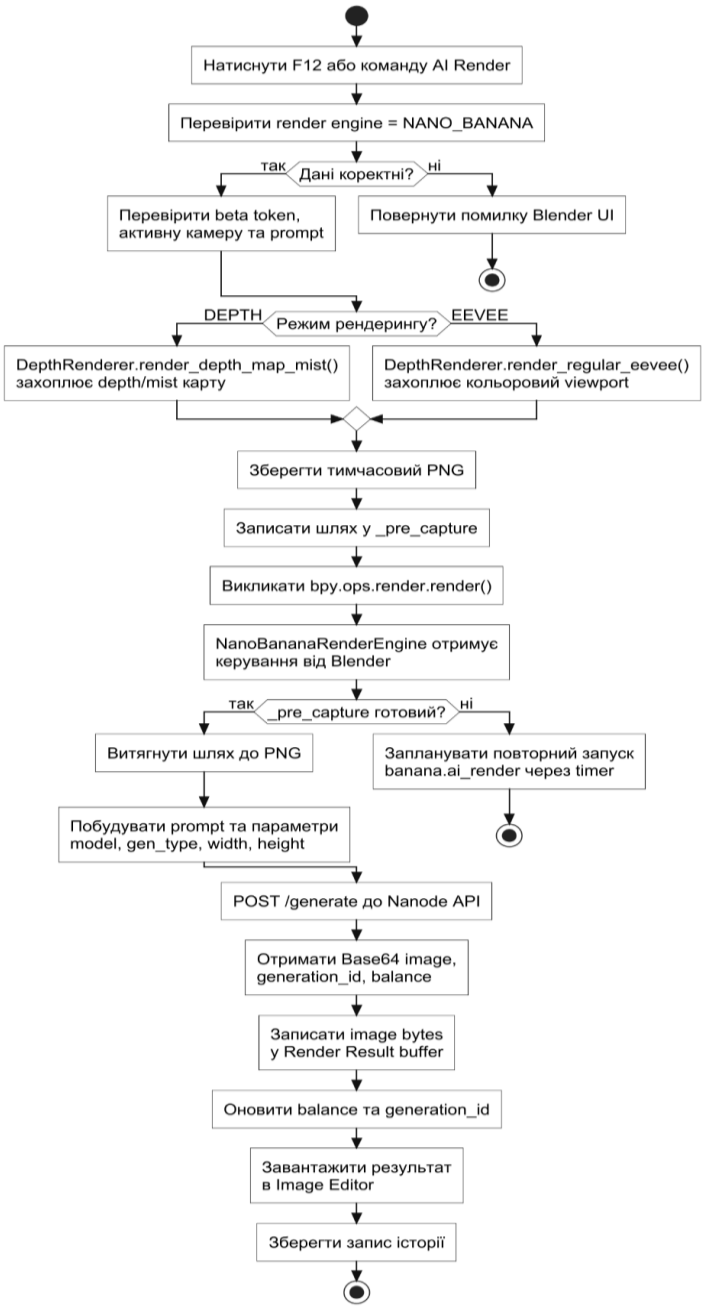


Рисунок 2.4 – Алгоритм випереджального захоплення кадру

Рис. 2.5 демонструє успішний результат роботи цього алгоритму. Завдяки механізму випереджального захоплення графічний інтерфейс програми залишається стабільним під час очікування відповіді від мережі.

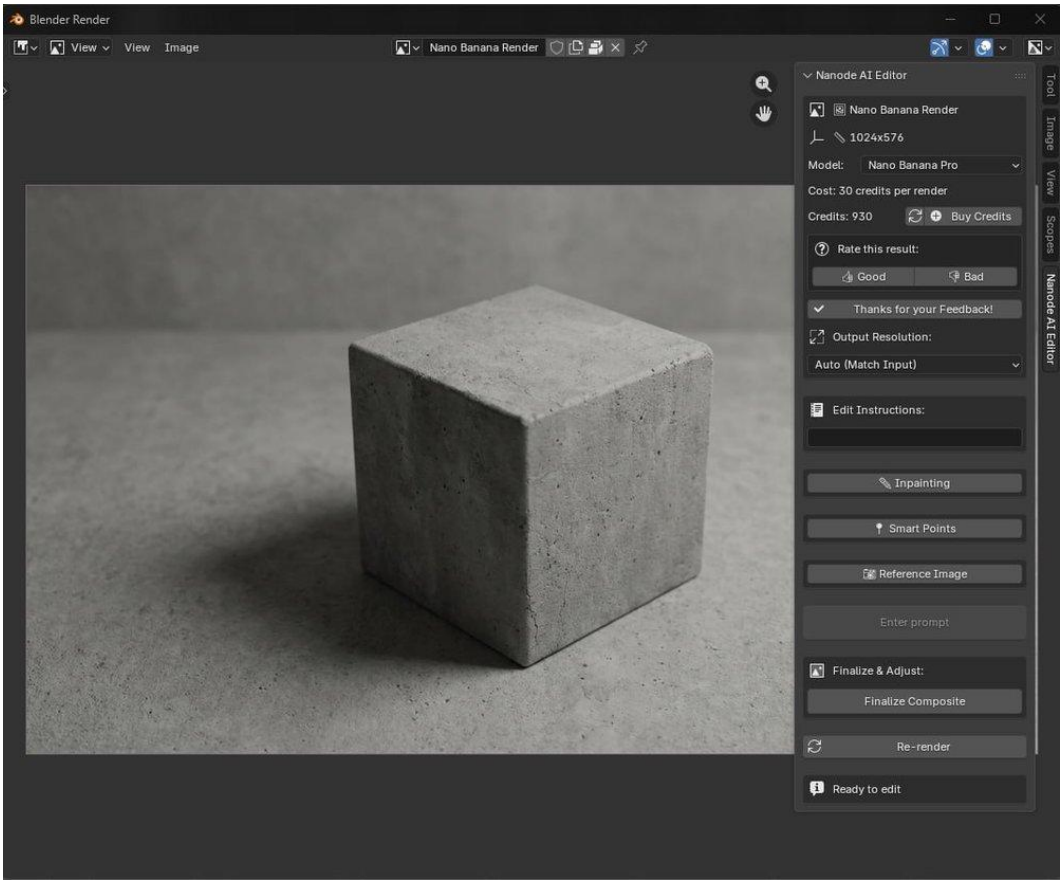


Рисунок 2.5 – Вивід результату рендеру в програмі

Окремої уваги заслуговує проєктування графічного інтерфейсу. Оскільки Nano Banana є повноцінним рушієм рендерингу, його основні налаштування логічно інтегровані у вкладку Render Properties (bl_space_type = PROPERTIES), а не у допоміжну бокову панель (N-Panel). Таке розміщення забезпечує інтуїтивно зрозумілий робочий процес.

Для збереження введених даних була використана внутрішня архітектура PropertyGroups. Налаштування рушія складаються переважно з текстового опису (StringProperty), вибору генеративної моделі (EnumProperty) та завантаження зображення-стилю. Ці змінні зареєстровані як глобальні

властивості сцени, що гарантує збереження всіх параметрів всередині файлу проєкту.

Як показано на рисунках 2.6 та 2.7, застосування нативних елементів інтерфейсу через методи layout.prop(), row та column забезпечує повну сумісність з візуальними темами програми. Бокова панель інструментів (N-Panel) також присутня в архітектурі плагіна, проте вона відведена під експериментальний AI Editor для інтелектуального редагування окремих об'єктів, що було реалізовано на пізніших етапах розвитку проєкту.

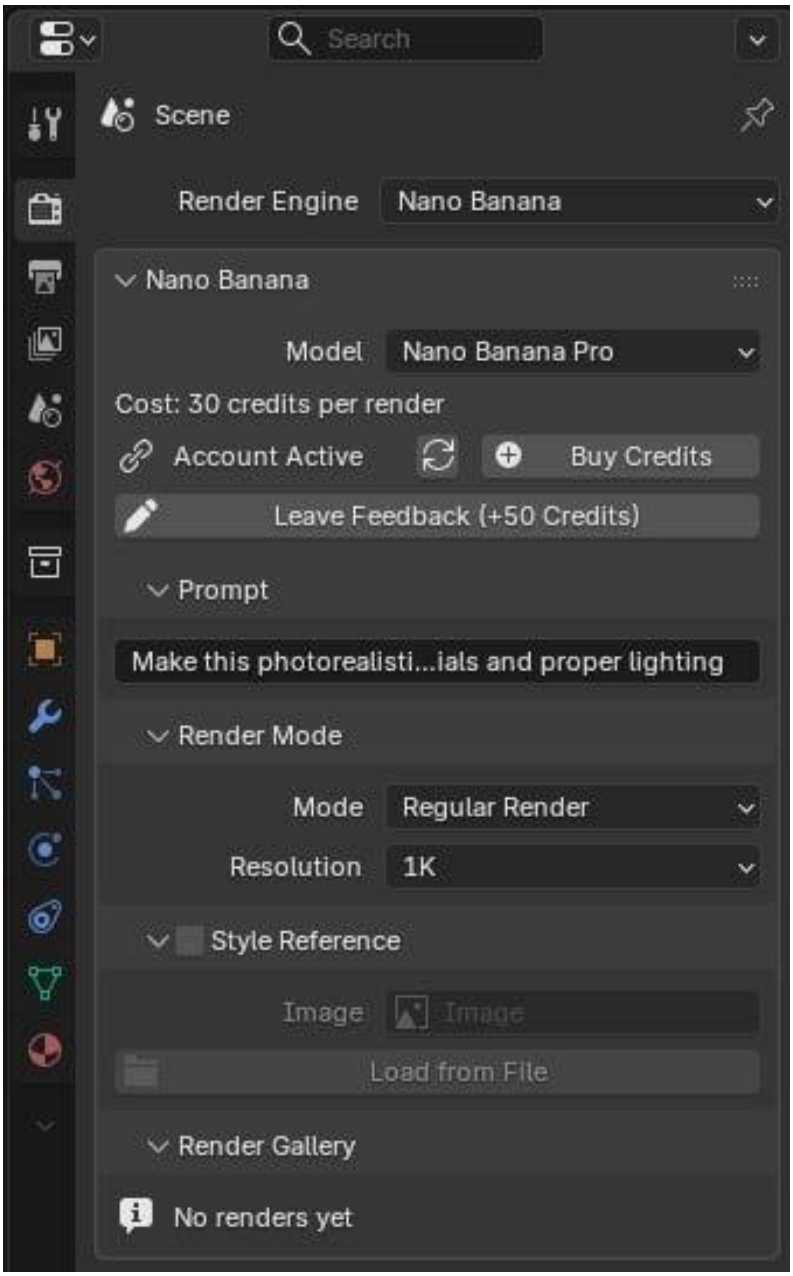


Рисунок 2.6 – Графічний інтерфейс налаштувань рушія

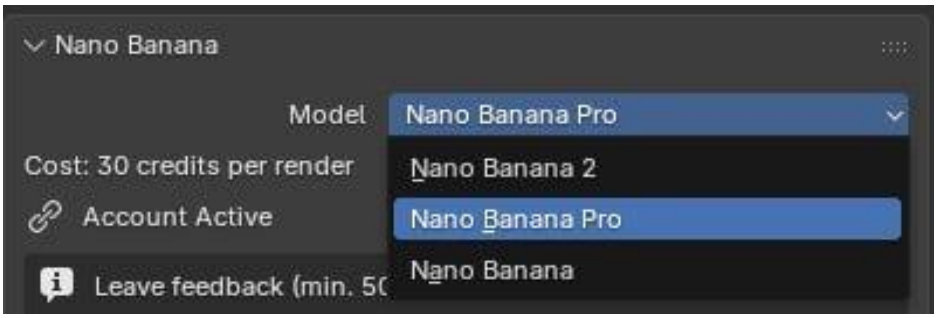


Рисунок 2.7 – Елемент EnumProperty для вибору моделі ШІ

З огляду на інформаційну безпеку, конфіденційні ключі доступу зберігаються в конфігурації AddonPreferences, а не в локальному проєкті, що запобігає витоку токенів (рис. 2.8).

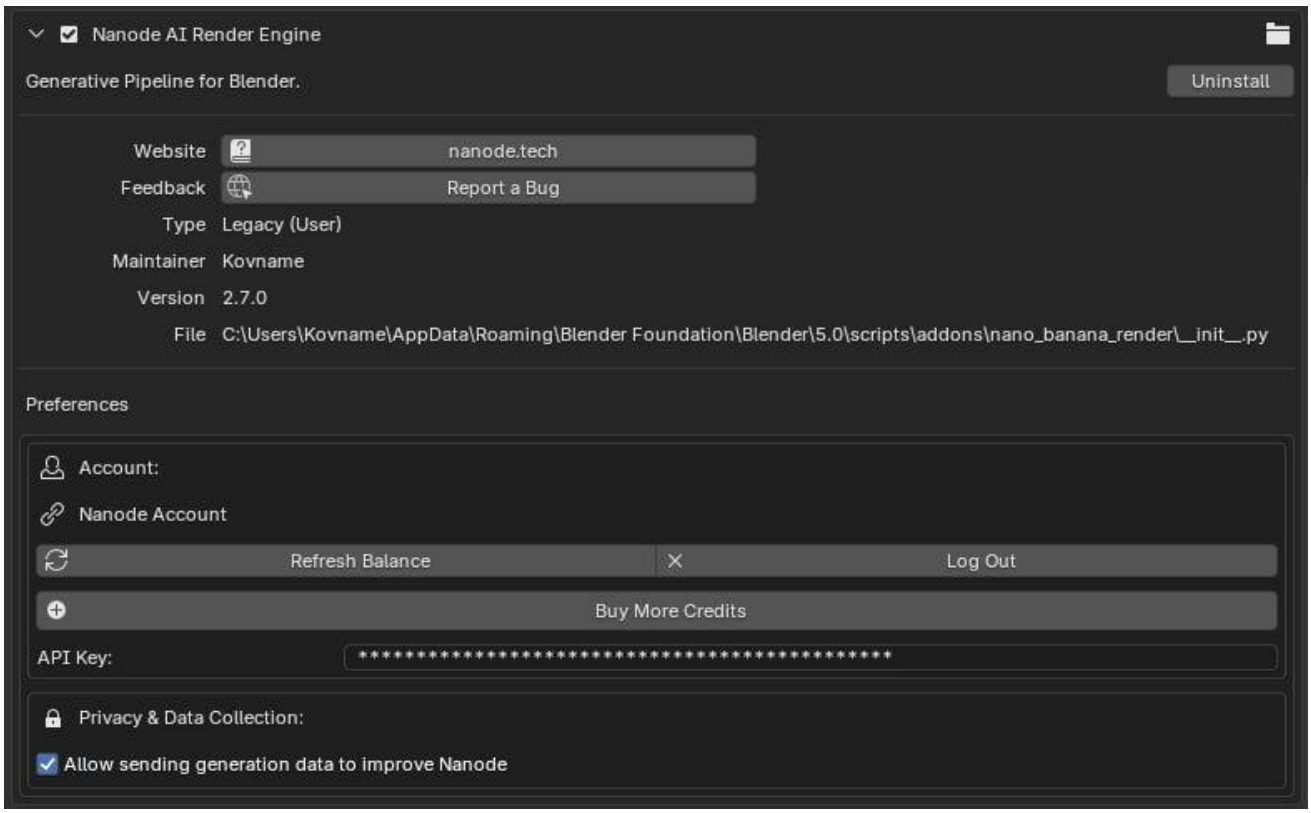


Рисунок 2.8 – Глобальні налаштування AddonPreferences

Сучасний пайплайн роботи 3D-художника вимагає гнучкого набору інструментів для локального редагування, точного контролю над розташуванням об’єктів та зручного управління версіями. З цією метою у клієнтській частині (Blender Addon) було реалізовано інтерактивні інструменти: Editor (Inpaint) для

перемальовування, Smart Points для розміщення деталей та розширену систему історії з функцією Smart Restore.

Інструмент Inpaint (внутрішній Editor) дозволяє художнику виділити певну область згенерованого або відрендереного зображення для її локальної перегенерації, залишаючи решту пікселів без змін. Це критично важливо для ітеративного процесу: виправлення артефактів нейромережі, додавання дрібних деталей (наприклад, вікон на будівлі або елементів одягу) чи зміни текстур в конкретній зоні.

З технічної точки зору процес Inpaint в Blender реалізовано через взаємодію вбудованих функцій малювання з API-клієнтом (Рисунок 2.9). Для створення маски використовується вбудований режим Blender «Texture Paint». Художник малює пензлем поверх поточного зображення у Image Editor. Написано спеціальний оператор, що автоматично створює новий data-block (bpy.data.images) з прозорим фоном, який слугує полотном для маски.

Після завершення малювання скрипт перехоплює оригінальне зображення та згенеровану чорно-білу маску. Щоб мінімізувати затримки та уникнути зайвого збереження тимчасових файлів на жорсткий диск, обидва зображення запікаються і конвертуються у формат Base64 безпосередньо в оперативній пам’яті.

Далі формується розширений JSON-payload. До нього входить: базове зображення (init_image), маска (mask_image), текстовий запит (Prompt), що описує бажані зміни в області маски, та параметр Reference image якщо користувач використовував цю функцію. Цей пакет відправляється POST-запитом на FastAPI бекенд.

Інструмент підтримує більш складні сценарії з використанням напрямних масок (скетчингу) для додавання освітлення, кольорових ефектів або специфічних матеріалів (рис. 2.10).

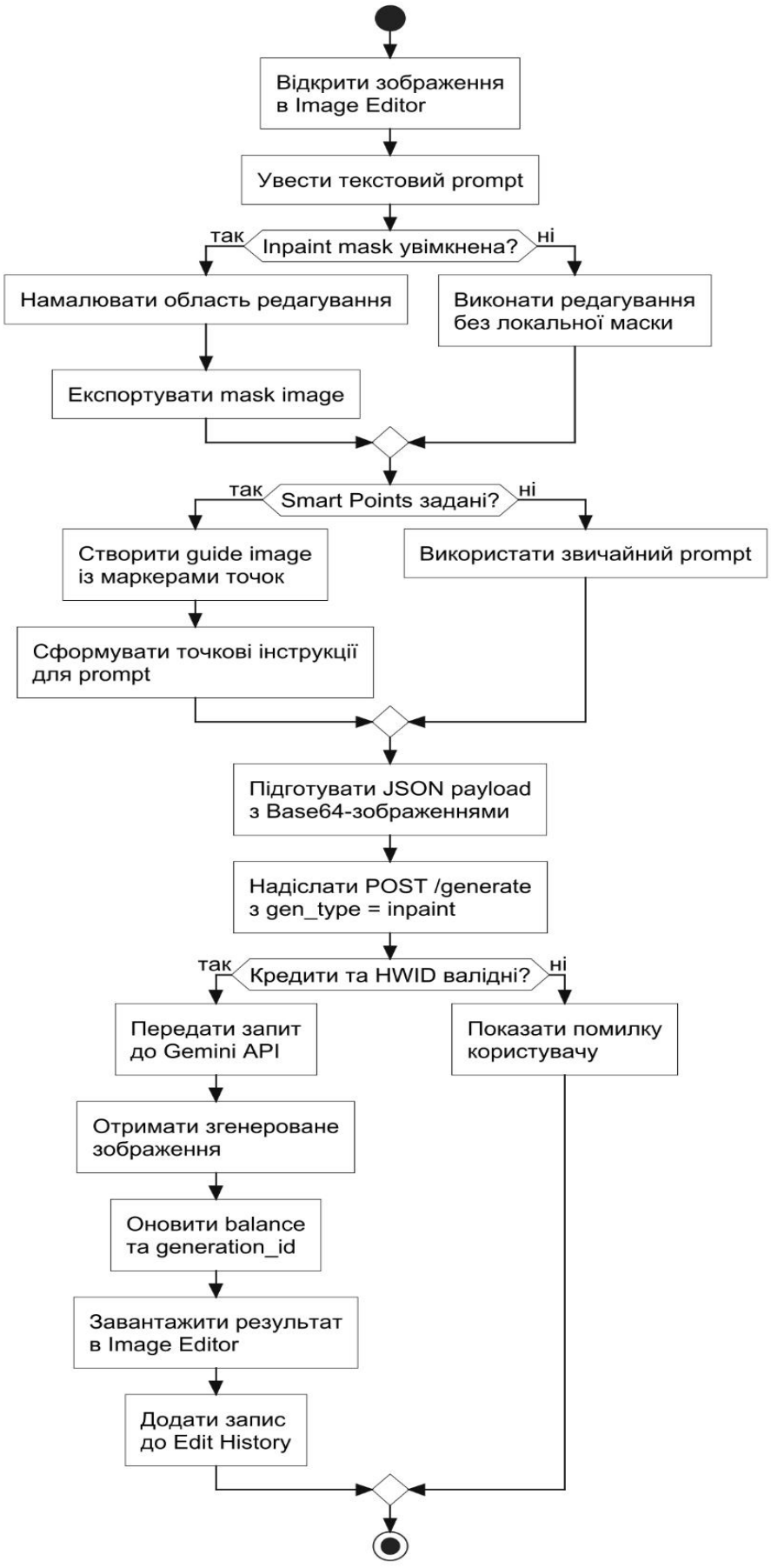


Рисунок 2.9 – Процес генерації Inpaint

Рисунок 2.10 – Використання редактора масок для додавання направленого світла

Для випадків, коли ручне малювання маски є незручним, або коли необхідно точно задати розташування ще не існуючих об’єктів, розроблено алгоритм Smart Points. Користувач може розмістити у 3D-сцені або на 2D-площині спеціальні маркери, які виступатимуть просторовими вказівниками для генерації. Архітектуру та покроковий процес обробки розумних точок наведено на рис. 2.11. Користувач клікає у Viewport або Image Editor. Перехоплюються екранні X/Y координати кліку через інтеграцію з віконною системою Blender.

Потім відбувається конвертація екранних координат у UV-простір (від 0.0 до 1.0) для забезпечення незалежності від поточної роздільної здатності екрану чи рендеру. Модулі gpu та blf малюють поверх вікна Blender інтерактивний маркер (точку), надаючи користувачу візуальний зворотний зв’язок. Утворений масив точок з параметрами (наприклад, кольором) додається у загальний JSON-запит для просторового керування генерацією на стороні сервера.

Наочний приклад повного циклу роботи з інструментом Smart Points наведено на рис. 2.12. Художник розставляє маркери в 3D-просторі, через спеціальну панель аддона призначає кожному маркеру окремий текстовий промпт, після чого сервер генерує об’єкти точно у вказаних позиціях.

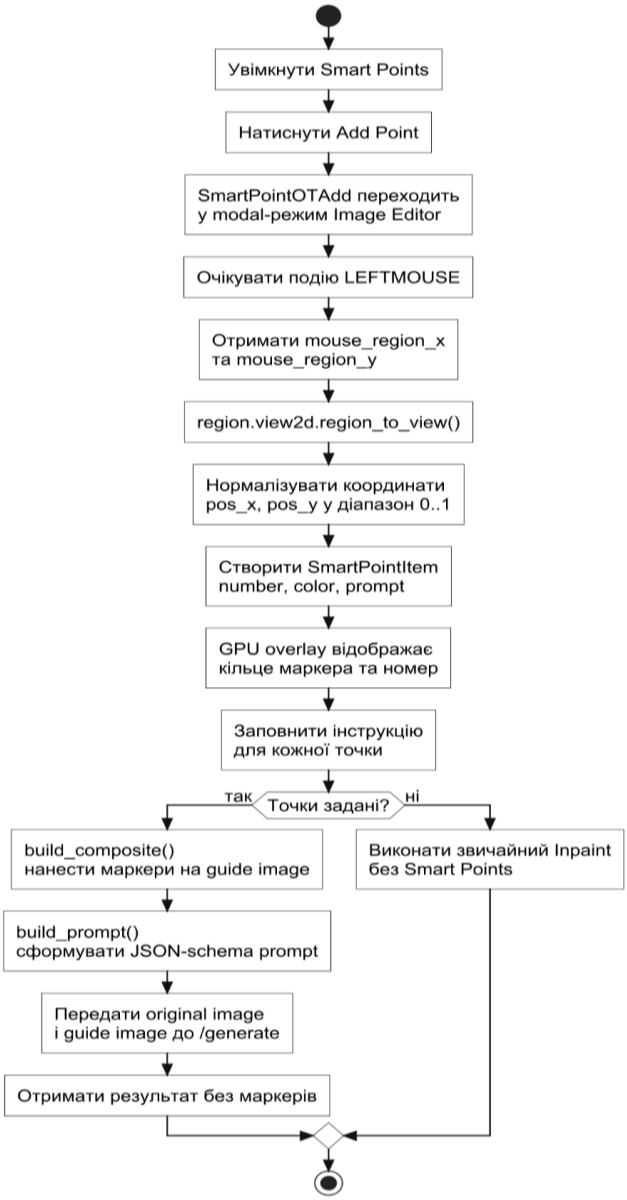


Рисунок 2.11 – Процес захоплення та обробки координат Smart Points

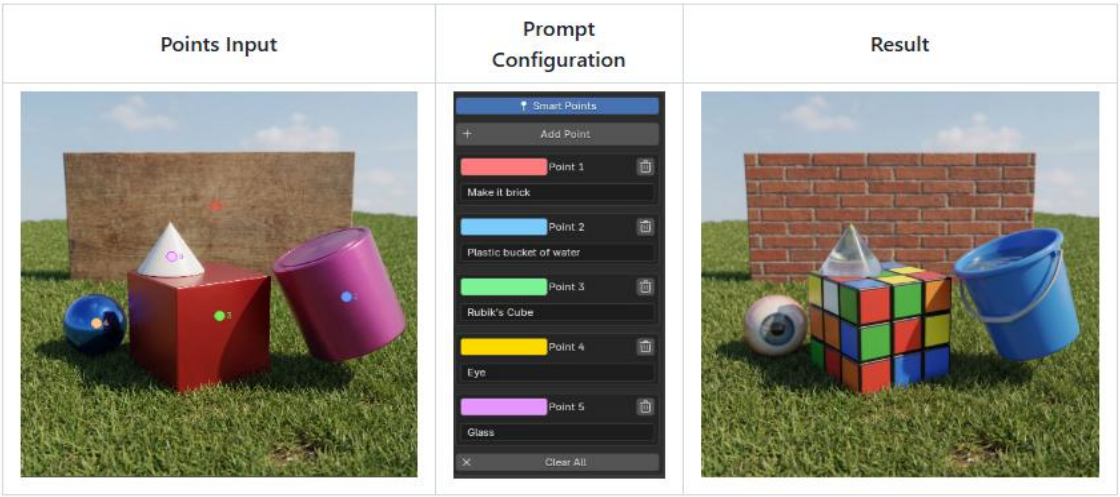


Рисунок 2.12 – Повний цикл роботи Smart Points

Робота з ШІ передбачає створення десятків, а іноді й сотень проміжних варіантів одного рендеру. Щоб художник не втрачав вдалі результати і міг гнучко перемикатися між ними, було спроєктовано та імплементовано просунуту локальну систему управління історією з архітектурою Smart Restore (рис. 2.13).

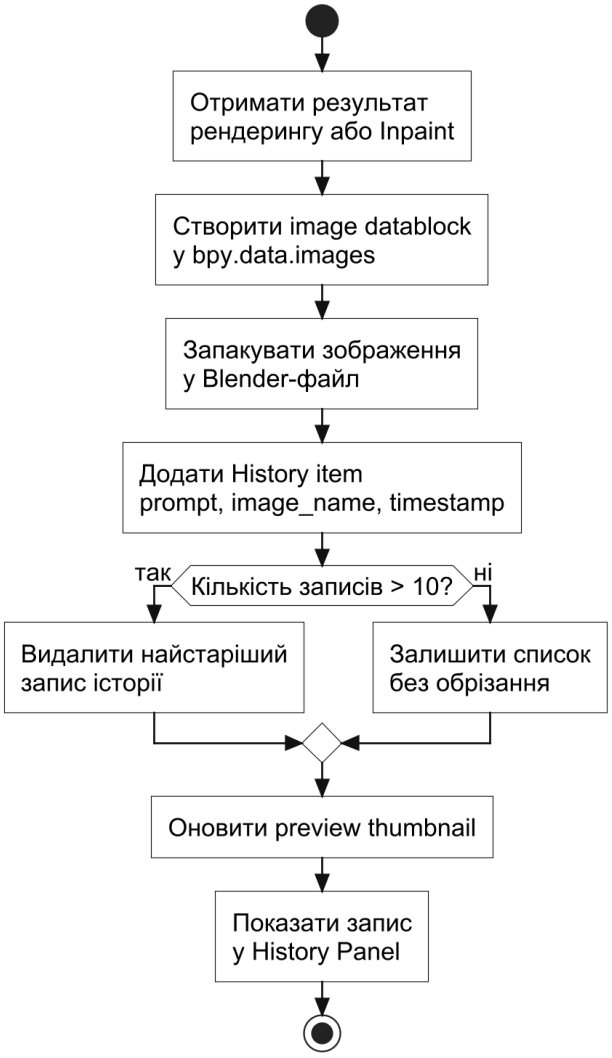


Рисунок 2.13 – Процес роботи системи History

Окрім механізму збереження результатів, система History підтримує набір користувацьких дій над окремим записом історії. Логіку вибору дії, відкриття результату, повторного використання prompt та видалення запису подано у вигляді UML-діаграми активності на рисунку 2.14.

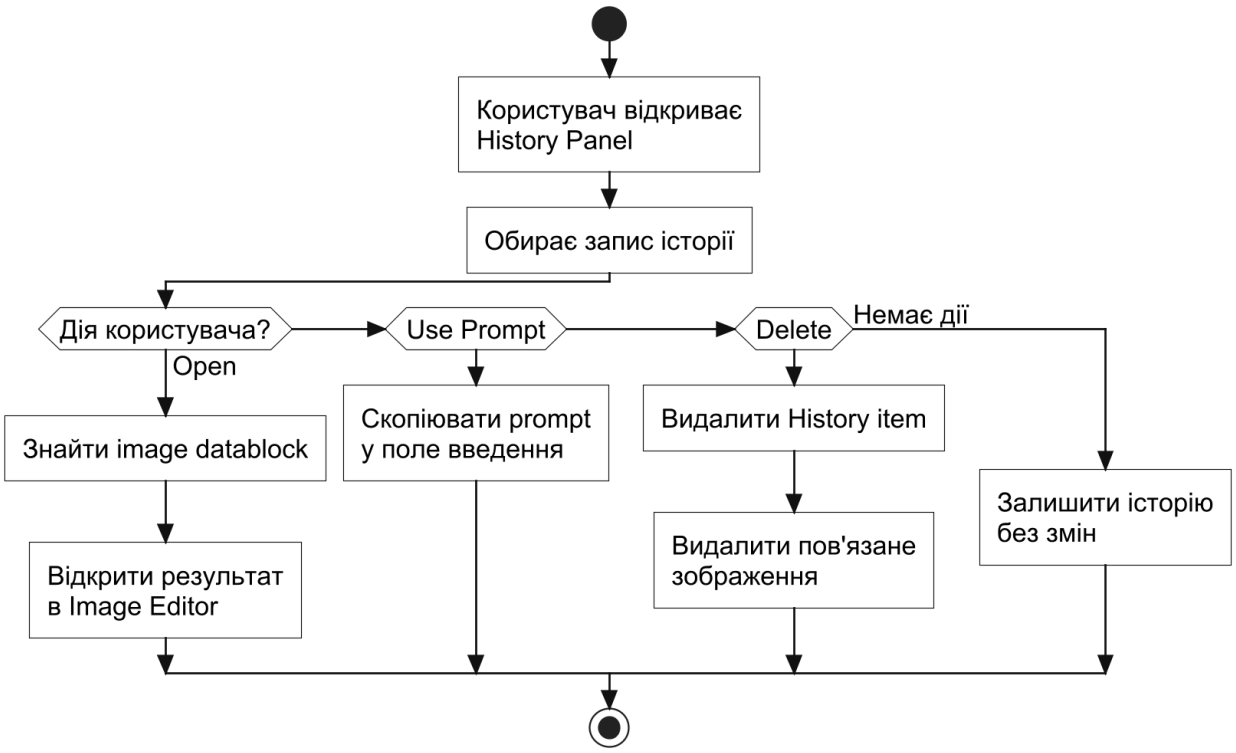


Рисунок 2.14 – UML-діаграма активності дій користувача в системі History

Особливістю цієї системи є те, що вона не просто зберігає зображення, а робить зліпок (snapshot) усього стану аддона на момент генерації. Кожна завершена генерація автоматично зберігається у спеціальну системну папку на диску. Паралельно із зображенням записується JSON-файл метаданих. Він містить повний набір параметрів: Prompt, використану модель, та Reference Image.

За допомогою модуля API bpy.utils.previews створюється кастомна колекція зображень. Вона виводиться у вигляді інтерактивної сітки безпосередньо в панель аддону, дозволяючи візуально обирати минулі генерації. При кліку на мініатюру з історії активується скрипт відновлення. Він десеріалізує прив’язаний JSON та автоматично заповнює всі текстові поля і слайдери у користувацькому інтерфейсі аддона тими ж параметрами, що були використані для даного зображення.

Інтерфейс системи історії (Render Gallery) інтегрований у панель аддона (рис. 2.15). Він виводить список усіх збережених генерацій разом із

мініатюрами, датою створення та частиною текстового промпту. Також передбачено контекстне меню для управління конкретним знімком.

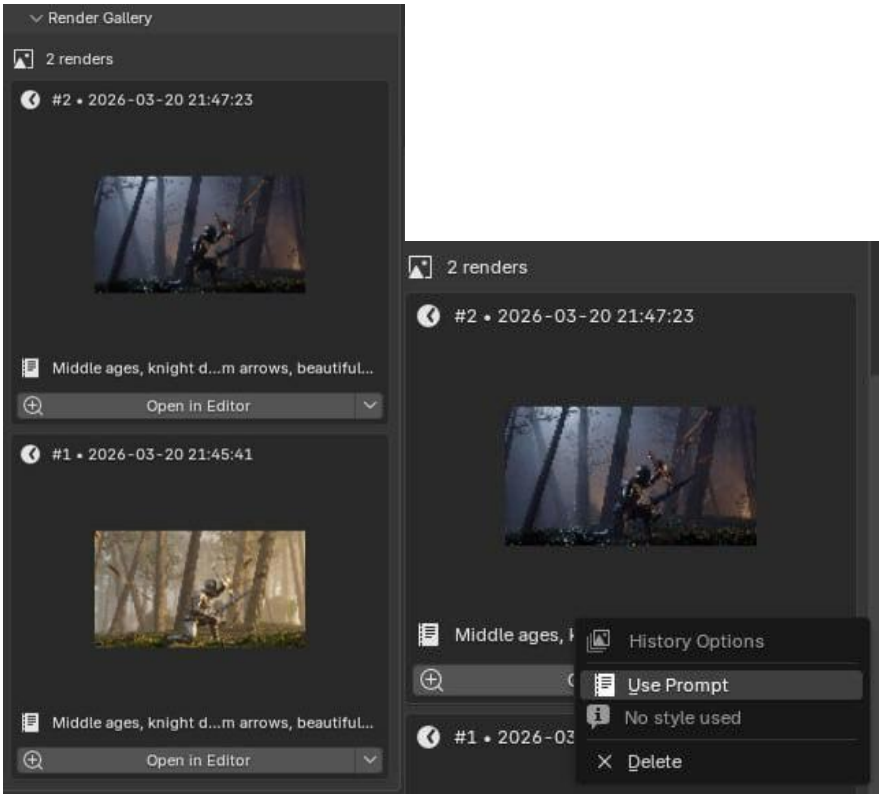


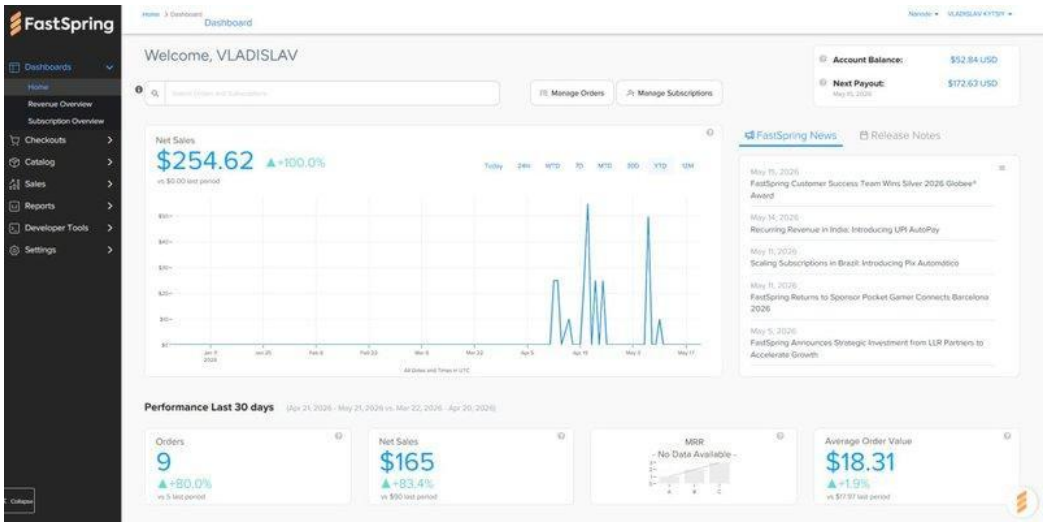
Рисунок 2.15 – Інтерфейс галереї у форматі сітки мініатюр та контекстне меню Smart Restore

Серверна частина платформи розроблена на високопродуктивному асинхронному фреймворку FastAPI. Бізнес-логіка розподілена на модулі для забезпечення легкої підтримки коду. Основним сховищем даних є реляційна база даних SQLite, до якої сервер підключається за допомогою асинхронного драйвера (AIOSQLite). У базі зберігаються дані про користувачів, їхні баланси (кредити), апаратні ідентифікатори (HWID) та історія транзакцій.

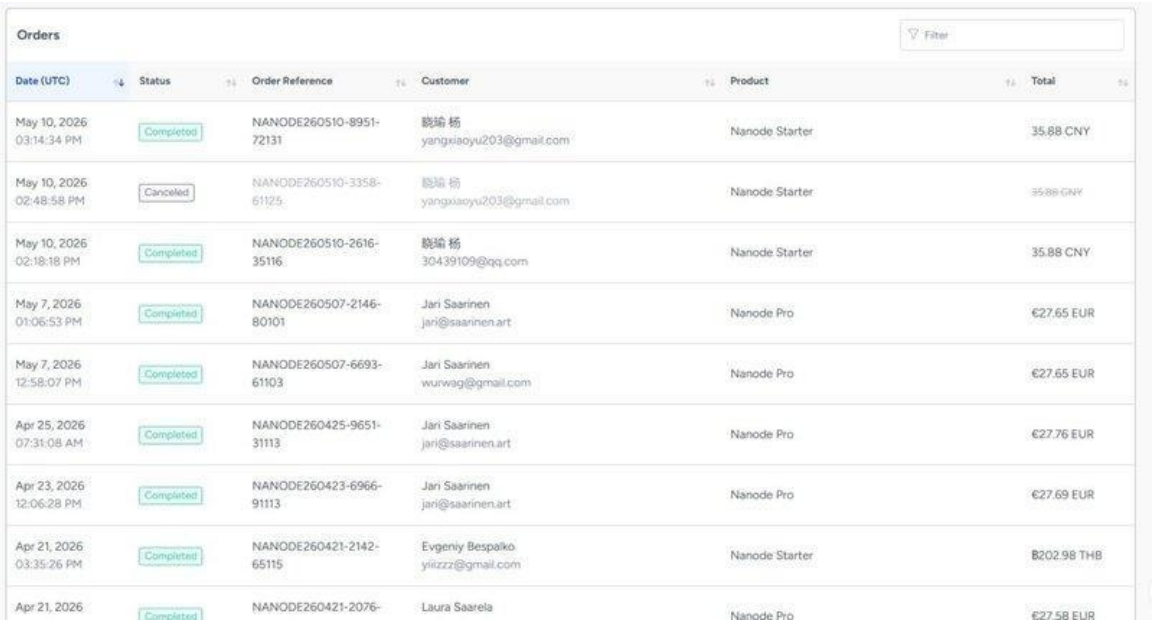
Ключовим елементом монетизації продукту є інтеграція з платіжним провайдером FastSpring [11] (Merchant of Record). FastSpring [11] бере на себе відповідальність за збір податків (ПДВ/VAT) у різних країнах, що критично для легального функціонування глобального сервісу. Інтеграція виконана двома шляхами: на фронтенді (через Storefront Builder Library) для відображення попап-

вікна оплати, та на бекенді через механізм Webhooks для підтвердження транзакцій.

Всі фінансові операції, включно з моніторингом вхідних транзакцій, детальною історією замовлень та управлінням поверненнями коштів (Refunds), здійснюються безпосередньо через безпечну панель керування платформи FastSpring [11] (рис. 2.16).



(а)



(б)

Рисунок 2.16 – Дашборд FastSpring [11]: а) статистика продажів та доходу; б) моніторинг вхідних транзакцій та історії замовлень

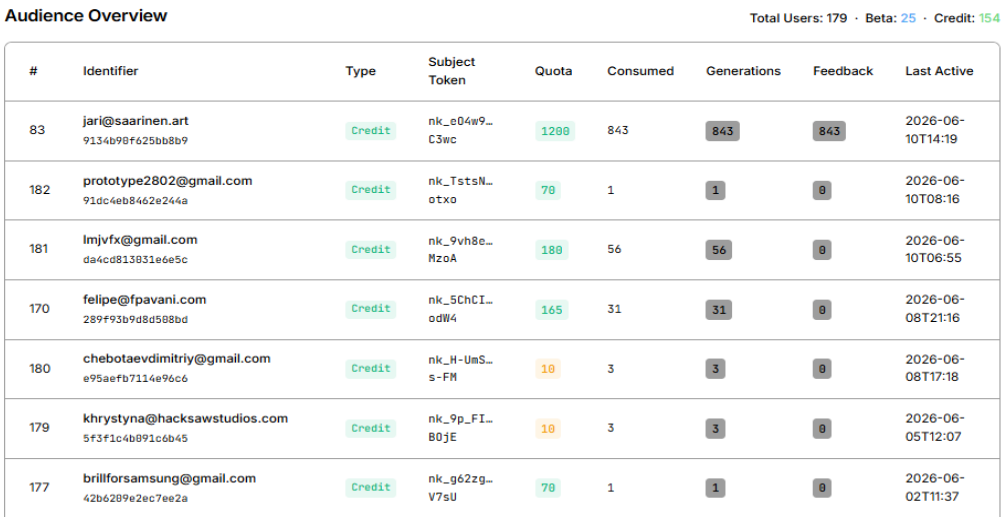
Процес обробки платежу виглядає так. Після успішного списання коштів з картки клієнта, FastSpring [11] надсилає асинхронний POST-запит (вебхук) на захищений ендпоінт /webhook FastAPI-сервера. Цей запит містить детальний JSON-пейлоад з інформацією про транзакцію (сума, email клієнта, кількість куплених кредитів). Для захисту від шахрайства та підміни даних (Spoofing) реалізовано перевірку криптографічного підпису. Сервер зчитує секретний ключ (HMAC Secret), хешує тіло вебхука за алгоритмом SHA-256 і порівнює результат із заголовком X-FS-Signature, що надійшов від FastSpring [11]. Лише у разі їх повного збігу відбувається оновлення балансу користувача у базі даних.

Для комплексного управління екосистемою було розроблено панель адміністратора, яка є невід'ємною частиною бекенду та рендериться за допомогою шаблонізатора Jinja2. Панель має декілька ключових модулів: моніторинг аудиторії (Audience & Users, рис. 2.17а), де можна переглядати активних користувачів та їхні баланси; систему зворотного зв'язку, яка акумулює відгуки безпосередньо з Blender-аддону (рис. 2.17б); та підсистему безпеки (Access Management).

Безпека платформи від аб'юзу (реєстрації множинних безкоштовних акаунтів для отримання стартових кредитів) реалізована через глибоку прив'язку до апаратного забезпечення користувача (HWID). Під час генерації зображення через Blender, локальний Python-скрипт збирає ідентифікатори процесора (CPU ID) та материнської плати (Baseboard Serial), генерує з них унікальний SHA-256 хеш та відправляє на сервер разом із зображенням та JWT-токеном. FastAPI бекенд зіставляє цей HWID з базою заблокованих пристроїв (HWID Blocks). Цей механізм дозволяє миттєво відсікати шахрайський трафік та захищати API-квоти Google Gemini.

Загалом, у другому розділі кваліфікаційної роботи було здійснено проєктування, розробку та імплементацію програмного комплексу «Nano Banana Render (Nanode)», архітектура якого подолала обмеження локального рендерингу, виявлені в першому розділі. Було створено надійну мікросервісну архітектуру, яка логічно розділяє ресурсомісткі обчислення на три незалежні

вузли (клієнт, API Gateway, вебінтерфейс). Реалізація серверної частини на асинхронному фреймворку FastAPI та її розгортання у середовищі Google Cloud Run забезпечили автоматичне масштабування та високу відмовостійкість. Розроблений клієнтський додаток для Blender інтегровано як повноцінний нативний рушій рендерингу. Для забезпечення стабільної роботи інтерфейсу під час мережевих запитів був реалізований алгоритм випереджального захоплення кадру, що ефективно усунув проблему блокування головного потоку програми.

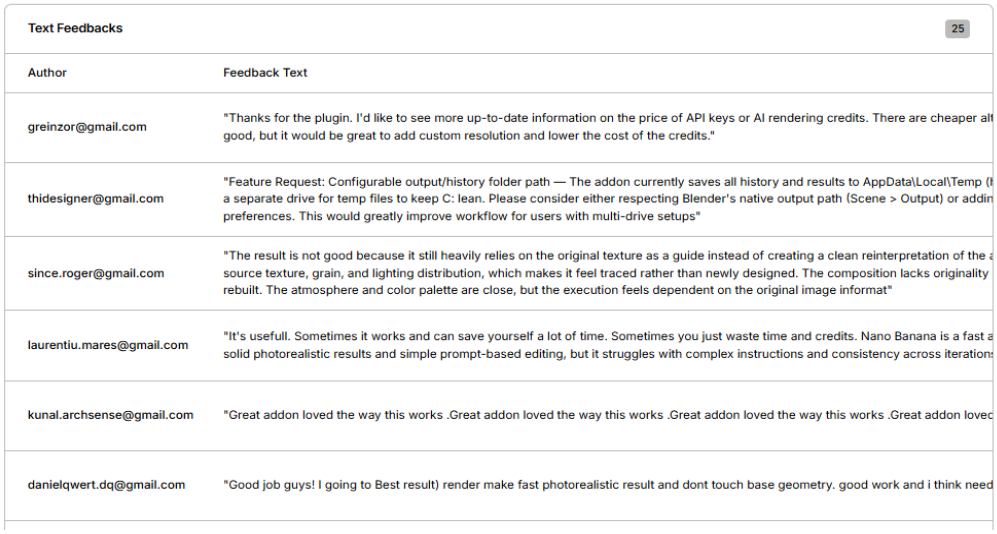


Audience Overview

Total Users: 179 · Beta: 25 · Credit: 154

#	Identifier	Type	Subject Token	Quota	Consumed	Generations	Feedback	Last Active
83	jari@saarinen.art 9134b90f625bb8b9	Credit	nk_e04w9… C3wc	1200	843	843	843	2026-06-10T14:19
182	prototype2802@gmail.com 91dc4eb8462e244a	Credit	nk_TstsN… otxo	70	1	1	0	2026-06-10T08:16
181	lmjvfx@gmail.com da4cd813031e6e5c	Credit	nk_9vh8e… MzoA	180	56	56	0	2026-06-10T06:55
170	felipe@fpavani.com 289f93b9d8d588bd	Credit	nk_5ChCI… odW4	165	31	31	0	2026-06-08T21:16
180	chebotaevdimitriy@gmail.com e95aefb7114e96c6	Credit	nk_H-UmS… s-FM	10	3	3	0	2026-06-08T17:18
179	khrystyna@hacksawstudios.com 5f3f1c4b891c6b45	Credit	nk_9p_FI… B0jE	10	3	3	0	2026-06-05T12:07
177	brillforsamsung@gmail.com 42b6289e2ec7ee2a	Credit	nk_g62zg… V7sU	70	1	1	0	2026-06-02T11:37

(а)



Text Feedbacks 25

Author	Feedback Text
greinzor@gmail.com	"Thanks for the plugin. I'd like to see more up-to-date information on the price of API keys or AI rendering credits. There are cheaper alt… good, but it would be great to add custom resolution and lower the cost of the credits."
thidesigner@gmail.com	"Feature Request: Configurable output/history folder path — The addon currently saves all history and results to AppData\Local\Temp (… a separate drive for temp files to keep C: lean. Please consider either respecting Blender's native output path (Scene > Output) or addin… preferences. This would greatly improve workflow for users with multi-drive setups"
since.roger@gmail.com	"The result is not good because it still heavily relies on the original texture as a guide instead of creating a clean reinterpretation of the … source texture, grain, and lighting distribution, which makes it feel traced rather than newly designed. The composition lacks originality … rebuilt. The atmosphere and color palette are close, but the execution feels dependent on the original image informat"
laurentiu.mares@gmail.com	"It's usefull. Sometimes it works and can save yourself a lot of time. Sometimes you just waste time and credits. Nano Banana is a fast a… solid photorealistic results and simple prompt-based editing, but it struggles with complex instructions and consistency across iteration…
kunal.archsense@gmail.com	"Great addon loved the way this works .Great addon loved the way this works .Great addon loved the way this works .Great addon lovec…
danielqwert.dq@gmail.com	"Good job guys! I going to Best result) render make fast photorealistic result and dont touch base geometry. good work and i think need…

(б)

Рисунок 2.17 – Панель адміністратора: а) керування базою користувачів та їх лімітами; б) відображення зворотного зв’язку з Blender-клієнта

Імплементовано ключовий інструментарій для художнього контролю: локальне редагування згенерованих зображень, використовуючи вбудований режим Texture Paint для створення масок, які конвертуються у Base64-формат для відправки на сервер. Створено алгоритм позиціонування, що дозволяє користувачу розміщувати маркери в 3D-сцені для точного просторового керування генерацією об’єктів.

РОЗДІЛ 3 ТЕСТУВАННЯ ТА РОЗГОРТАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Статичний аналіз та аудит безпеки коду

Сучасний життєвий цикл розробки програмного забезпечення неможливий без глибокого автоматизованого контролю якості коду (Code Quality Assurance) та аудиту безпеки. Для забезпечення найвищого рівня стабільності системи Nano Banana Render (Nanode), яка об’єднує локальний клієнт Blender Addon та хмарний мікросервіс FastAPI у Google Cloud Run, було застосовано платформу статичного аналізу SonarCloud [23] (рис. 3.1). Даний інструмент був безшовно інтегрований у CI/CD конвеєр на базі GitHub Actions [24]. При кожному коміті (push) у гілку main або створенні Pull Request, ініціюється процес глибокого сканування абстрактного синтаксичного дерева (AST), що виявляє паттерни вразливостей, витоки пам’яті та так звані «запахи коду».

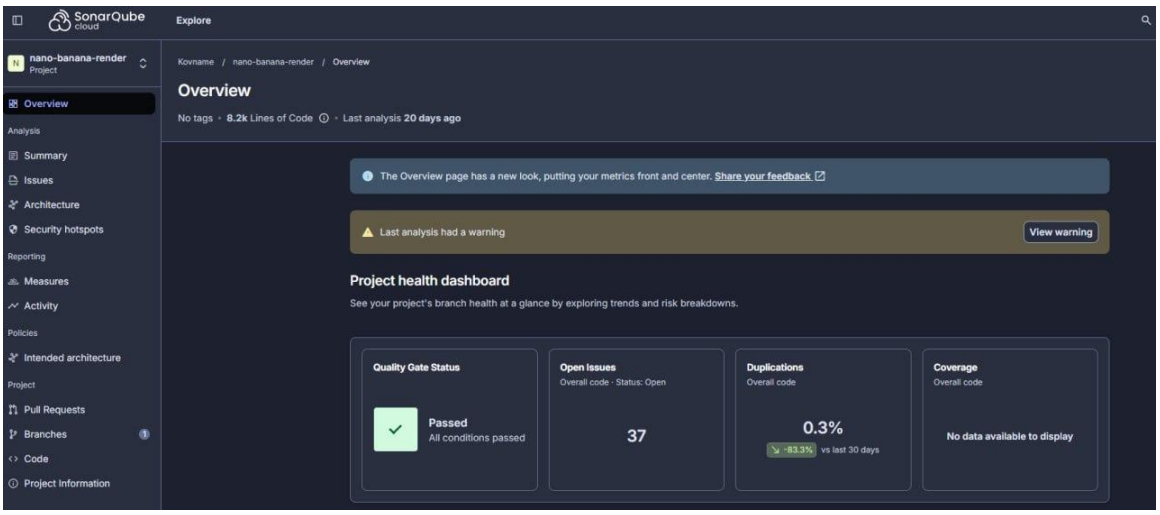
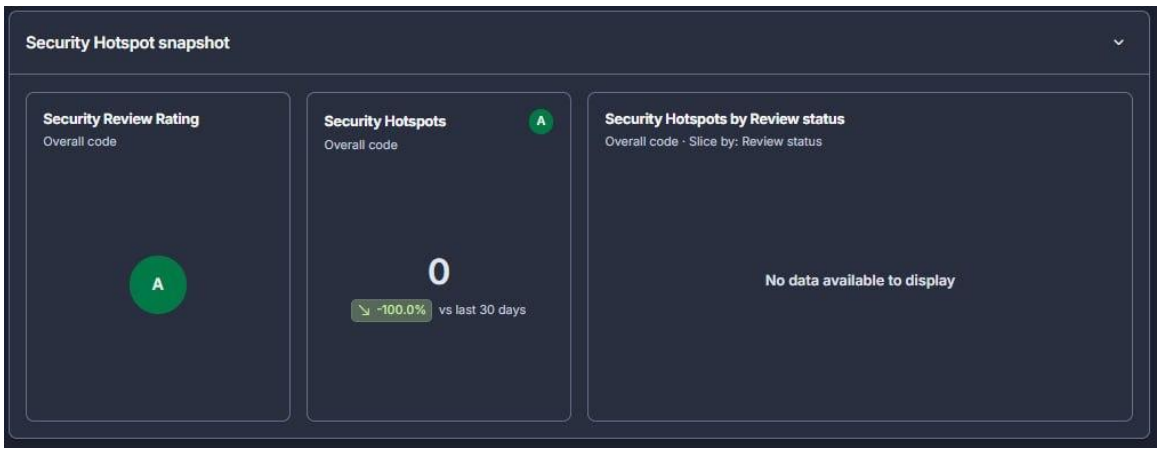


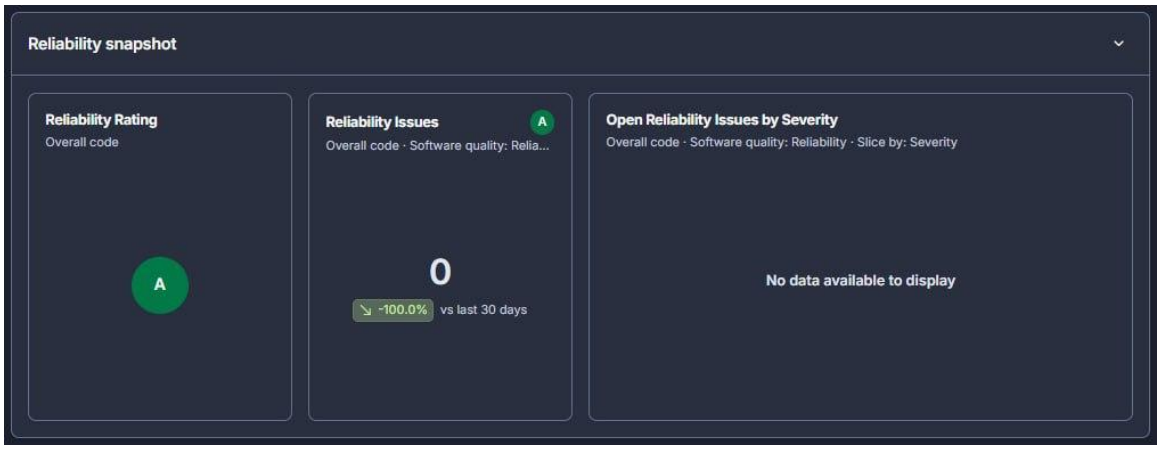
Рисунок 3.1 – Загальний огляд проєкту та метрик якості у SonarCloud [23]

Ключовими метриками, на які зверталася особлива увага під час аналізу, стали показники безпеки та надійності. Як видно з результатів сканування, кодова база отримала найвищу оцінку «A» за критерієм безпеки – 0 Security Hotspots (нуль потенційних вразливостей, рис. 3.2а). Цей показник є критично важливим для архітектури системи. Оскільки бекенд обробляє чутливі дані,

включаючи JWT-токени авторизації через Google OAuth 2.0 [10], а також приймає та верифікує платіжні вебхуки від еквайрингової системи FastSpring [11] (перевірка криптографічних підписів HMAC SHA-256), відсутність «гарячих точок» безпеки підтверджує, що реалізовані Pydantic-моделі та Middleware повністю захищені від типових атак, таких як SQL Injection, XSS чи CSRF.



(а)



(б)

Рисунок 3.2 – Деталізація показників: а) безпеки – відсутність вразливостей; б) надійності системи

Ще одним досягненням стала метрика дублювання коду (Duplication), яка склала всього 0.4%. Це свідчить про суворе дотримання принципу DRY (Don’t Repeat Yourself) та високу модульність архітектури. Усі базові операції, такі як

мережеві запити (google_proxy.py), генерація карти глибин (depth_utils.py) та обробка матеріалів (texture_pipeline.py), винесені в ізольовані сервісні класи.

Варто детально розглянути метрику супроводжуваності (Maintainability). Згідно зі звітом SonarCloud [23] (рис. 3.3), у клієнтському коді на Python було виявлено 37 зауважень, всі з яких стосуються перевищення порогу когнітивної складності (Cognitive Complexity). Наприклад, статичний аналізатор вимагав знизити складність окремих функцій з 17 до рекомендованих 15 балів шляхом рефакторингу.

Рисунок 3.3 – Показники супроводжуваності та технічного боргу

Однак, у контексті розробки доповнень під Blender (використання бібліотеки bpy [25]), такий рефакторинг є архітектурно недоцільним і навіть небезпечним. Специфіка середовища Blender полягає в тому, що всі модифікації 3D-сцени повинні виконуватися в єдиному строгому контексті (Context Overrides). Функції змушені виконувати глибоку навігацію по ієрархічному дереву об’єктів (наприклад, звернення bpy.data.objects[...].modifiers[...].node_tree.nodes[...]). Спроба розбити ці монолітні блоки логіки на дрібні ізольовані функції, як того вимагає класичний принцип єдиної відповідальності (SRP), неминуче призводить до втрати контексту виконання між викликами функцій. У середовищі Blender це викликає критичні збої та помилки доступу до пам’яті ядра (Access Violation C0000005), що призводить до «вильоту» програми. Таким чином, залишення дещо завищеної

когнітивної складності є свідомим архітектурним компромісом, спрямованим на забезпечення абсолютної стабільності роботи застосунку на апаратних ресурсах користувача.

3.2 Автоматизація процесів постачання програмного забезпечення

Завершальним етапом реалізації платформи стало розгортання серверної частини (Deployment) на хмарній інфраструктурі. На відміну від класичних VPS, бекенд упаковано у Docker-контейнер. Створений Dockerfile описує всі залежності (requirements.txt), копіює код FastAPI-додатку та запускає сервер за допомогою Uvicorn. Контейнер збирається та завантажується у Google Artifact Registry, після чого розгортається за допомогою сервісу Google Cloud Run. Такий підхід забезпечує ізоляцію середовища, високу відмовостійкість та автоматичну прив'язку SSL-сертифікатів до домену. Всі секретні ключі (OAuth Secret, FastSpring [8; 9] HMAC Key, Gemini API Keys) безпечно передаються у контейнер у вигляді змінних середовища (Environment Variables), не потрапляючи до публічного репозиторію GitHub.

Для забезпечення безперебійної інтеграції та доставки коду клієнтської частини («Nanode-site») було впроваджено інструментарій GitHub Actions [24]. У кореневій директорії репозиторію налаштовано конвеєри автоматизації (workflows) у папці .github/workflows, зокрема файл deploy.yml. Він конфігурований таким чином, щоб автоматично запускатися при кожному коміті або створенні Pull Request у головну гілку (master). Пайплайн включає підняття віртуального середовища Ubuntu, встановлення залежностей пакетного менеджера (npm install), та безпосередню збірку оптимізованої версії React-додатку за допомогою збирача Vite (npm run build). Після успішного білду згенеровані статичні файли (директорія dist) автоматично вивантажуються (deploy) на безкоштовний хостинг GitHub Pages. Завдяки цій автоматизації було успішно проведено понад 27 ітерацій розгортання у робоче середовище без жодної ручної інтервенції (рис. 3.4), що звело людський фактор до мінімуму.

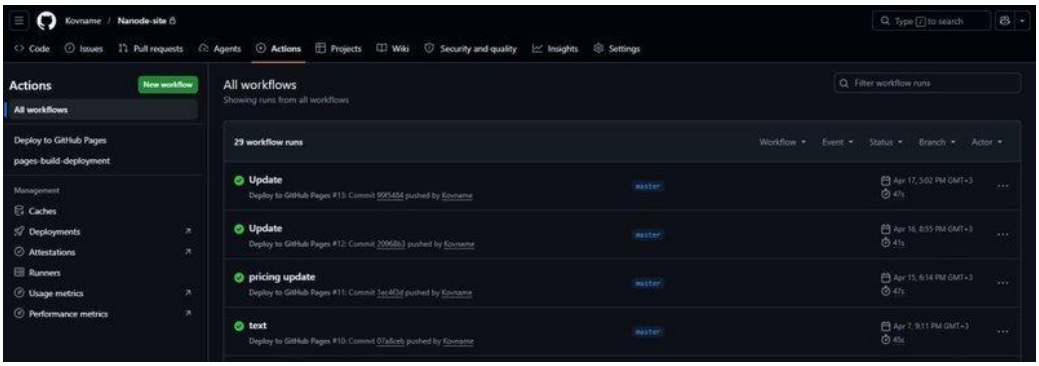


Рисунок 3.4 – Налаштовані CI/CD конвеєри в GitHub Actions [24]

Окрім автоматизації збірки, важливим етапом було налаштування користувацького домену для вебпорталу. У налаштуваннях репозиторію на GitHub Pages (розділ Pages) було підключено кастомний домен nanode.tech. Для коректної маршрутизації на стороні DNS-провайдера створено A-записи, що вказують на IP-адреси серверів GitHub, та CNAME-запис для піддомену www. Додатково в налаштуваннях GitHub Pages було активовано опцію Enforce HTTPS, яка автоматично генерує та оновлює SSL/TLS сертифікати від Let’s Encrypt, забезпечуючи захищене з’єднання для всіх відвідувачів сайту.

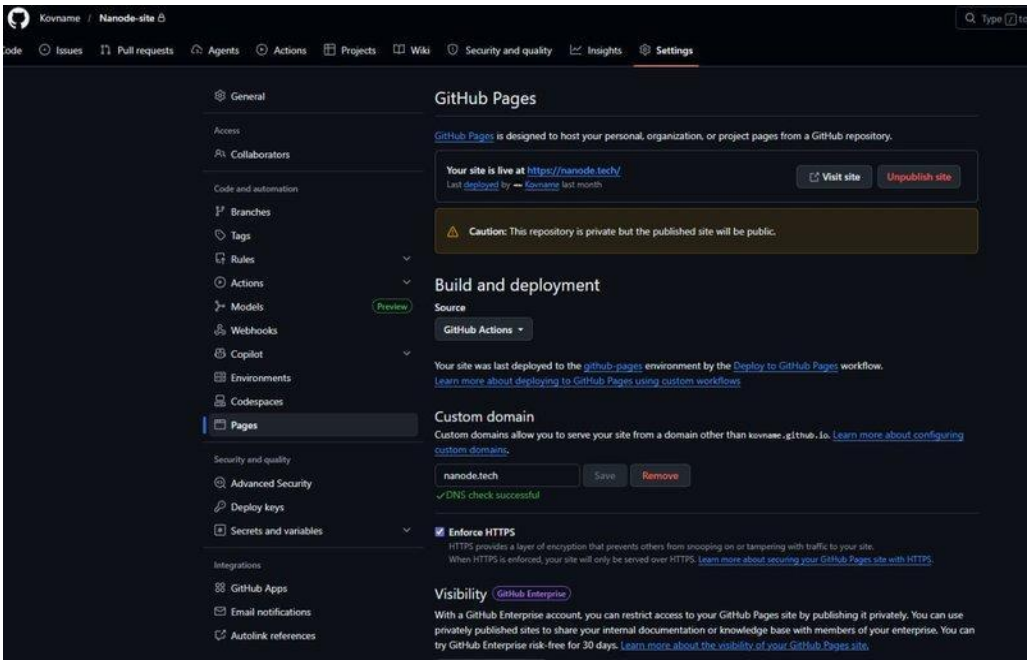


Рисунок 3.5 – Налаштування користувацького домену nanode.tech та HTTPS у GitHub Pages

Для розгортання високопродуктивного бекенду (API) стандартних VPS-серверів було недостатньо, тому я спроектував інфраструктуру на базі контейнеризації (Docker). Аби забезпечити проект надійними хмарними обчисленнями для обробки зображень та взаємодії зі штучним інтелектом, було підготовано архітектурну документацію та подано заявку на участь у престижній програмі «Google for Startups Cloud Program». Заявка була успішно схвалена, що дозволило отримати грант та доступ до потужностей Google Cloud Platform.

Отримавши доступ до хмарної інфраструктури, серверну частину (FastAPI додаток) було розгорнуто з використанням сервісу Google Cloud Run під назвою ресурсу nanode-api у регіоні europe-west4 (Нідерланди), що забезпечує мінімальні затримки (ping) для користувачів з Європи. Використання Cloud Run дозволило повністю реалізувати парадигму Serverless: контейнери з бекендом масштабуються автоматично від нуля (Scale-to-Zero) до необхідної кількості екземплярів залежно від поточного навантаження. Графіки моніторингу в консолі Google Cloud чітко демонструють динаміку використання ресурсів, представлену на рис. 3.6. Такий підхід забезпечує стабільну роботу системи обробки платежів та генерації зображень, а також радикально оптимізує витрати стартапу, оскільки оплата знімається лише за час фактичної обробки запитів.

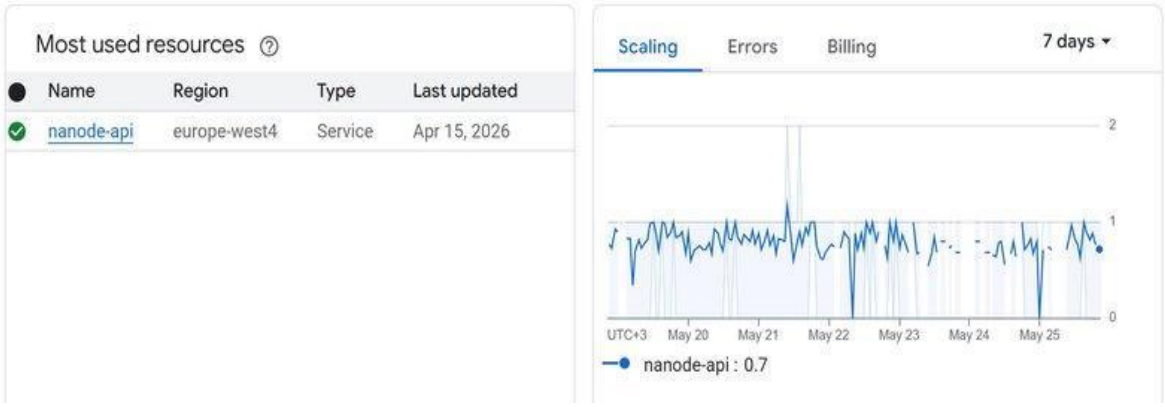


Рисунок 3.6 – Моніторинг сервісу nanode-api у панелі Google Cloud Run: використання ресурсів та автомасштабування

Розробка сучасних систем генеративного штучного інтелекту вимагає постійного налаштування параметрів нейромережі на основі реального користувацького досвіду. Для цього в систему «Nanode AI Render» було імплементовано просунутий модуль телеметрії, що працює за парадигмою RLHF (Reinforcement Learning from Human Feedback – навчання з підкріпленням на основі людських відгуків).

З боку клієнта, в інтерфейсі N-панелі Blender (модуль ui_panel.py), була розроблена система швидкого реагування. Коли хмарний бекенд повертає згенероване зображення, і воно автоматично застосовується як матеріал на 3D-об’єкт, під панеллю прев’ю активуються дві кнопки зворотного зв’язку: «Вдала генерація» (Like) та «Невдала генерація» (Dislike), як показано на рис. 3.7. Кожна кнопка обробляється власним класом, успадкованим від базового bpy.types.Operator, що забезпечує нативну взаємодію з Event Loop самого Blender.

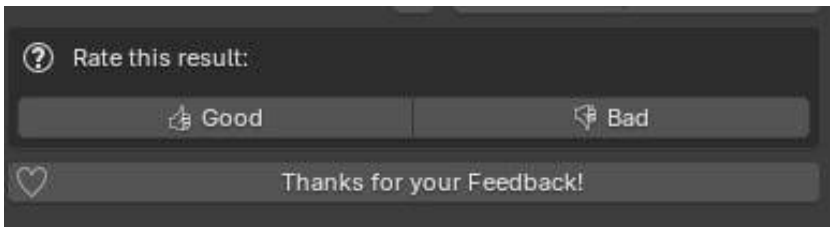


Рисунок 3.7 – Інтерфейс швидкої оцінки результатів генерації (Like/Dislike) у N-панелі Blender

При взаємодії з інтерфейсом оцінки (рис. 3.8), оператор формує структурований JSON-об’єкт, який інкапсулює унікальний ідентифікатор транзакції (Generation UUID), маркер оцінки (+1 для успіху, -1 для невдачі), а також службові дані про використаний промпт і seed-значення. Щоб уникнути ефекту «зависання» інтерфейсу Blender (блокування GUI Main Thread), відправка цього пакета реалізована асинхронно через модуль threading_utils.py. Він створює відокремлений фоновий потік (threading.Thread), який виконує HTTP POST-запит до бекенду за допомогою бібліотеки requests.

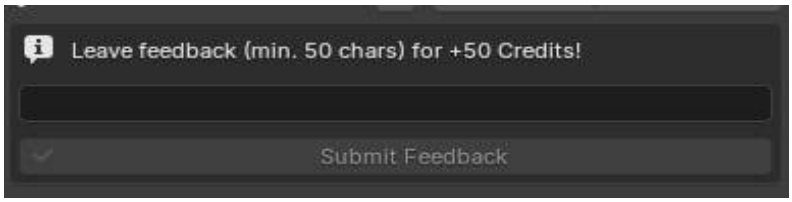


Рисунок 3.8 – Форма деталізованого зворотного зв’язку всередині аддону

На серверній стороні (FastAPI у Google Cloud Run) розроблено RESTful ендпоінт, захищений прошарком авторизації, який перевіряє валідність клієнтського JWT-токена. Це унеможливлює здійснення спам-атак та несанкціоноване забруднення бази даних. Після успішної авторизації, бекенд оновлює запис генерації у базі даних (SQLite через AIOSQLite), фіксуючи результат.

Всі зібрані дані агрегуються та виводяться у розроблений веб-інтерфейс панелі адміністратора (Admin Dashboard). Інтерфейс дозволяє адміністратору переглядати історію генерацій, візуально співставляти введений текстовий промпт із отриманим зображенням (Image Preview) та аналізувати реакцію користувача (оцінки Like/Dislike). Ця аналітична база стала ключовим інструментом для покращення якості сервісу: на її основі проводилось тонке налаштування (Fine-tuning) прихованих системних промптів, що передаються до Gemini API, а також оптимізувалися параметри згладжування та контрастності для карти глибин (Mist Pass), що суттєво зменшило кількість відхилених користувачами результатів. [10; 25]

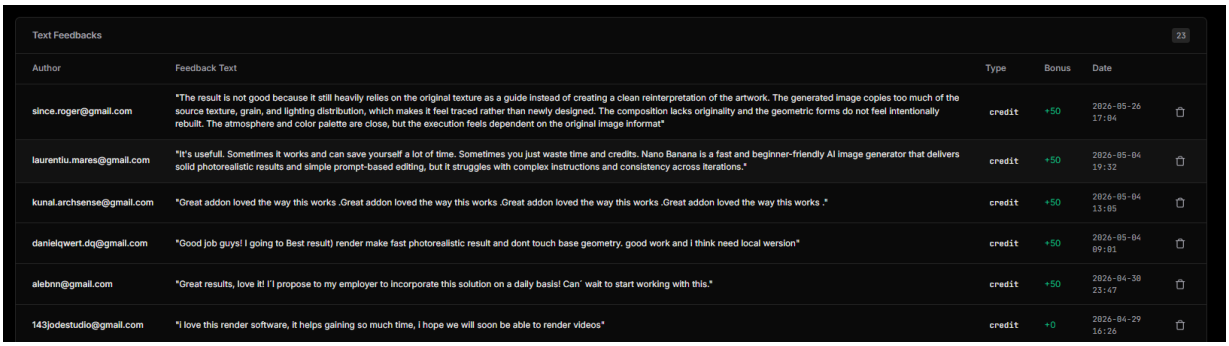


Рисунок 3.9 – Моніторинг та аналітика зібраних відгуків користувачів у панелі адміністратора (Backend)

3.3 Організація та проведення закритого бета-тестування

Невід'ємним етапом перед офіційним комерційним релізом програмного продукту є проведення фази закритого бета-тестування. Головною метою цього етапу було виявлення крайніх випадків використання в реальних умовах, стрес-тестування хмарної інфраструктури та валідація UX-рішень клієнтського аддону.

Для формування репрезентативної фокус-групи було організовано збір заявок через платформу Google Forms (рис. 3.10). Цільовою аудиторією виступали виключно професійні 3D-художники, які інтегрують інструменти ШІ у свої пайплайни. У результаті було відібрано 22 незалежних бета-тестери. Щоб забезпечити їх інструментарієм, бекенд автоматично згенерував кожному учаснику унікальний JWT-токен доступу (API Key), який надавав ліміт на 100 безкоштовних генерацій з максимальною роздільною здатністю 4K.

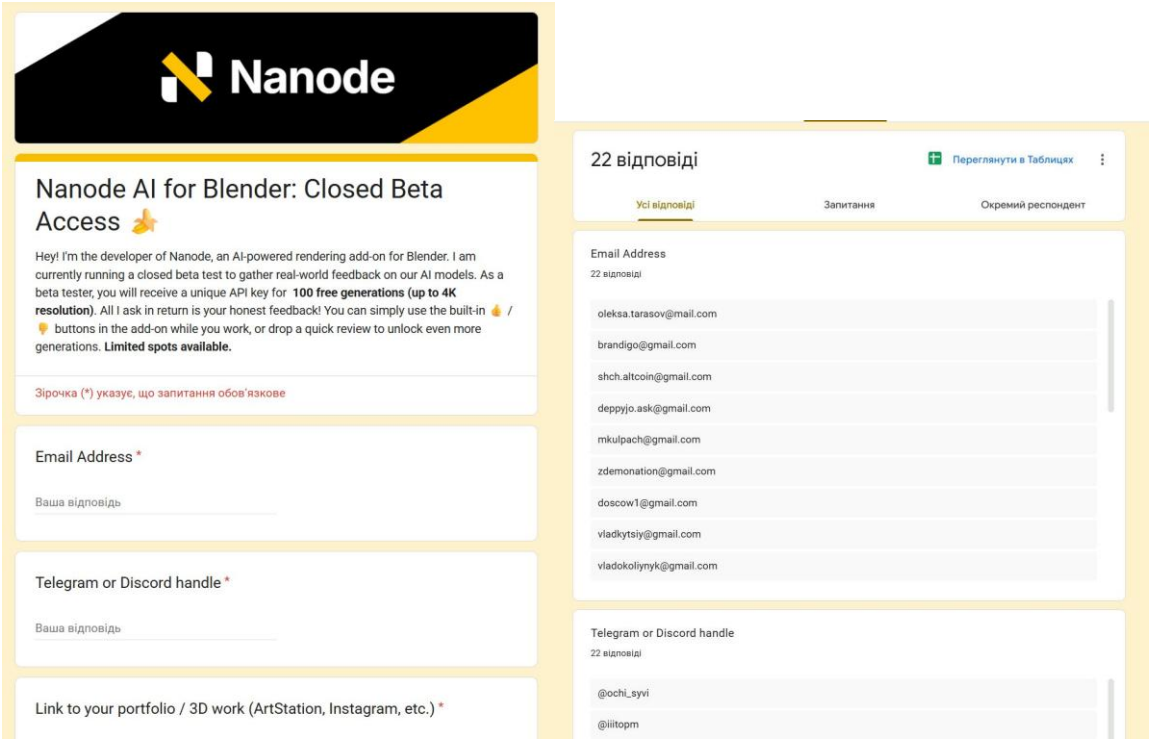


Рисунок 3.10 – Формування фокус-групи закритого бета-тестування серед 3D-художників (22 учасники)

Під час двотижневого інтенсивного тестування було зібрано цінний масив емпіричних даних та баг-репортів. Серед критичних зауважень, які вдалося ізолювати та виправити завдяки CBT, були такі:

– таймаути при холодному старті інстансів Google Cloud Run: через архітектуру Scale-to-Zero перші запити до бекенду іноді відхилялися за таймаутом; вирішено шляхом оптимізації механізму Keep-Alive та перероблення архітектури клієнтського опитування статусу рендеру в Blender;

– галюцінації в редакторі в складних сценах: виявлено проблеми при редагуванні; вирішено шляхом архітектурного вдосконалення JSON-промпту.

Усунення цих інфраструктурних та логічних вразливостей дозволило досягти винятково стабільної роботи клієнтської частини (Crash-free rate > 99%) до моменту офіційного запуску платформи.

3.4 Апробація результатів розроблення

Важливим показником успішності та актуальності будь-якого сучасного IT-рішення є його об’єктивне сприйняття професійною спільнотою розробників. Клієнтська частина доповнення «Nanode AI Render» була опублікована на платформі GitHub за ліцензією Open Source. Такий підхід стимулював стрімке зростання органічної аудиторії: репозиторій отримав понад 85 відміток (Stars) та численні форки (Forks), що беззаперечно підтверджує високу затребуваність розроблених інструментів.

Проєкт також привернув увагу профільних IT-експертів світового рівня та отримав офіційне визнання та персональну відзнаку від Стерена Джаніні (Steren Giannini), продуктового директора Google Cloud, який є одним із ключових візіонерів архітектури Cloud Run. Публічна підтримка проєкту з боку керівництва Google Cloud є переконливим доказом якості та корпоративного рівня спроєктованої архітектури (відмовостійкість, масштабування, інформаційна безпека). Крім того, завдяки цій експертній взаємодії, проєкт успішно пройшов валідацію та був відібраний до міжнародної програми екосистеми стартапів Google for Startups Cloud Program.

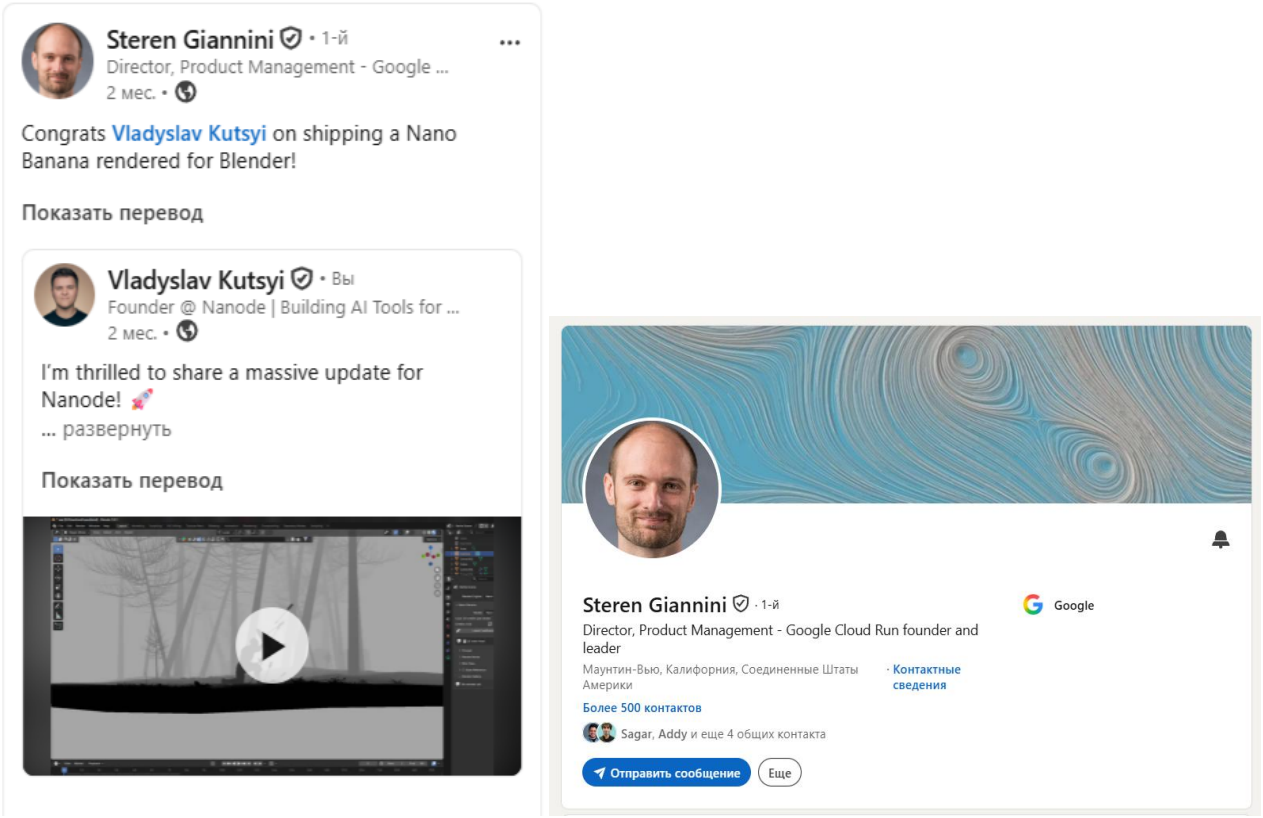


Рисунок 3.11 – Публічна рецензія архітектури проєкту від Директора з продукту Google Cloud

Отриманий фінансово-інфраструктурний грант (рис. 3.12) забезпечив проєкту необхідні хмарні потужності для безперебійного безпечного проксіювання запитів (Secure Proxying) до магістральної нейромережі Gemini API. Це повністю покрило витрати на забезпечення високої доступності (High Availability) під час пікових сплесків активності користувачів відразу після релізу продукту.

Загалом, у третьому розділі проведено комплексне тестування, моніторинг та апробацію програмного комплексу «Nano Banana Pro Render» з метою верифікації його стабільності, безпеки та відповідності технічним вимогам. Отримані результати підтверджують високу якість реалізованого рішення. Інтеграція статичного аналізатора SonarCloud [23] у конвеєр CI/CD підтвердила еталонні показники надійності. Свідоме збереження підвищеної когнітивної складності окремих модулів обґрунтовано жорсткими архітектурними обмеженнями Blender API.

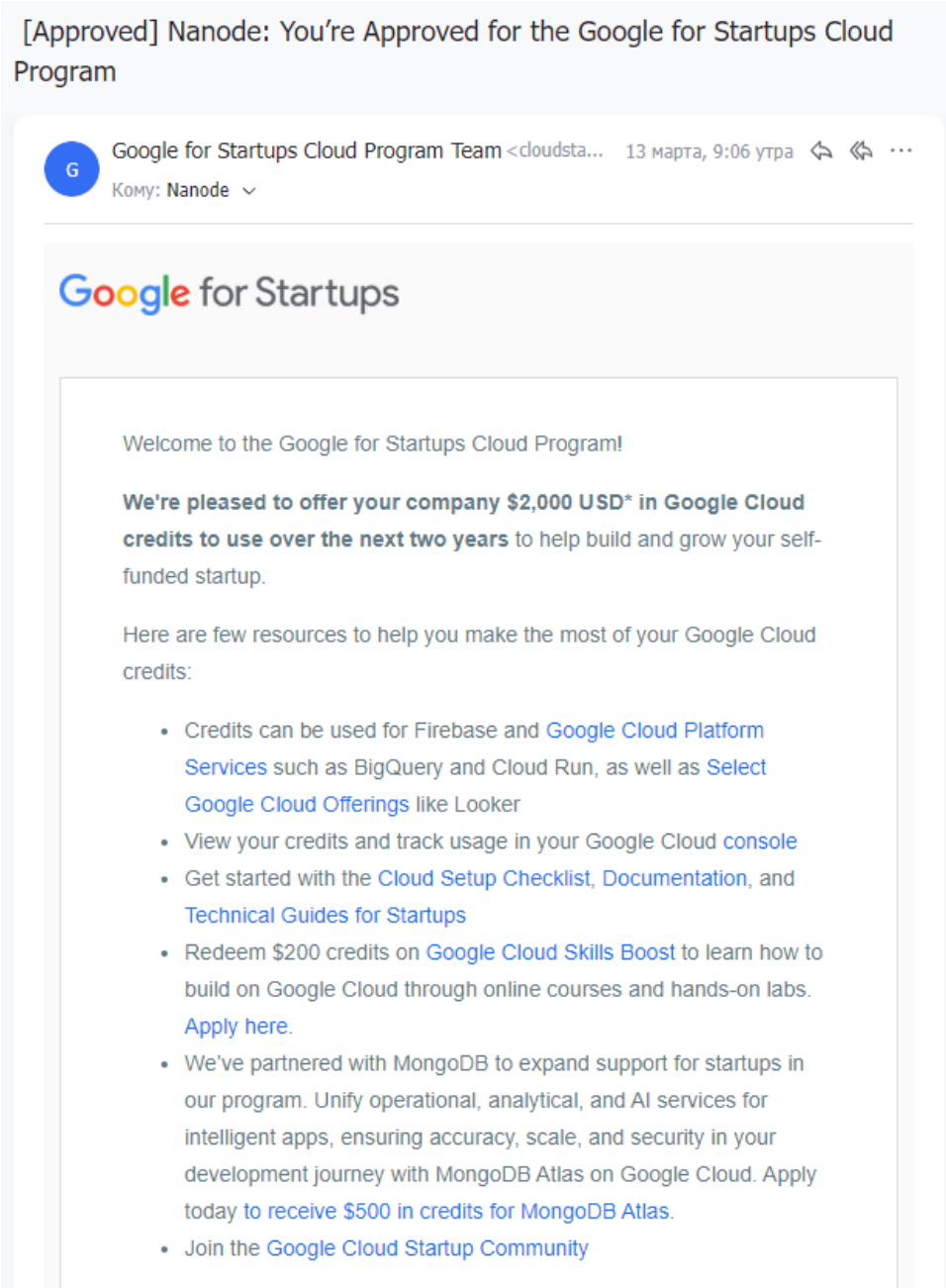


Рисунок 3.12 – Отримання інфраструктурного гранту Google for Startups Cloud Program

Розроблена асинхронна система телеметрії забезпечила збір якісного фідбеку (Like/Dislike) безпосередньо з інтерфейсу аддону. Це дозволило провести тонке налаштування системних промптів та оптимізувати параметри карти глибин. Проведене закрите бета-тестування за участю 22 професійних 3D-художників дозволило виявити та успішно усунути критичні вразливості,

включаючи таймаути архітектури Cloud Run при холодному старті та помилки Inpaint.

Досягнутий технічний рівень отримав підтвердження від зовнішньої IT-спільноти та корпоративних експертів. Успішна валідація дозволила отримати інфраструктурний грант Google for Startups Cloud Program, що забезпечило безперебійне та безпечне функціонування хмарної інфраструктури для проксіювання запитів Gemini API.

ВИСНОВКИ

У кваліфікаційній роботі вирішено прикладне завдання підвищення ефективності процесу візуалізації 3D-сцен шляхом розроблення програмного забезпечення для середовища Blender, що поєднує структурні дані тривимірної сцени з можливостями генеративних нейронних мереж. У першому розділі проаналізовано сучасний стан технологій комп'ютерної графіки, класичного та нейронного рендерингу. Встановлено, що традиційні методи трасування шляхів забезпечують високу фізичну достовірність зображення, однак залишаються ресурсоємними та залежать від продуктивності локального апаратного забезпечення. Окрему увагу приділено проблемам використання відеопам'яті, складності налаштування матеріалів, тривалості ітераційного перегляду результатів і високому порогу входження для користувачів без професійної підготовки у сфері CGI. Аналіз генеративних підходів показав, що локальні дифузійні моделі частково знімають проблему швидкості, але створюють нові обмеження: потребу у значному обсязі VRAM, складність інсталяції та ризик втрати відповідності між згенерованим зображенням і геометрією сцени.

У другому розділі виконано проєктування програмного комплексу «Nano Banana Render (Nanode)». Розроблено клієнт-серверну структуру, у якій клієнтська частина функціонує як розширення Blender, а серверна частина відповідає за обробку запитів, взаємодію з генеративною моделлю, автентифікацію та керування інфраструктурними процесами.

Практичним результатом роботи стала реалізація програмного модуля, інтегрованого у Blender за допомогою Python API. Реалізовано механізми отримання структурних даних сцени, формування допоміжних зображень і використання карти глибини як одного з ключових засобів просторового керування генерацією. Окремо реалізовано алгоритм Smart Points, який дозволяє уточнювати просторове розміщення об'єктів у сцені та підвищувати керованість результату порівняно з генерацією лише за текстовим описом.

У процесі реалізації особливу увагу приділено стабільності взаємодії користувача з Blender. Застосування алгоритму випереджального захоплення

кадру дало змогу уникнути блокування інтерфейсу під час виконання мережевих запитів і забезпечити більш передбачувану роботу розширення. Також реалізовано інструменти інтерактивного редагування результатів, зокрема локальне доопрацювання зображення та механізм керування історією змін, що підвищує зручність роботи з кількома варіантами генерації.

У третьому розділі проведено тестування, статичний аналіз і підготовку програмного продукту до розгортання. Перевірка якості коду дала змогу оцінити його підтримуваність, надійність і безпеку. Автоматизація процесів постачання програмного забезпечення забезпечила відтворюваність розгортання та спростила оновлення серверної частини. Окремо розглянуто питання автентифікації користувачів, захисту API-квот, адміністрування доступу та інтеграції зовнішніх сервісів.

Отримані результати підтверджують доцільність використання гібридного підходу до рендерингу, за якого 3D-сцена не замінюється повністю генеративною моделлю, а використовується як джерело просторових, композиційних і візуальних обмежень. Це дозволяє зберегти контроль над геометрією сцени та водночас скористатися перевагами генеративного штучного інтелекту для прискорення створення фотореалістичних візуалізацій.

Отже, мету кваліфікаційної роботи досягнуто: розроблено програмний комплекс для гібридного нейронного рендерингу 3D-сцен у середовищі Blender, що використовує карти глибини, хмарні генеративні моделі та клієнт-серверну архітектуру для підвищення швидкості, доступності й керованості процесу візуалізації. Водночас робота має напрями для подальшого розвитку. Доцільним є розширення набору підтримуваних структурних карт сцени, удосконалення механізмів контролю відповідності результату початковій геометрії, запровадження детальнішої телеметрії якості генерації та розширення інструментів локального редагування. Перспективним також є додавання режимів пакетної обробки сцен, підтримка анімаційних послідовностей і гнучкіше налаштування параметрів взаємодії з різними генеративними моделями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Grand View Research. 3D Rendering Market Size, Share & Trends Analysis Report, 2026–2033. URL: https://www.grandviewresearch.com/industry-analysis/3d-rendering-market-report (дата звернення: 02.06.2026).
2. Fact.MR. 3D Rendering Market Outlook (2026–2036). URL: https://www.factmr.com/report/3d-rendering-market (дата звернення: 02.06.2026).
3. Rombach R., Blattmann A., Lorenz D., Esser P., Ommer B. High-Resolution Image Synthesis with Latent Diffusion Models. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 2022. P. 10684–10695. URL: https://arxiv.org/abs/2112.10752 (дата звернення: 02.06.2026).
4. Zhang L., Rao A., Agrawala M. Adding Conditional Control to Text-to-Image Diffusion Models. ICCV 2023. URL: https://arxiv.org/abs/2302.05543 (дата звернення: 02.06.2026).
5. Google AI for Developers. Gemini API Documentation. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 02.06.2026).
6. Google AI for Developers. Gemini API: Image Generation (Nano Banana). URL: https://ai.google.dev/gemini-api/docs/image-generation (дата звернення: 02.06.2026).
7. Google AI for Developers. Gemini API: Image Understanding. URL: https://ai.google.dev/gemini-api/docs/image-understanding (дата звернення: 02.06.2026).
8. FastAPI. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 02.06.2026).
9. Google Cloud. Cloud Run Documentation. URL: https://cloud.google.com/run/docs (дата звернення: 02.06.2026).
10. Google Identity. Using OAuth 2.0 to Access Google APIs. URL: https://developers.google.com/identity/protocols/oauth2 (дата звернення: 12.06.2026).

11. FastSpring. Merchant of Record Platform Overview. URL: https://fastspring.com/merchant-of-record/ (дата звернення: 02.06.2026).
12. Blender Foundation. Cycles: Introduction. Blender Manual. URL: https://docs.blender.org/manual/en/latest/render/cycles/introduction.html (дата звернення: 02.06.2026).
13. Kajiya J. T. The Rendering Equation. Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH ’86). 1986. P. 143–150. DOI: https://doi.org/10.1145/15922.15902.
14. Blender Foundation. GPU Rendering. Blender Manual. URL: https://docs.blender.org/manual/en/latest/render/cycles/gpu_rendering.html (дата звернення: 02.06.2026).
15. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 4th ed. Cambridge: MIT Press, 2023. URL: https://pbr-book.org/ (дата звернення: 02.06.2026).
16. NVIDIA. GeForce RTX 3090 Family Specifications. URL: https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/ (дата звернення: 02.06.2026).
17. Cybernews. RAM prices skyrocket as AI demand grows. 2025. URL: https://cybernews.com/tech/ram-prices-skyrocket-as-ai-demand-grows/ (дата звернення: 12.06.2026).
18. Intel. Open Image Denoise Documentation. URL: https://www.openimagedenoise.org/documentation.html (дата звернення: 02.06.2026).
19. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 02.06.2026).
20. Dosovitskiy A., Beyer L., Kolesnikov A., Weissenborn D., Zhai X., Unterthiner T., Dehghani M., Minderer M., Heigold G., Gelly S., Uszkoreit J., Houlsby N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

ICLR 2021. URL: https://arxiv.org/abs/2010.11929 (дата звернення: 02.06.2026).

21. Katri C. Dream Textures: Stable Diffusion built-in to Blender. GitHub repository. URL: https://github.com/carson-katri/dream-textures (дата звернення: 02.06.2026).
22. Stability AI. Stability Blender Addon Public. GitHub repository. URL: https://github.com/Stability-AI/stability-blender-addon-public (дата звернення: 02.06.2026).
23. SonarSource. Understanding Quality Gates: SonarQube Cloud Documentation. URL: https://docs.sonarsource.com/sonarqube-cloud/standards/managing-quality-gates/introduction-to-quality-gates/ (дата звернення: 02.06.2026).
24. GitHub Docs. Continuous Integration with GitHub Actions. URL: https://docs.github.com/en/actions/get-started/continuous-integration (дата звернення: 02.06.2026).
25. Blender Foundation. Blender Python API Documentation. URL: https://docs.blender.org/api/current/ (дата звернення: 02.06.2026).

ДОДАТКИ

Додаток А – Посилання на репозиторій

Репозиторій з програмною реалізацією, виконаною в межах кваліфікаційної роботи, розташовано за адресою https://github.com/Kovname/nano-banana-render