ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Автоматизація етапів розробки комп'ютерних ігор за допомогою використанням інтелектуальних моделей генерації контенту»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим БІДЗЮРА
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-63
Максим БІДЗЮРА

Керівник: Максим КУКЛІНСЬКИЙ
канд. техн. наук, доц.

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бідзюрі Максиму Максимовичу

1. Тема кваліфікаційної роботи: «Автоматизація етапів розробки комп'ютерних ігор за допомогою використанням інтелектуальних моделей генерації контенту»

керівник кваліфікаційної роботи Максим КУКЛІНСЬКИЙ, професор кафедри ІТ, канд. техн. наук, доцент

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література з питань генеративного штучного інтелекту (GAN, Diffusion Models), документація бібліотек PyTorch та Diffusers, вимоги до графічних ассетів у сучасних ігрових рушіях (Unity/Unreal Engine), набори даних для тестування генерації текстур.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз сучасних підходів та інтелектуальних моделей генерації контенту в ігровій індустрії.

2. Розробка методу автоматизованої генерації безшовних PBR-текстур на основі латентних дифузійних моделей.
3. Математичне моделювання процесів дифузії та оцінки якості згенерованих зображень.
4. Програмна реалізація системи генерації та експериментальне дослідження її ефективності порівняно з традиційними методами.

5. Перелік ілюстративного матеріалу: *презентація*
    1. Математична модель дифузійного процесу
    2. Структурна схема запропонованого методу
    3. Алгоритм генерації карт нормалей
    4. Математичне забезпечення пост-обробки
    5. Критерії оцінки ефективності
    6. Практичний результат
    7. Практичний результат
    8. Аналіз ефективності розробки

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-07.11	
2	Обґрунтування вибору методів дослідження (Latent Diffusion Models)	07.11-08.11	
3	Розробка математичної моделі та алгоритму генерації безшовних текстур	08.11-15.11.2025	
4	Проєктування архітектури програмного забезпечення	15.11-18.11.2025	
5	Програмна реалізація прототипу системи генерації (Python, Diffusers)	18.11-20.11.2025	
6	Розробка модуля генерації карт нормалей (Normal Maps)	20.11-22.11.2025	
7	Проведення експериментальних досліджень та порівняльний аналіз	22.11-25.11.2025	
8	Оформлення роботи: вступ, висновки, реферат	25.11-17.12.2025	
9	Розробка демонстраційних матеріалів	17.12-19.12.2025	

Здобувач вищої освіти _____ Максим БІДЗЮРА

(підпис)

Керівник

кваліфікаційної роботи _____ Максим КУКЛІНСЬКИЙ

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 78 стор., 3 табл., 31 рис., 31 джерел.

Мета роботи - підвищення ефективності та скорочення часових витрат на етапах створення графічного контенту для комп’ютерних ігор шляхом розробки автоматизованого методу на основі латентних дифузійних моделей.

Об’єкт дослідження - процес генерації та обробки візуальних активів (ассетів) у розробці комп’ютерних ігор.

Предмет дослідження - метод автоматизованої генерації безшовних текстур та карт нормалей з використанням технологій глибокого навчання.

У роботі використано різноманітні методи, такі як системний аналіз предметної області, математичне моделювання дифузійних процесів, методи глибокого навчання (Deep Learning) для синтезу зображень та алгоритми комп'ютерного зору для оцінки глибини сцени.

Проведено аналіз сучасних методів процедурної генерації контенту та порівняльний аналіз архітектур нейронних мереж GAN, VAE та Diffusion Models.

Розроблено та оптимізовано гібридний метод створення PBR-матеріалів, що поєднує генерацію зображень за текстовим описом, автоматичне забезпечення безшовності (tiling) та створення карт нормалей.

Проведено експерименти для оцінки швидкодії запропонованого методу на споживчому обладнанні та перевірки якості згенерованих ассетів порівняно з ручним створенням.

КЛЮЧОВІ СЛОВА: КОМП’ЮТЕРНІ ІГРИ, ГЕНЕРАЦІЯ КОНТЕНТУ, ГЛИБОКЕ НАВЧАННЯ, STABLE DIFFUSION, PBR-ТЕКСТУРИ, КАРТИ НОРМАЛЕЙ, АВТОМАТИЗАЦІЯ, GAMEDEV.

ABSTRACT

Text part of the master's qualification work: 78 pages, 3 pictures, 31 table, 31 sources.

The purpose of the work is to increase efficiency and reduce time costs at the stages of creating graphic content for computer games by developing an automated method based on latent diffusion models.

Object of research - the process of generating and processing visual assets in computer game development.

Subject of research - the method of automated generation of seamless textures and normal maps using deep learning technologies.

Summary of the work: Various methods such as system analysis of the subject area, mathematical modeling of diffusion processes, Deep Learning methods for image synthesis, and computer vision algorithms for depth estimation were used in the work.

An analysis of modern approaches to procedural content generation and a comparative analysis of GAN, VAE, and Diffusion Models neural network architectures were conducted. A hybrid method for creating PBR materials has been developed and optimized, combining text-to-image generation, automatic tiling, and normal map creation. Experiments were conducted to evaluate the performance of the proposed method on consumer hardware and to verify the quality of generated assets compared to manual creation.

KEYWORDS: COMPUTER GAMES, CONTENT GENERATION, DEEP LEARNING, STABLE DIFFUSION, PBR TEXTURES, NORMAL MAPS, AUTOMATION, GAMEDEV.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AAA - Клас високобюджетних комп'ютерних ігор (Triple-A).

AI - Штучний інтелект (Artificial Intelligence).

API - Інтерфейс прикладного програмування (Application Programming Interface).

CLIP - Контрастне попереднє навчання "мова-зображення" (Contrastive Language-Image Pre-training).

CNN - Згорткова нейронна мережа (Convolutional Neural Network).

DDPM - Імовірнісні дифузійні моделі знешумлення (Denoising Diffusion Probabilistic Models). FID - Відстань початкової інцепції Фреше (Fréchet Inception Distance).

GAN - Генеративно-змагальна мережа (Generative Adversarial Network).

GPU - Графічний процесор (Graphics Processing Unit).

GUI - Графічний інтерфейс користувача (Graphical User Interface).

LDM - Латентна дифузійна модель (Latent Diffusion Model).

MPS - Шейдери продуктивності Metal (Metal Performance Shaders).

MSE - Середньоквадратична помилка (Mean Squared Error).

PBR - Фізично коректний рендеринг (Physically Based Rendering).

PCG - Процедурна генерація контенту (Procedural Content Generation).

RGB - Адитивна колірна модель (Red, Green, Blue).

UV - Координати двовимірної текстури в тривимірному просторі (U and V axes).

VAE - Варіаційний автокодувальник (Variational Autoencoder).

VRAM - Відеопам'ять (Video Random Access Memory).

ВСТУП

З інтенсивним розвиток індустрії розваг і зростанням вимог до візуальної якості цифрових продуктів постає гостра необхідність вдосконалення процесів створення контенту. Однією з ключових галузей, що потребує інновацій, є розробка комп'ютерних ігор, де створення графічних активів (асетів) займає левову частку часу та бюджету.

Впровадження інтелектуальних моделей генерації контенту на основі глибокого навчання відкриває нові перспективи для вирішення проблеми «контентного голоду». Ці методи мають великий потенціал не лише у великих студіях (AAA-розробка), але й для незалежних розробників, дозволяючи автоматизувати рутинні процеси створення текстур, моделей та оточення.

Хоча традиційні методи створення графіки (ручне моделювання, скульптингу, текстурування) залишаються стандартом якості, сучасний напрямок розвитку зосереджується на застосуванні генеративного штучного інтелекту для кардинального прискорення виробництва. Використання нейронних мереж, зокрема латентних дифузійних моделей, у цьому контексті дозволяє створювати високодеталізовані матеріали за лічені секунди, забезпечуючи при цьому необхідну для сучасних рушіїв технічну коректність.

Дана кваліфікаційна робота присвячена аналізу та дослідженню сучасних підходів до генерації контенту в ігровій індустрії, зокрема використанню дифузійних моделей для створення PBR-матеріалів. У результаті роботи виконаний детальний аналіз архітектур нейромереж, їх переваг та недоліків, а також розроблено новий гібридний метод з метою повної автоматизації процесу створення безшовних текстур та карт нормалей.

Ця робота спрямована на вирішення важливих проблем в області комп'ютерної графіки та дослідження перспективних напрямків інтеграції AI-інструментів у виробничі пайплайни, що має велике значення для економічної ефективності розробки ігор.

Таким чином, завдання розробки методу автоматизації етапів розробки комп’ютерних ігор за допомогою використанням інтелектуальних моделей генерації контенту є сучасним та актуальним.

Мета роботи - підвищення ефективності та скорочення часових витрат на етапах створення графічного контенту для комп’ютерних ігор шляхом розробки автоматизованого методу на основі латентних дифузійних моделей.

Об’єкт дослідження - процес генерації та обробки візуальних активів (асетів) у розробці комп’ютерних ігор.

Предмет дослідження - метод автоматизованої генерації безшовних текстур та карт нормалей з використанням технологій глибокого навчання.

Для досягнення мети вирішувалися наступні завдання:

1. Аналіз предметної області. Проведено дослідження сучасного стану ігрової індустрії, виявлено основні проблеми виробництва контенту (зростання бюджетів, часові витрати) та проаналізовано існуючі наукові підходи до генерації зображень (GAN, VAE, Diffusion Models).
2. Обґрунтування вибору методів. На основі порівняльного аналізу архітектур нейронних мереж обґрунтовано вибір латентних дифузійних моделей (Stable Diffusion) як базового інструменту дослідження та визначено вимоги до PBR-матеріалів.
3. Математичне моделювання. Розроблено математичну модель процесу дифузії та алгоритм гібридної генерації, що включає методи інженерії промптів, оцінки глибини сцени та обчислення градієнтів для створення карт нормалей.
4. Програмна реалізація. Створено програмний комплекс мовою Python з використанням бібліотек PyTorch та Diffusers. Реалізовано графічний інтерфейс користувача та оптимізовано роботу алгоритмів для апаратної платформи Apple Silicon (MPS).
5. Експериментальні дослідження. Проведено серію експериментів з генерації матеріалів різних типів (органіка, камінь, метал). Виконано оцінку якості

отриманих асетів та досліджено стабільність роботи системи під навантаженням.

6. Порівняльний аналіз. Здійснено порівняння ефективності розробленого автоматизованого методу з традиційним ручним підходом за критерієм часових витрат на створення одиниці контенту.

Вирішення цих задач сприяє розвитку методів автоматизації в GameDev та демонструє практичну цінність застосування генеративного штучного інтелекту для оптимізації виробничих процесів.

Апробація результатів магістерської роботи. Основні положення, теоретичні висновки та практичні результати дослідження доповідалися та обговорювалися на Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» (24 квітня 2025 р., м. Київ) та на II Всеукраїнській науково-технічній конференції «Виклики та рішення в програмній інженерії» (2025 р.).

1 АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ГЕНЕРАЦІЇ КОНТЕНТУ В ІГРОВІЙ ІНДУСТРІЇ

1.1 Аналіз сучасного стану індустрії розробки комп'ютерних ігор.

Ігрова індустрія пройшла шлях від нішевого технічного захоплення до домінантного сектору глобальної економіки розваг. Сьогодні це складна екосистема, де технологічні інновації тісно переплітаються з жорсткими фінансовими обмеженнями. Незважаючи на стабільне зростання ринку, розробники стикаються з парадоксальною ситуацією: створювати ігри стає складніше і дорожче, швидше, ніж зростають прибутки від них.

Згідно з аналітичним звітом Pixune Studios (2024), середня вартість розробки ігрових проєктів класу AAA за останній рік зросла на 20%, досягнувши позначки у 200 мільйонів доларів і більше [1]. Таке стрімке зростання бюджетів не можна пояснити лише інфляцією чи маркетинговими витратами. Основний драйвер зростання ціни - це експоненційне зростання вимог до якості та обсягу контенту. Структуру розподілу типового бюджету великого ігрового проєкту наведено на рисунку 1.1

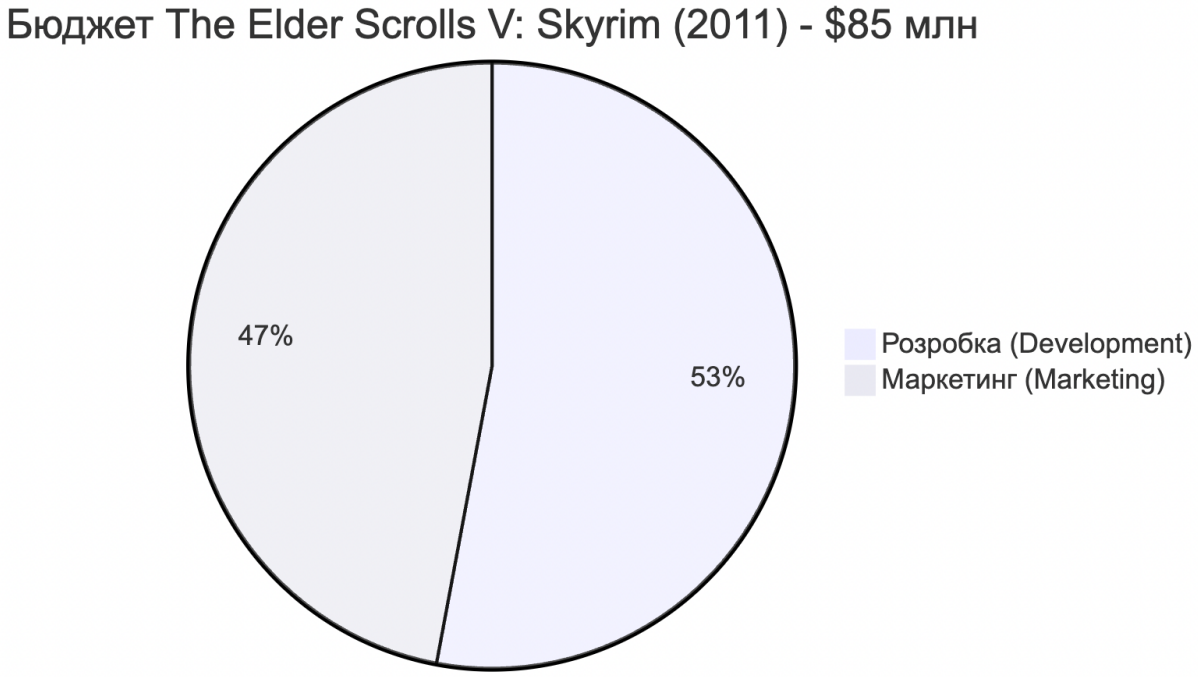


Рис. 1.1 Структура бюджету ААА-гри
на прикладі The Elder Scrolls V: Skyrim

Сучасний гравець очікує від ігор кінематографічної якості зображення та величезних відкритих світів, що ставить перед студіями колосальні виробничі виклики.

Цю тезу підтверджує фінансовий аналіз компанії EJAW (2023), який вказує на диспропорцію у розподілі бюджетів: левова частка коштів витрачається саме на візуальну складову (арт, моделі, текстури). Наприклад, витрати лише на аутсорсинг 3D-арту для кінематографічних ігор можуть перевищувати 500 000 доларів на один ігровий рівень [2]. Враховуючи високі погодинні ставки кваліфікованих художників, традиційний підхід «ручного виробництва» стає фінансово обтяжливим гальмом для індустрії, обмежуючи можливості експериментів та творчого ризику.

Фундаментальною причиною такої трудомісткості є не лише обсяг роботи, а й зміна технологічних стандартів. Як зазначає Джейсон Грегорі у *«Game Engine Architecture»*, сучасні ігрові рушії повністю перейшли на архітектуру відкладеного затінення та фізично коректного рендерингу (PBR) [3]. Цей перехід фактично

змінив парадигму створення графіки: ми більше не малюємо «колір» об'єкта, ми моделюємо його фізичні властивості.

Технічна складність цього процесу детально описана у роботі Метта Фарра *«Physically Based Rendering»*. Автори наголошують, що для коректної взаємодії світла з поверхнею недостатньо одного зображення. Кожен об'єкт вимагає створення та точної синхронізації мінімум чотирьох карт: “Albedo”, “Normal”, “Roughness” та “Metallic” [4]. Для художника це означає, що створення одного цифрового ресурсу або об'єкту (англ. assets, далі - ассети), перетворюється на комплексний інженерний процес, де помилка на одному етапі (наприклад, у карті нормалей) руйнує візуальне сприйняття всього об'єкта, тут такі етапи як запікання (baking) та ручне забезпечення безшовності (tiling) займають левову частку часу. Це створює ефект «виробничого вузького місця», коли команда програмістів змушена чекати на завершення роботи арт-відділу. Порівняння часових витрат на етапи створення ассету в традиційному виробничому конвеєрі (англ. pipeline, далі - пайплайн) візуалізовано на рисунку 1.2.

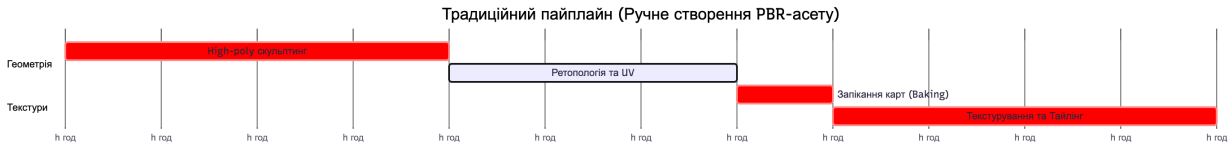


Рис. 1.2 Часові витрати традиційного пайплайну створення PBR-ассету

Проте, ринок диктує свої умови, і попит на контент продовжує зростати незалежно від складності його виробництва. Згідно з даними Global Market Insights (2024), ринок цифрових 3D-ассетів вже оцінюється у 24,9 мільярда доларів із прогнозованим щорічним зростанням на 12,7% [5]. Ця статистика свідчить про те, що індустрія потребує не просто більше ігор, а більше насичених, деталізованих віртуальних просторів, які фізично неможливо заповнити виключно ручною працею в розумні терміни.

Реакцією на цей дисбаланс став активний пошук інструментів автоматизації. У звіті Industry Research (2024) зазначається, що вже 61% ігрових студій інтегрували інструменти штучного інтелекту у свої виробничі процеси [6]. Це

свідчить про те, що галузь готова до зміни технологічного укладу, переходячи від ручного ремісництва до індустріальної генерації контенту.

Таким чином, ми спостерігаємо ситуацію, коли традиційні методи створення графічних ресурсів досягли своєї межі ефективності. Подальше нарощування штату художників стає економічно невигідним, а вимоги до якості продовжують зростати. Це формує об’єктивну потребу в детальному аналізі існуючих виробничих процесів, щоб виявити конкретні етапи, які піддаються автоматизації за допомогою новітніх інтелектуальних моделей.

Саме розгляд структури цих процесів (пайплайнів) та класифікація активів дозволить зрозуміти, де саме впровадження ШІ дасть найбільший економічний та якісний ефект.

1.2 Класифікація ігрових активів та традиційні пайплайни їх створення.

1.2.1 Класифікація активів: Фокус на оточенні (Environment Art)

У сучасному пайплайні розробки ігрових рушіїв під терміном «ассет» (asset) розуміють будь-який цифровий об'єкт (модель, текстура, звук, анімація), що завантажується у пам'ять та використовується грою. Згідно з класифікацією, наведеною Дж. Грегорі у *«Game Engine Architecture»*, графічні ассети поділяються на динамічні (персонажі, транспорт, фізичні об'єкти) та статичні (архітектура, ландшафт, рослинність) [3].

У рамках даної роботи основна увага зосереджена на категорії Environment Art (Мистецтво оточення). Це зумовлено тим, що саме статичне оточення формує візуальну атмосферу гри та займає найбільший обсяг пам'яті. В межах цієї категорії критично важливо розрізняти два типи активів, методи виробництва яких кардинально відрізняються: унікальні ассети та тайлові матеріали.

Унікальні ассети (Unique Assets / Props)

До цієї групи належать об'єкти, що мають специфічну, неповторну геометрію та унікальну текстурну розгортку (UV-mapping). Математично UV-розгортка є

відображенням координат тривимірної поверхні на двовимірну площину текстури, що детально описано в роботах з теорії рендерингу [4].

Приклади: Статуя героя, складний механізм, специфічні меблі.

Особливість виробництва: Текстури для таких об'єктів жорстко прив'язані до координат вершин моделі й не можуть бути використані на інших об'єктах без спотворень.

Тайлові матеріали (Tileable / Seamless Materials)

Це текстури, що не прив'язані до конкретної геометрії та призначені для покриття великих площ поверхні.

Приклади: Цегляна стіна, бетонна підлога, асфальт, трава.

Особливість виробництва: Головною технічною вимогою до таких матеріалів є безшовність (tileability) - здатність текстури стикуватися сама з собою по горизонталі та вертикалі без видимих швів чи артефактів, що є критичним для PBR-пайплайну [4].

Рис. 1.3 Візуальна відмінність між унікальним ассетом (знизу) та тайловими матеріалами (зверху)

Аналіз візуальної структури сучасних ААА-ігор демонструє диспропорцію між унікальним та повторюваним контентом. Хоча унікальні ассети («Hero props») часто слугують фокусом уваги гравця, кількісний аналіз ігрових сцен показує, що до 80% візуального простору екрану займають саме тайлові матеріали: стіни будівель, дорожнє покриття, ландшафт та небо. Якість виконання цих фонових елементів безпосередньо впливає на занурення гравця у віртуальний світ.

Ручне створення варіацій таких матеріалів являє собою значний виробничий виклик. Наприклад, для забезпечення візуального різноманіття однієї локації художнику може знадобитися створити десятки варіацій матеріалу «старого бетону» (з тріщинами, з мохом, мокрого, зруйнованого тощо). Цей процес характеризується високою монотонністю та значними часовими витратами. Більше того, механічна помилка у забезпеченні “безшовності” на цьому етапі призводить до появи повторюваних візуальних артефактів (ефект «сітки»), які руйнують реалістичне сприйняття сцени.

Саме тому розробка методу автоматизованої генерації безшовних тайлових PBR-текстур має найвищий пріоритет для індустрії. Це дозволяє вирішити проблему заповнення великих ігрових просторів якісним контентом з мінімальними витратами часу художника. Така технічна необхідність прямо корелює зі зростаючим попитом на цифрові активи, зафіксованим у звітах Global Market Insights [5], які вказують на необхідність масштабування виробництва контенту.

1.2.2. Стандартний пайплайн «High-to-Low Poly» (Геометричний етап)

Сучасним виробничим стандартом для створення високоякісних ігрових активів є пайплайн «High-to-Low Poly». Цей підхід виник як необхідний компроміс між художнім прагненням до необмеженої деталізації та жорсткими технічними обмеженнями апаратного забезпечення, яке не здатне обробляти моделі з мільйонами полігонів у реальному часі. Процес створення геометрії є ітеративним і поділяється на три критичні етапи.

1. Високополігональне моделювання (Digital Sculpting)

Першим етапом є створення еталонної моделі (High-Poly), яка слугує джерелом візуальної інформації. Художник використовує спеціалізоване програмне забезпечення для цифрового скульптингу (наприклад, ZBrush або Mudbox) для роботи з «цифровою глиною». Як зазначає Вільям Вон у праці *«Digital Modeling»*, головна мета цього етапу - максимально точно передати художній задум та форму, повністю ігноруючи технічні обмеження топології сітки [8].

На цьому етапі щільність полігональної сітки може досягати десятків мільйонів трикутників. Це дозволяє художнику моделювати найдрібніші деталі поверхні: пори шкіри, фактуру тканини, сколи на камені чи подряпини на металі.

Процес є суто художнім і вимагає глибокого розуміння анатомії (для персонажів) або структурних властивостей матеріалів (для оточення). Створення одного складного пропсу (унікальний ассет), наприклад, декоративної колони з барельєфом, може займати від 4 до 16 годин ручної праці, оскільки кожен елемент поверхні опрацьовується вручну.

Рис. 1.4 Процес цифрового скульптингу високополігональної моделі (ліворуч) та фінальна деталізація (праворуч)

2. Ретопологія (Retopology)

Отримана High-Poly модель є непридатною для використання в ігровому рушії через надмірну кількість полігонів, що призвело б до критичного падіння продуктивності. Наступним кроком є ретопологія - процес створення нової, оптимізованої низькополігональної оболонки (Low-Poly) поверх високополігонального оригіналу.

Згідно з Т. Акенін-Моллером, головною вимогою до Low-Poly моделі є не просто зменшення кількості трикутників, а правильна організація полігональних петель (edge loops) [7]. Це необхідно для забезпечення коректної деформації об'єкта при анімації та мінімізації артефактів затінення (shading artifacts). Це переважно ручний інженерний процес: художник повинен фактично побудувати нову модель, вершина за вершиною, «обгортаючи» скульптуру. Автоматичні інструменти ретопології (auto-retopology) існують, але для PBR-ассетів високої якості вони часто створюють спіралеподібну топологію, що ускладнює подальше розгортання, тому професіонали надають перевагу ручному методу.

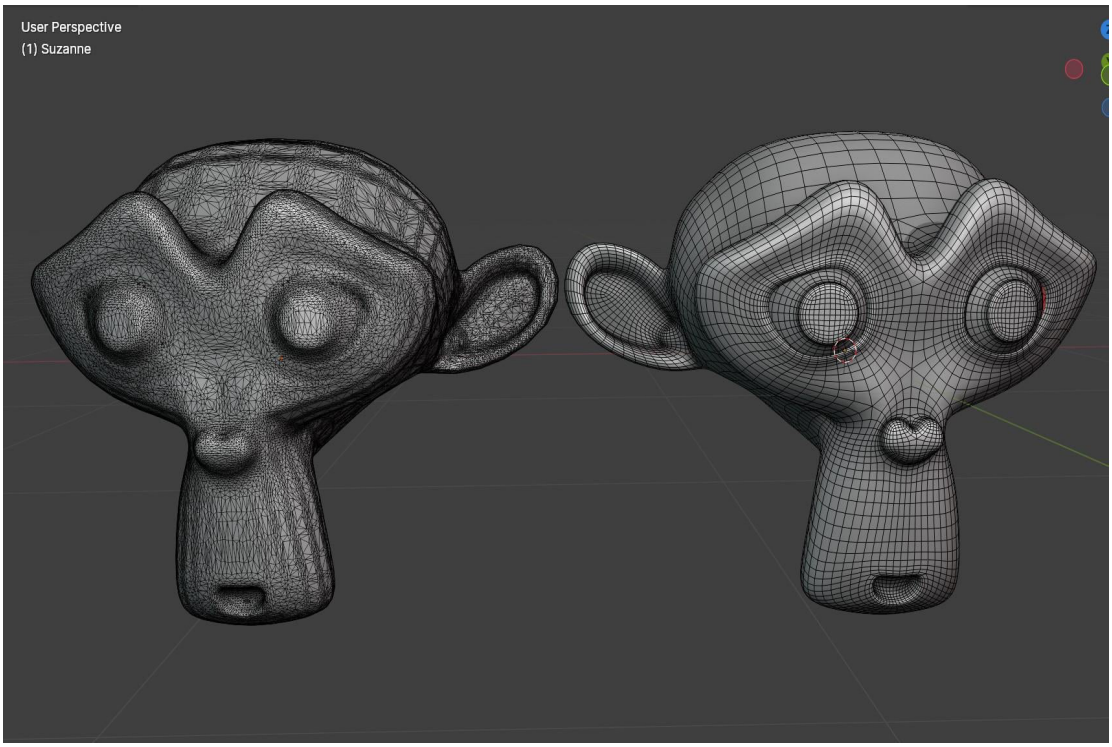


Рис. 1.5 Порівняння полігональної сітки: High-Poly скульлт (ліворуч) та оптимізована Low-Poly сітка після ретопології (праворуч)

3. UV-розгортка (UV Mapping)

Для того, щоб накласти двовимірну текстуру на складний тривимірний об'єкт, необхідно виконати процес параметризації поверхні, відомий як UV-розгортка. Це процес проєктування координат поверхні 3D-моделі (XYZ) на двовимірну площину текстури (UV). В. Вон влучно порівнює цей процес із розгортанням картонної коробки або зняттям шкірки з апельсина так, щоб розкласти її на столі максимально плоско [8].

Критичними параметрами якості на цьому етапі є:

- Texel Density (Щільність текселів): Забезпечення однакової кількості пікселів текстури на одиницю площі моделі. Різна щільність призводить до того, що одні частини об'єкта виглядають чіткими, а сусідні - розмитими.
- Відсутність потягувань (Distortion): Мінімізація геометричних спотворень, коли квадратна текстура на моделі виглядає розтягнутою або стиснутою.
- Оптимізація простору (Packing): Максимально щільне розміщення "острівців" розгортки на карті (UV space) для економії відеопам'яті.
- Ручне створення оптимальної UV-розгортки для об'єкта складної форми (наприклад, дерева або скелі) є технічно складною задачею, яка займає значну частину робочого часу 3D-художника (до 20% від загального часу) і є необхідною передумовою для наступного етапу - текстурування.

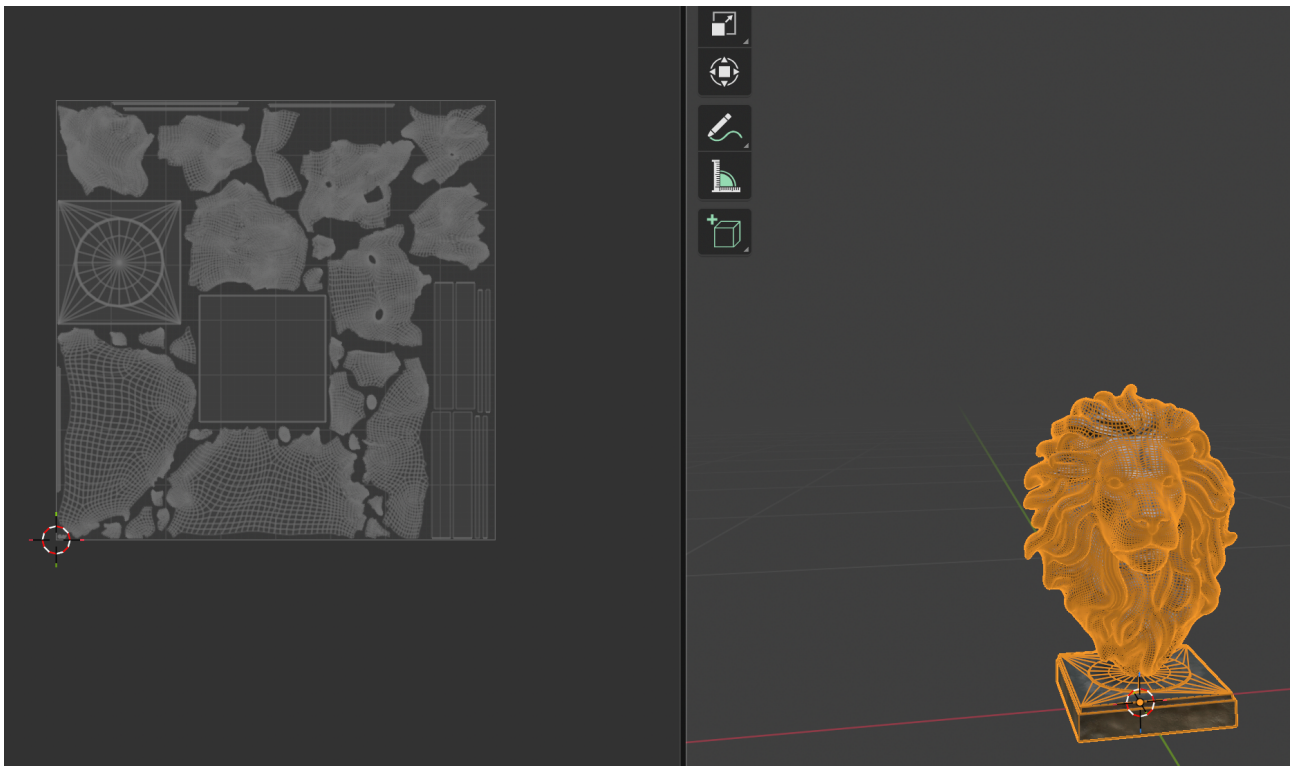


Рис. 1.6 Приклад UV-розгортки: проекція тривимірної геометрії на двовимірний простір текстур

1.2.3 Процес текстурування та «запікання» (Baking)

Після завершення роботи над геометрією та UV-координатами розпочинається етап перенесення деталізації з високополіагональної моделі на ігрову. Цей процес називається «запіканням» (Baking).

Згідно з принципами, описаними в *«Real-Time Rendering»*, мікрорельєф поверхні не моделюється геометрично в рушії, оскільки це призвело б до неприпустимого навантаження на GPU. Замість цього, деталізація імітується за допомогою спеціальних текстур, які «обманюють» розрахунки освітлення [7].

Ключовою операцією є запікання карти нормалей (Normal Map Baking). Технічно цей процес виглядає так:

- Програмне забезпечення (Marmoset Toolbag, Substance Painter) проєктує промені (ray casting) від поверхні Low-Poly моделі до поверхні High-Poly моделі.
- Обчислюється різниця кутів між нормаллю (вектором перпендикуляра) низькополігонального трикутника та деталізованою нормаллю скульпту.

- Ця різниця кодується у вигляді кольору (R, G, B) і записується у пікселі текстури.

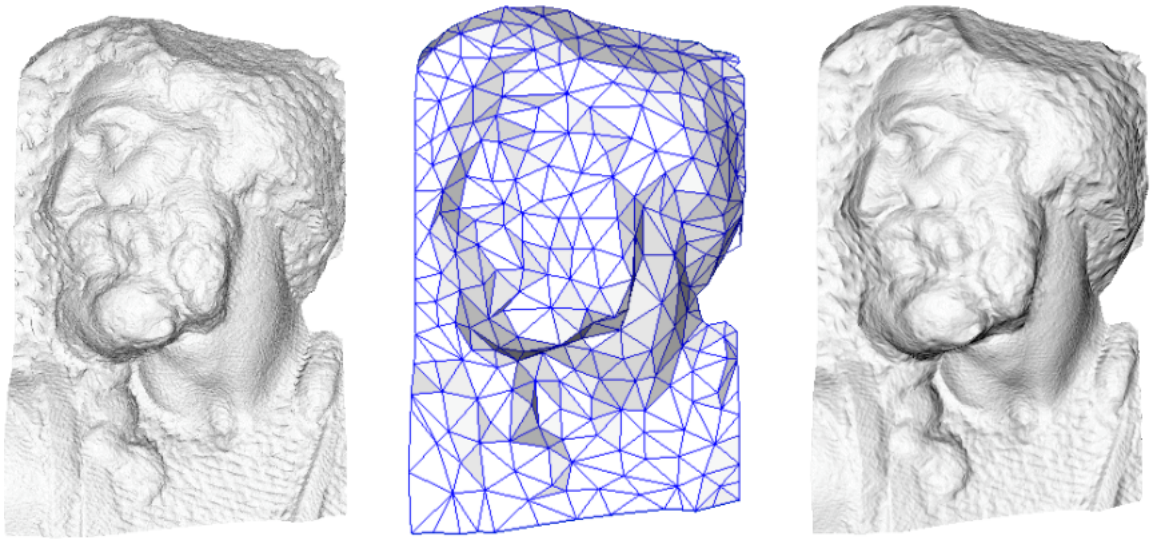


Рис. 1.7 Процес перенесення деталізації (Baking): High-Poly модель (ліворуч), оптимізована Low-Poly сітка (по центру) та результат застосування Normal Map (праворуч)

Проблематика етапу: Цей процес є технічно складним і чутливим до помилок. Неправильне налаштування "клітки" (cage) при запіканні призводить до артефактів. Крім того, процес є деструктивним: будь-яка, навіть мінімальна зміна геометрії або UV-розгортки, вимагає повного повторного циклу запікання всіх карт, що робить ітерації дуже дорогими у часі.

Після запікання розпочинається етап текстурування - створення карт кольору (Albedo), шорсткості (Roughness) та металічності (Metallic). Хоча сучасні інструменти підтримують процедурні маски, створення реалістичної унікальної поверхні (наприклад, іржі, що стікає саме у щілинах) все ще вимагає значної частки ручного малювання.

1.2.4 Проблема безшовності (Tiling) у ручному пайплайні

Окремим і найбільш трудомістким викликом при створенні матеріалів для оточення (стіни, ландшафт, дороги) є забезпечення їх безшовності (Tileability).

Текстура вважається безшовною, якщо при її багаторазовому повторенні по вертикалі та горизонталі місця стиків залишаються невидимими.

У традиційному пайплайні (наприклад, при обробці фотосканованих поверхонь у Photoshop) цей процес виконується вручну:

1. Зміщення (Offset): Зображення зсувається на 50% по осях X та Y, внаслідок чого краї зображення опиняються в центрі, утворюючи видимий хрестоподібний шов (рисунок 1.8).
2. Ручна ретуш: Художник використовує інструменти клонування (Clone Stamp) або відновлення (Healing Brush), щоб вручну "замалювати" шов, намагаючись синтезувати перехід, який збереже логіку візерунка та частотність деталей.
3. Синхронізація карт: Це найскладніший аспект. Оскільки PBR-матеріал складається з 4-5 карт, художник повинен внести ідентичні зміни на кожній з них. Якщо прибрати камінець на карті кольору (Albedo), але забути прибрати його рельєф на карті нормалей (Normal), у грі це виглядатиме як "привид" камінця, що руйнує реалізм.

Ручне забезпечення безшовності для складних органічних текстур може займати до 30-40% загального часу роботи над ассетом. Саме монотонність цього процесу та складність синхронізації карт роблять його ідеальним кандидатом для автоматизації методами глибокого навчання.

Рис. 1.8 Візуалізація проблеми тайлінгу: поява видимих швів при багаторазовому повторенні текстури без корекції країв

1.3 Огляд методів процедурної та генеративної генерації контенту

Проблема автоматизації створення контенту не є новою для ігрової індустрії. Історично підходи до вирішення цього завдання еволюціонували від простих алгоритмічних методів до сучасних систем на базі штучного інтелекту. Для обґрунтування вибору методу дослідження необхідно провести порівняльний аналіз існуючих підходів.

На ранніх етапах розвитку індустрії основним інструментом була процедурна генерація контенту (Procedural Content Generation, PCG). Цей підхід базується на використанні детермінованих алгоритмів та псевдовипадкових чисел для створення даних.

Найпоширенішим методом є використання функцій шуму, таких як шум Перліна (Perlin Noise) або шум Вороного. Як зазначають Тогеліус та Яннакакіс у своїй праці, ці алгоритми дозволяють генерувати природні, органічні текстури (хмари, ландшафт, мармур) з мінімальними витратами пам'яті [9].

Переваги: Висока швидкість виконання, нескінченна варіативність, повна математична передбачуваність.

Недоліки: Результат часто виглядає «штучним» та одноманітним. Алгоритмічні методи важко налаштувати для створення складних, стилізованих зображень (наприклад, «стімпанк-метал» або «кіберпанк-неон»), оскільки вони не «розуміють» семантики об'єкта, а лише маніпулюють пікселями за формулою.

Обмеженість класичних алгоритмів призвела до переходу до генеративного глибокого навчання (Deep Generative Learning). На відміну від PCG, ці моделі не програмуються вручну, а навчаються на великих наборах даних (dataset), вивчаючи статистичний розподіл ознак реальних зображень. Це дозволяє їм генерувати новий контент, який візуально схожий на навчальні приклади. Серед архітектур нейронних мереж ключову роль відіграють три підходи.

Генеративно-змагальні мережі (GAN) ззапропоновані Яном Гудфеллоу у 2014 році, GAN (Generative Adversarial Networks) тривалий час були стандартом індустрії. Архітектура складається з двох мереж: Генератора, що створює зображення, та Дискримінатора, що намагається відрізнити підробку від оригіналу.

Застосування в GameDev: Використовуються для покращення роздільної здатності текстур (Super-Resolution) та перенесення стилю.

Недоліки для нашої задачі: Головною проблемою GAN є «колапс мод» (mode collapse), коли модель починає генерувати однакові зображення, ігноруючи варіативність. Крім того, GAN складно контролювати текстовими командами, що робить їх незручними для художників, яким потрібен точний результат за описом [10].

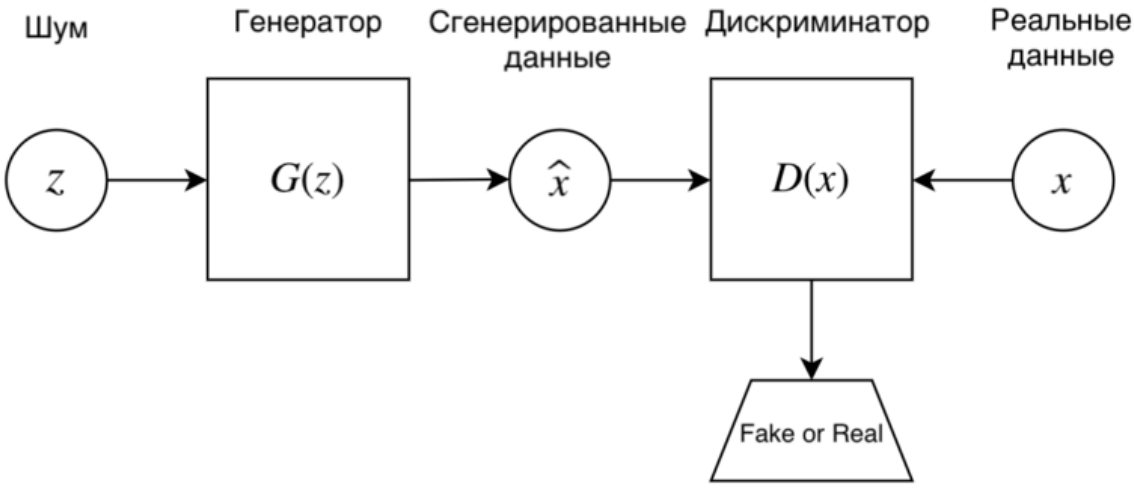


Рис 1.9 Принципова схема навчання генеративно-змагальної мережі (GAN)

Варіаційні автокодувальники (VAE), архітектура VAE (Variational Autoencoders) працює шляхом стиснення вхідного зображення у компактний латентний код (вектор) і його подальшого відновлення. Це дозволяє плавно інтерполювати між різними об'єктами (наприклад, плавно перетворити текстуру трави на текстуру піску).

Недоліки: Головною проблемою VAE є схильність до генерації розмитих (blurry) зображень через особливості функції втрат. Для задач PBR-текстурування, де критично важлива чіткість деталей (мікрорельєф), цей недолік є блокуючим.

Дифузійні моделі (Diffusion Models) сучасним "state-of-the-art" рішенням є ймовірнісні дифузійні моделі (Denoising Diffusion Probabilistic Models, DDPM).

Принцип їх роботи полягає у поступовому додаванні шуму до зображення до його повного руйнування (прямий процес) та навчання нейромережі відновлювати зображення з шуму (зворотний процес).

Особливу цінність для індустрії становлять латентні дифузійні моделі (Latent Diffusion Models, LDM), такі як Stable Diffusion. Вони виконують процес дифузії не у піксельному просторі (що потребує величезних ресурсів), а у стиснутому латентному просторі, що робить їх доступними для запуску на споживчих відеокартах [14].

Висока керованість: Механізм Cross-Attention дозволяє точно керувати генерацією через текст (Prompts).

Стабільність: На відміну від GAN, дифузійні моделі мають стабільний процес навчання.

Якість: Вони забезпечують найвищу деталізацію та різноманітність текстур, що є критичним для PBR-матеріалів.

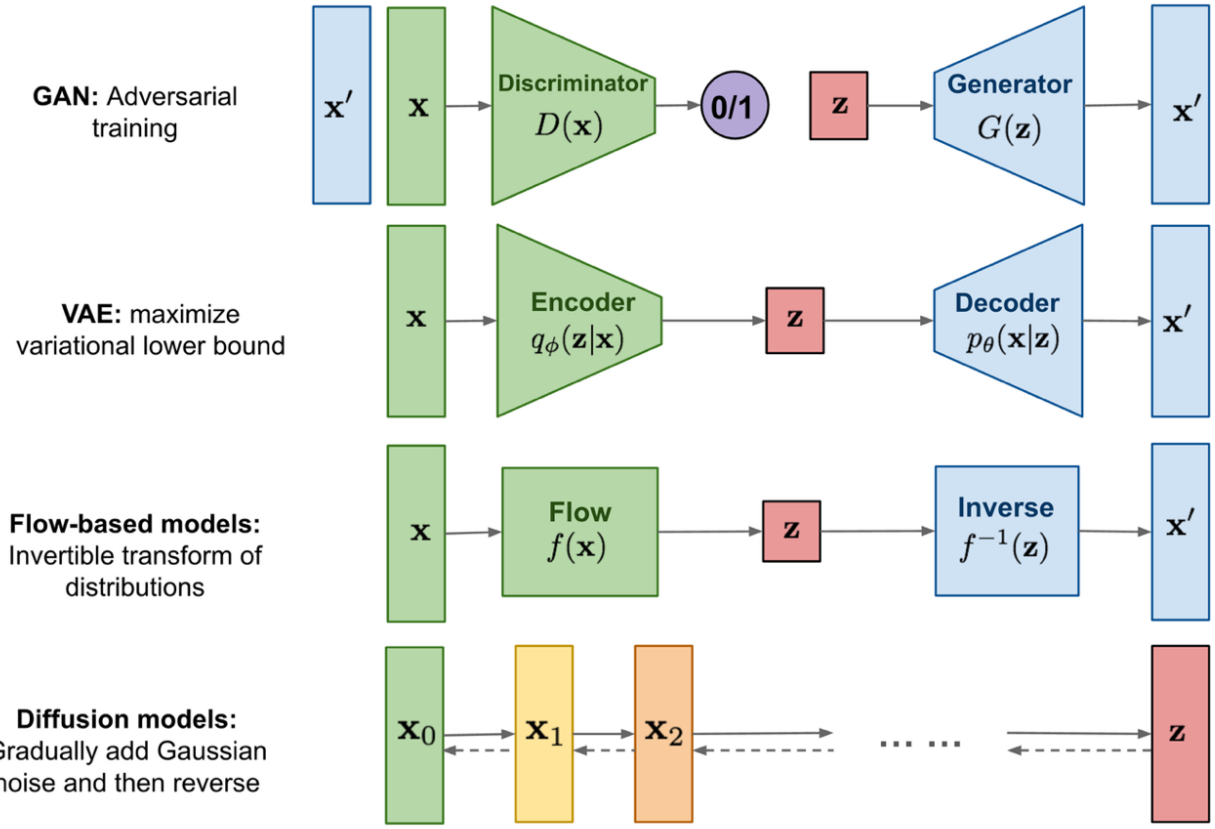


Рис. 1.10 Порівняння архітектур генеративних моделей (GAN, VAE, Diffusion)

Проведений аналіз показує, що класичні методи PCG вичерпали свій потенціал для створення фотореалістичного контенту. Серед методів глибокого навчання саме латентні дифузійні моделі демонструють найкращий баланс між якістю зображення, варіативністю та можливістю керування. Тому саме ця архітектура обрана як основа для розробки автоматизованого методу генерації PBR-текстур у даній роботі.

1.4 Проблематика створення PBR-матеріалів за допомогою нейромереж

Незважаючи на вражаючі успіхи генеративного штучного інтелекту в створенні художніх зображень, пряме використання результатів генерації у професійному ігровому пайплайні наштовхується на низку суттєвих технічних перешкод. Головна проблема полягає у невідповідності семантики вихідних даних нейромережі вимогам фізично коректного рендерингу.

1.4.1 Обмеження базових дифузійних моделей (RGB vs PBR)

Сучасні дифузійні моделі такі як Stable Diffusion, Midjourney навчаються на величезних масивах даних (наприклад, LAION-5B), які складаються переважно з фотографій реального світу та творів цифрового мистецтва. Як наслідок, нейромережа генерує фінальне зображення (Final Render), яке вже містить інформацію про освітлення сцени.

Згенероване зображення неминуче включає:

- Затінення (Shadows): Власні та падаючі тіні від джерел світла, які "існували" в уяві моделі при генерації.
- Бліки (Specular Highlights): Яскраві плями на глянцевих поверхнях.
- Глобальне освітлення (Ambient Occlusion): Затемнення у кутах та щілинах.

Для PBR-пайплайну, описаного М. Фарром [4], така інформація на карті кольору (Albedo) є критичним дефектом. Карта Albedo повинна містити лише інформацію про власний колір поверхні (diffuse color) без жодного впливу світла.

Суть проблеми: Якщо використати згенероване нейромережею зображення "як є" в ігровому рушії, виникне візуальний конфлікт. "Намальовані" тіні залишатимуться нерухомими навіть тоді, коли віртуальне джерело світла у грі змінить позицію. Це руйнує ілюзію об'єму і робить об'єкт пласким. Тому першим викликом автоматизації є задача «знебарвлення» освітлення (Delighting) - алгоритмічного видалення тіней та бліків зі згенерованого зображення для отримання "чистого" Albedo.

1.4.2 Проблема «One-to-Many» (Відновлення карт властивостей)

Наступним критичним викликом є генерація допоміжних карт PBR (Normal, Roughness, Displacement) на основі одного згенерованого зображення (Albedo). З математичної точки зору, це задача, що належить до класу некоректно поставлених (ill-posed problems), або задач типу «один-до-багатьох» (one-to-many mapping).

Суть проблеми полягає у втраті інформації при проекції тривимірного світу на двовимірну площину зображення. Коли нейромережа (або камера) фіксує піксель, інформація про те, чому цей піксель має саме такий колір, втрачається.

Приклад: Темна пляма на текстурі стіни може бути наслідком трьох різних фізичних явищ:

- геометричне заглиблення (тріщина), куди не потрапляє світло.
- зміна кольору матеріалу (пляма бруду або фарби) на абсолютно рівній поверхні.
- тінь від іншого об'єкта.

Для людини контекст часто дозволяє розрізнити ці випадки, але для алгоритмів комп'ютерного зору це джерело невизначеності.

Традиційні методи (фільтри у Photoshop або ShaderMap), які намагаються конвертувати яскравість пікселя у висоту (Dark is Deep, Bright is High), часто дають помилкові результати. Вони інтерпретують будь-яку темну пляму (наприклад, чорний напис на білій стіні) як глибоку яму, що є геометрично некоректним.

Сучасні дифузійні моделі "бачать" лише колір і не мають вбудованого розуміння 3D-геометрії об'єкта. Тому пряма генерація карт нормалей за допомогою стандартного Stable Diffusion часто призводить до семантичних галюцинацій, коли рельєф не відповідає зображенню (наприклад, гладка текстура генерує бугристу карту нормалей). Це вимагає розробки спеціалізованого модуля (наприклад, на базі оцінки глибини, Depth Estimation), який буде інтегрований у пайплайн генерації.

1.4.3 Технічні виклики забезпечення безшовності (Tiling) у латентному просторі

Третім, і не менш критичним обмеженням базових дифузійних моделей, є неможливість генерації математично точних безшовних (tileable) зображень "з коробки". Ця проблема лежить на перетині архітектури згорткових нейронних мереж (CNN) та особливостей навчальних наборів даних.

Основою архітектури Stable Diffusion є мережа U-Net, яка використовує операції згортки (convolution) для обробки інформації. У стандартній реалізації ці згортки використовують Zero-padding (доповнення нулями) або Reflection-padding на краях зображення. Це означає, що коли нейромережа "дивиться" на лівий край зображення, вона "бачить" за його межами порожнечу або дзеркальне відображення, але ніяк не правий край того самого зображення.

Внаслідок цього, нейронна мережа генерує зображення як окрему, замкнуту композицію (наприклад, фотографію стіни), а не як безперервний патерн. Пікселі на лівій границі ($x = 0$) та правій границі ($x = W$) генеруються незалежно один від одного, без урахування умови безперервності.

Спроби вирішити цю проблему виключно методами Prompt Engineering (додаванням слів «seamless», «tileable», «texture» у запит) дають лише частковий результат. Модель, навчена на прикладах текстур, може спробувати створити візуально однорідну структуру, але вона не здатна забезпечити попіксельну точність на стиках.

Результат: При спробі замостити таким зображенням велику площу у грі (Tiling), виникають чітко помітні шви у вигляді ліній розриву візерунка або різких перепадів яскравості. Це явище називається «grid artifact» (ефект сітки), який є неприпустимим для ААА-проектів.

Традиційні методи вирішення цієї проблеми (Offset + Clone Stamp у Photoshop) працюють у піксельному просторі. Однак, оскільки Stable Diffusion є латентною моделлю (LDM), генерація відбувається у стиснутому просторі ознак (latent space), який має меншу розмірність (зазвичай 64x64 для зображення

512x512). Спроба просто "зсунути" латентний код не гарантує безшовності після декодування, оскільки VAE-декодер також може вносити артефакти на краях.

1.5 Постановка задачі дослідження

Отже, для створення дійсно безшовних ассетів необхідна модифікація самого процесу генерації, зокрема впровадження механізму кільцевих згорток (Circular Convolution), який змусить нейромережу сприймати зображення як тор (топологічно замкнуту поверхню), де лівий край є продовженням правого, а верхній нижнього.

Проведений аналіз дозволяє зробити наступні висновки:

Ігрова індустрія перебуває в стані «контентної кризи»: вимоги до якості та обсягів графіки (PBR, Open World) зростають швидше, ніж можливості ручного виробництва, що призводить до критичного збільшення бюджетів.

Традиційний пайплайн створення ассетів має чітко виражені «вузькі місця» (High-Poly скульптинг, запікання карт нормалей, ручний тайлінг), які погано піддаються масштабуванню.

Існуючі методи генеративного ШІ (зокрема, базові версії Stable Diffusion) хоч і забезпечують високу якість зображення, не вирішують специфічних інженерних задач GameDev: вони генерують зображення із «запеченим» світлом, не створюють карти нормалей та не гарантують математичної безшовності.

На основі цього формується мета та завдання даної роботи:

Необхідно розробити гібридний автоматизований метод, який поєднає творчий потенціал латентних дифузійних моделей з інженерними алгоритмами комп'ютерного зору.

Такий метод повинен забезпечувати:

1. Генерацію базової текстури за текстовим описом.
2. Автоматичне забезпечення безшовності (Tiling) на рівні генерації.
3. Синтез коректних PBR-карт (зокрема Normal Map) з урахуванням фотометричних особливостей.

2 МЕТОДИ ТА МАТЕМАТИЧНІ МОДЕЛІ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ІГРОВИХ АССЕТІВ

2.1 Математична модель дифузійного процесу

В основі роботи системи автоматизованої генерації текстур лежить клас імовірнісних генеративних моделей, відомих як дифузійні ймовірнісні моделі (Denoising Diffusion Probabilistic Models - DDPM). Математичний апарат цих моделей базується на принципах нерівноважної термодинаміки та стохастичних диференціальних рівняннях.

Ключова ідея методу полягає у розбитті процесу генерації на два етапи:

1. Прямий процес дифузії (Forward Diffusion Process): Поступове руйнування структури даних шляхом додавання гауссівського шуму доти, доки зображення не перетвориться на чистий хаос.
2. Зворотний процес дифузії (Reverse Diffusion Process): Навчання нейронної мережі відновлювати структуру даних, покроково видаляючи шум.

Для задач генерації високоякісних ігрових ассетів, які вимагають високої роздільної здатності, класичний піксельний підхід є обчислювально неефективним. Тому в даній роботі використовується модифікація методу - латентна дифузійна модель (Latent Diffusion Model - LDM), запропонована Р. Ромбахом [14]. Основна відмінність полягає в тому, що процес дифузії відбувається не в просторі пікселів, а в стиснутому латентному просторі ознак.

2.1.1 Моделювання прямого процесу дифузії

Прямий процес дифузії визначається як марковський ланцюг (Markov Chain) фіксованої довжини T, який поступово додає гауссівський шум до початкового розподілу даних $x_0 \sim q(x_0)$ впродовж T кроків. Кожен перехід у ланцюзі залежить лише від попереднього стану.

Математично перехід від стану x_{t-1} до стану x_t описується умовним розподілом ймовірностей (2.1):

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right), \quad (2.1)$$

де x_t - латентний вектор зображення на кроці t; x_{t-1} - латентний вектор зображення на попередньому кроці;

β_t - дисперсія шуму на кроці t, яка визначається заздалегідь обраним графіком (variance schedule);

$\mathcal{N}$ - нормальний (гауссівський) розподіл;

I - одинична матриця, що визначає незалежність шуму по всіх розмірностях [11].

Параметр β_t зазвичай змінюється у діапазоні $(0,1)$. Для забезпечення стабільності навчання використовується лінійний або косинусний графік зміни β_t.

Оскільки процес є марковським, ми можемо виразити стан x_t безпосередньо через початковий стан x_0, не обчислюючи всі проміжні кроки. Для цього введемо допоміжні позначення: $\alpha_t = 1 - \beta_t$ та кумулятивний добуток $\overline{\alpha_t} = \prod_{i=1}^{t} \alpha_i$.

Тоді умовний розподіл $q(x_t|x_0)$набуває вигляду (2.2):

$$q(x_t|x_0) = \mathcal{N}\left(x_t; \sqrt{\overline{\alpha_t}}x_0, (1-\overline{\alpha_t})I\right), \quad (2.2)$$

де α_t - коефіцієнт збереження сигналу на кроці t;

$\overline{\alpha_t}$ - кумулятивний коефіцієнт збереження сигналу від початку процесу до кроку t [11].

Використовуючи властивість репараметризації (reparameterization trick), можна записати рівняння для отримання зашумленого зразка x_t у явному вигляді, придатному для програмної реалізації (2.3):

$$x_t = \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}\epsilon, \quad (2.3)$$

де x_0 - початковий (чистий) вектор даних;

ϵ - вектор стандартного нормального шуму, $\epsilon \sim \mathcal{N}(0, I)$.

Рівняння (2.3) є критично важливим для оптимізації процесу навчання, оскільки дозволяє миттєво отримати зашумлене зображення для будь-якого довільного кроку t, що значно прискорює формування навчальної вибірки. При $t \to T$ значення $\overline{\alpha_t} \to 0$, отже розподіл x_Tнаближається до стандартного нормального розподілу $\mathcal{N}(0, I)$, що означає повну втрату початкової інформації.

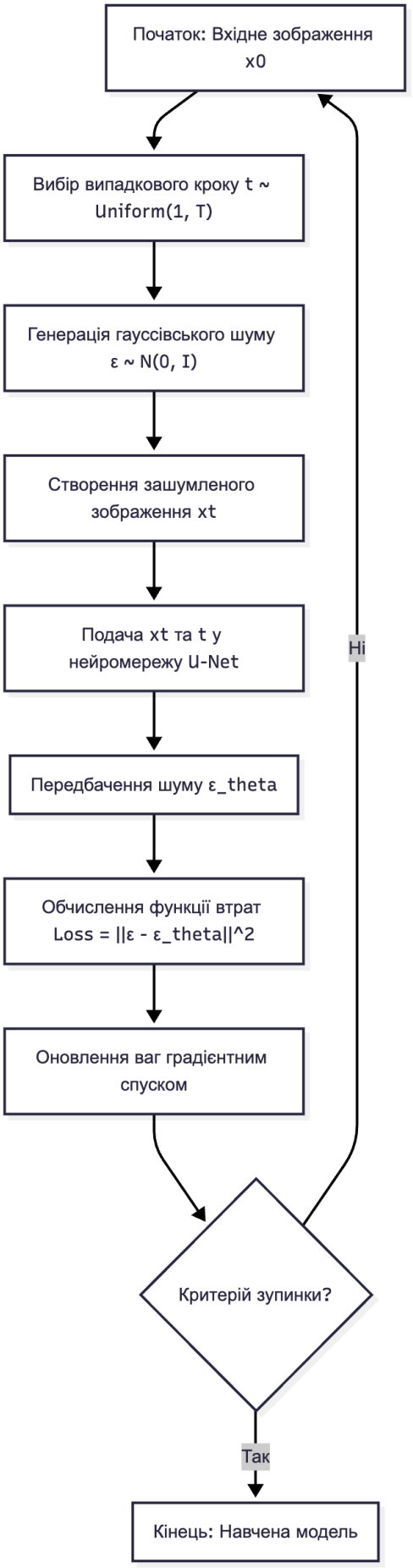


Рис. 2.1 Блок-схема алгоритму навчання імовірнісної дифузійної моделі

2.1.2 Моделювання зворотного процесу та функція втрат

Якщо прямий процес дифузії (2.1) дозволяє легко отримати зашумлені дані, то зворотний процес - відновлення $x_{t-1}зx_t$ - є нетривіальною задачею. Точний апостеріорний розподіл $q(x_{t-1}|x_t)$ є математично нерозв'язним (intractable), оскільки вимагає знання розподілу всіх можливих зображень у світі $q(x_0)$.

Для вирішення цієї проблеми використовується апроксимація зворотного розподілу за допомогою нейронної мережі з параметрами θ. Цей процес також моделюється як марковський ланцюг з гауссівськими переходами (2.4):

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\big(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\big), \tag{2.4}$$

де $\mu_\theta(x_t, t)$ - середнє значення розподілу, яке прогнозується нейромережею;

$\Sigma_\theta(x_t, t)$ - дисперсія, яка в класичній реалізації DDPM фіксується як $\beta_t I$ або навчається окремо;

t - часовий крок, який подається в мережу як вхідний параметр (через Time Embedding) [11].

Параметризація та функція втрат (Loss Function)

Головна задача навчання моделі полягає у тому, щоб налаштувати ваги нейромережі θ так, щоб розподіл p_0 максимально наближався до реального процесу дифузії. Хоча теоретично можна навчати мережу прогнозувати безпосередньо зображення x_{t-1}, дослідження Хо та ін. показали, що значно ефективнішим є підхід, при якому мережа прогнозує доданий шум ϵ.

Використовуючи рівняння (2.3) для вираження μ_0, можна спростити варіаційну нижню межу (Variational Lower Bound - ELBO) до простої функції втрат, що базується на середньоквадратичній помилці (MSE).

Цільова функція втрат (Objective Function), яка мінімізується під час навчання, має вигляд (2.5):

$$L_{simple}(\theta) = E_{x_0,\epsilon,t}\big[|\epsilon - \epsilon_\theta\big(\sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}\epsilon, t\big)|^2\big], \tag{2.5}$$

де x_0 - реальне зображення з навчальної вибірки;

ϵ - істинний гауссівський шум, доданий до зображення на кроці t, $\epsilon \sim \mathcal{N}(0, I)$;

$\epsilon_\theta(\dots)$ - нейронна мережа (U-Net), яка намагається передбачити цей шум, дивлячись на зашумлене зображення x_t та знаючи крок t;

$|\dots|^2$ - квадрат евклідової норми (MSE).

Фізичний зміст формули (2.5):

Нейромережа отримує на вхід зашумлене зображення і повинна відповісти на запитання: "Який саме шум був доданий до цього зображення?". Якщо мережа правильно вгадує шум ϵ_0, ми можемо математично відняти його від x_t і отримати трохи чистіше зображення x_{t-1}. Повторюючи цей процес ітеративно від T до 0, ми генеруємо нове зображення з чистого шуму.

Саме ця здатність ітеративного видалення шуму (Denoising) лежить в основі генеративного пайплайну, розробленого в даній роботі (див. рис. 2.2).

Процедура генерації зображення (семплювання) складається з наступних кроків:

1. Ініціалізується початковий тензор x_T з нормального розподілу $\mathcal{N}(0, I)$.
2. Запускається ітераційний цикл від T до 1.
3. На кожному кроці, якщо $t > 1$, генерується стохастичний шум z, інакше $z = 0$.
4. Нейромережа приймає поточний стан x_t і прогнозує доданий шум ϵ .
5. Виконується перерахунок стану x_{t-1} за формулою (2.3) з урахуванням коефіцієнтів α_t.
6. Отриманий латентний вектор x_0 декодується за допомогою VAE у простір пікселів.

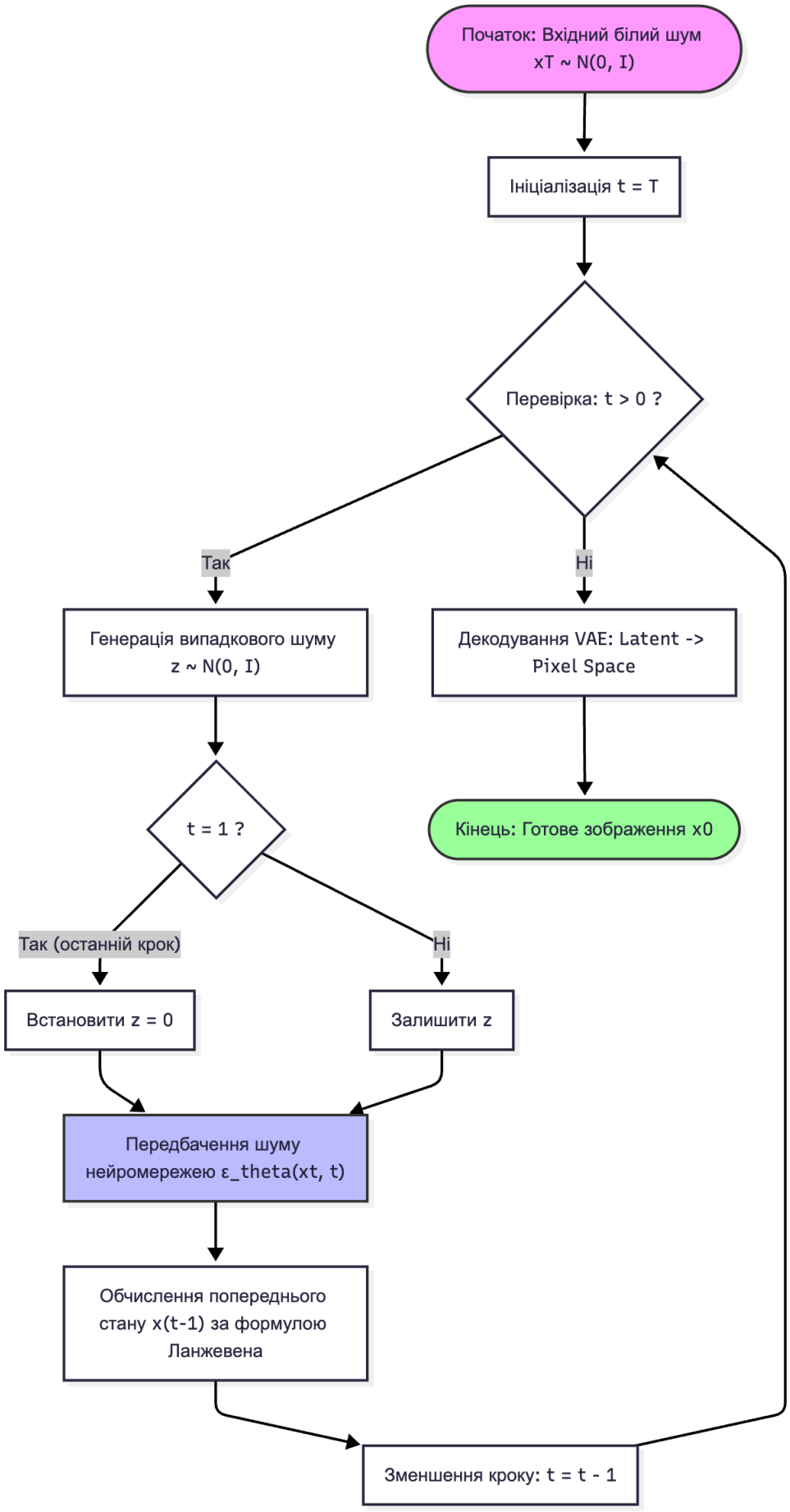


Рис. 2.2 Алгоритм зворотного семплювання (генерації) зображення

2.2 Розробка методу керованої генерації текстур на основі текстових описів

На відміну від безумовної генерації (unconditional generation), де модель створює випадкові зображення з шуму, задача автоматизації GameDev-пайплайну вимагає суворого контролю над семантикою результату. Для цього необхідно розробити метод, який дозволяє керувати зворотним процесом дифузії за допомогою природної мови (Text-to-Image Synthesis). В основі пропонованого методу лежить механізм умовного керування (Conditioning), реалізований через архітектуру трансформерів та механізм перехресної уваги (Cross-Attention).

2.2.1 Мультимодальне вбудовування тексту (CLIP Embedding)

Першим етапом керованої генерації є перетворення текстового опису (промпту) у числовий вектор, який "розуміє" нейромережа. Для цього в роботі використовується модель CLIP (Contrastive Language-Image Pre-training), розроблена OpenAI [12].

Модель CLIP складається з двох енкодерів: текстового $\mathcal{E}_{text}$ та візуального $\mathcal{E}_{image}$. Вона навчається на парах "зображення-текст" таким чином, щоб максимізувати косинусну схожість між вектором правильного тексту та вектором відповідного зображення у спільному багатовимірному просторі.

Процес обробки вхідного запиту P (prompt) складається з наступних кроків:

Токенізація: Текст розбивається на послідовність токенів:

$$T = [t_1, t_2, \dots, t_L],$$

де L - максимальна довжина послідовності (зазвичай 77 токенів).

Векторизація: Кожен токен перетворюється на векторне представлення за допомогою навченого текстового енкодера CLIP (2.6):

$$c = \mathcal{E}_{text}(P) \in R^{L \times d_{model}}, \quad (2.6)$$

де c - контекстний вектор (embedding), який містить семантичну інформацію про бажаний об'єкт (наприклад, "brick wall"), d_{model} - розмірність простору ознак (наприклад, 768 або 1024).

Саме цей вектор c подається на вхід дифузійній моделі U-Net для керування генерацією.

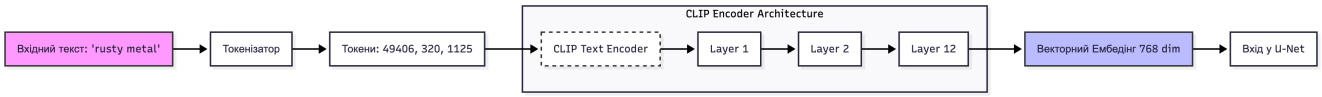


Рис. 2.3 Схема перетворення текстового запиту у векторний простір за допомогою CLIP-енкодера

2.2.2 Механізм перехресної уваги (Cross-Attention Mechanism)

Ключовою проблемою є те, як саме "впорснути" текстову інформацію c у процес генерації зображення. Архітектура U-Net, що використовується в дифузійних моделях, працює з просторовими ознаками зображення. Для інтеграції тексту використовується механізм перехресної уваги (Cross-Attention), запозичений з архітектури Transformer [13].

Механізм уваги дозволяє кожному пікселю проміжного зображення "звертатися" до відповідних слів у текстовому описі. Математично це реалізується через відображення трьох векторів:

- Q (Query): Запит, що формується з поточних ознак зображення $\phi(x_t)$.
- K (Key): Ключ, що формується з текстового вектора c.
- V (Value): Значення, що також формується з текстового вектора c.

Формула розрахунку уваги має вигляд (2.7):

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d}}\right)V, \quad (2.7)$$

де d - розмірність векторів, що використовується для нормалізації.

Для адаптації цього механізму до U-Net, шари перехресної уваги вбудовуються між згортковими блоками. Рівняння перетворення просторових ознак $\phi(x_t)$під впливом тексту c записується як (2.8):

$$\hat{\phi}(x_t) = Attention\big(W_Q \cdot \phi(x_t), W_K \cdot c, W_V \cdot c\big), \quad (2.8)$$

де W_Q, W_K, W_V - навчальні матриці ваг проекції.

Фізичний зміст:

Цей механізм дозволяє моделі динамічно фокусуватися на різних частинах тексту в залежності від того, яку частину зображення вона зараз генерує. Наприклад, при генерації пікселів стіни модель "звертає увагу" на слово "цегла", а при генерації поверхні - на слово "мох".

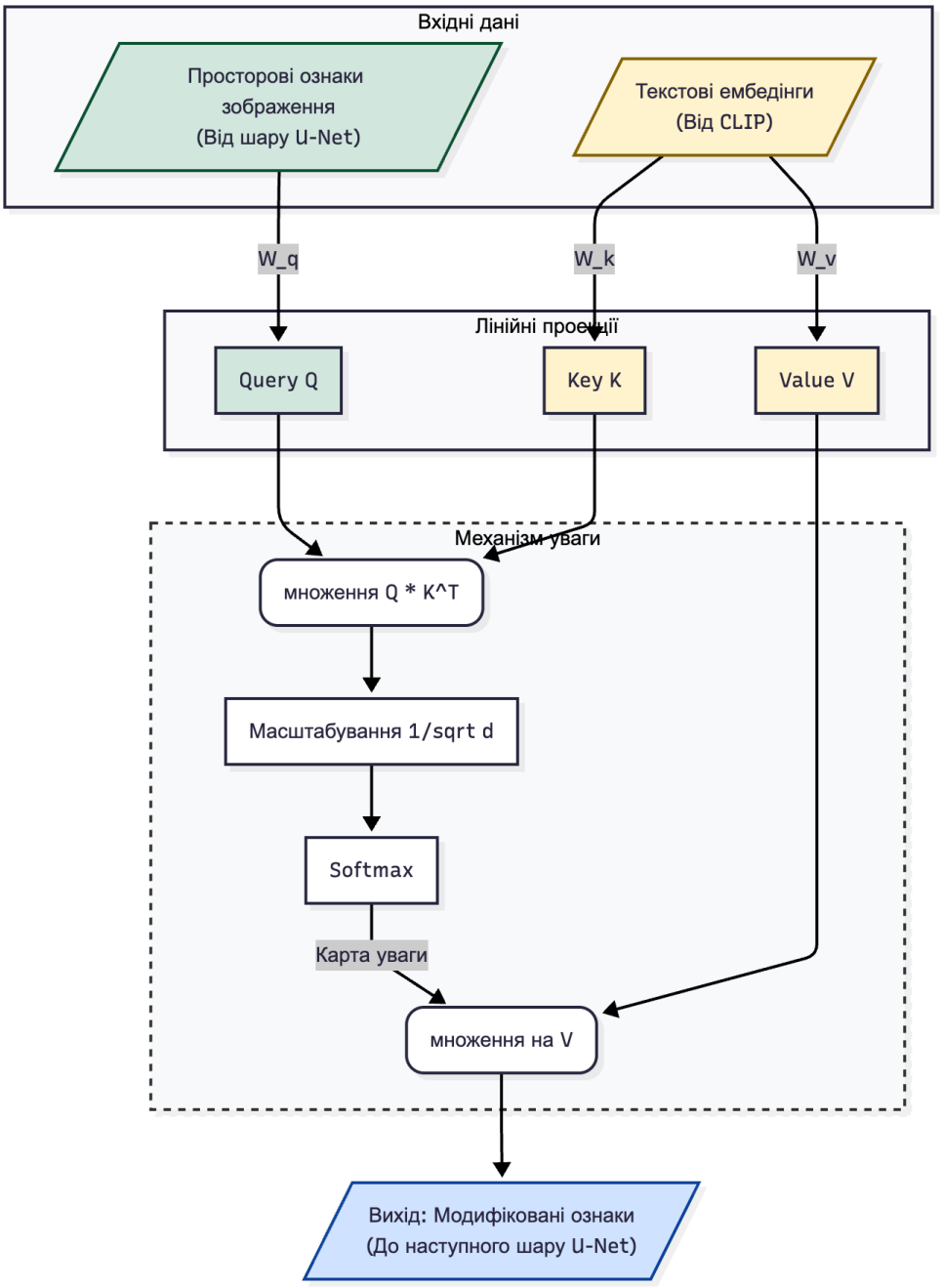


Рис. 2.4 Схема інтеграції текстових ембедінгів у шари U-Net через механізм Cross-Attention

2.2.3 Інженерія промптів (Prompt Engineering) як метод керування

Оскільки ми розробляємо автоматизовану систему, критично важливо формалізувати процес створення вхідних запитів. "Сирий" запит користувача (наприклад, "стіна") є недостатнім для генерації PBR-матеріалу.

Розроблений метод передбачає алгоритмічну модифікацію промпту (Prompt Augmentation). Вхідний запит P_{raw} автоматично доповнюється набором технічних дескрипторів D_{tech} та стилістичних модифікаторів M_{style} (2.9):

$$P_{final} = P_{raw} \oplus D_{tech} \oplus M_{style}, \tag{2.9}$$

Де $\oplus$ - операція конкатенації рядків.

Для задач генерації PBR-текстур було визначено оптимальний набір дескрипторів D_{tech}, який включає токени: *«texture», «top down view», «flat lighting», «seamless», «4k», «highly detailed»*. Це дозволяє мінімізувати перспективні спотворення та ефекти освітлення на етапі генерації, наближаючи результат до чистої карти Albedo.

Також важливим елементом є використання негативного промпту (Negative Prompt) - вектора c_{neg}, що описує небажані ознаки (наприклад, *«shadows», «watermark», «blur», «perspective»*). Генерація відбувається шляхом руху в напрямку вектора $c - c_{neg}$ за формулою класифікаторно-вільного керування (Classifier-Free Guidance) (2.10):

$$\widetilde{\epsilon_\theta}(x_t, c) = \epsilon_\theta(x_t, c_{neg}) + w \cdot \Big(\epsilon_\theta(x_t, c) - \epsilon_\theta(x_t, c_{neg})\Big), \tag{2.10}$$

де w - вага керування (Guidance Scale), що визначає, наскільки суворо модель має слідувати тексту. Експериментально встановлено, що для текстур оптимальне значення $w \in [7.0,9.0]$.

Таблиця 2.1

Вплив модифікаторів промпту на параметри генерації PBR-матеріалів

Категорія токена	**Приклад токена**	**Вплив на латентний вектор (Інтерпретація)**	**Очікуваний візуальний ефект**
Об'єкт	brick wall	Формує базову геометрію та структуру патерну.	Поява прямокутних блоків.
Вид	top down view	Обмежує перспективні спотворення, наближаючи проекцію до ортогональної.	Відсутність горизонту, плоске зображення.
Освітлення	flat lighting	Мінімізує ваги, що відповідають за генерацію тіней та бліків.	Рівномірний колір (Albedo), придатний для PBR.
Деталізація	highly detailed	Активує високочастотні шуми на останніх кроках дифузії.	Поява дрібних тріщин, пор, зернистості.
Стиль	photorealistic	Зсуває розподіл у бік фотографічних текстур, пригнічуючи стилізацію "під малюнок".	Реалістичні матеріали.
Безшовність	seamless	Спроба нейромережі узгодити граничні пікселі (не гарантовано математично).	Зменшення видимості стиків.

2.3 Метод генерації карт нормалей (Normal Maps) на основі дифузних текстур

Критичним недоліком базових дифузійних моделей є те, що вони генерують лише карту кольору (Albedo/Diffuse). Для повноцінного використання в ігровому рушії необхідно відновити інформацію про рельєф поверхні. У даній роботі пропонується гібридний підхід, що поєднує нейромережеву оцінку глибини (Monocular Depth Estimation) з класичними методами комп'ютерного зору для обчислення нормалей поверхні.

2.3.1 Оцінка глибини сцени (Depth Estimation)

Першим етапом запропонованого алгоритму є перетворення згенерованого RGB-зображення I на карту глибини D, де значення кожного пікселя $D(x, y)$ відповідає відстані від камери до точки поверхні або її відносній висоті (Height Map).

Оскільки ми працюємо з одним зображенням, задача відноситься до класу Monocular Depth Estimation. Традиційні методи (Shape from Shading) тут неефективні через складне освітлення. Тому в роботі використано попередньо навчену згорткову нейронну мережу (CNN) архітектури MiDaS або LeReS (Learning Recovering Surface) [15], яка апроксимує функцію відображення (2.11):

$$D = \mathcal{F}_{depth}(I; \theta_{depth}), \tag{2.11}$$

де $I \in R^{H \times W \times 3}$ - вхідне зображення;

$D \in R^{H \times W}$ - матриця глибини, нормалізована в діапазоні $[0, 1]$;

θ_{depth} - ваги моделі оцінки глибини.

Отримана карта глибини D є скалярним полем, яке описує топологію поверхні текстури.

Рис. 2.5 Структурна схема модуля генерації карт нормалей

2.3.2 Обчислення градієнтів поверхні

Карта нормалей (Normal Map) - це текстура, де в кожному пікселі закодовано вектор нормалі $n = (n_x, n_y, n_z)$, перпендикулярний до дотичної площини поверхні в даній точці.

Для знаходження вектора n необхідно обчислити градієнт зміни висоти по осях X та Y. Для дискретного зображення це робиться за допомогою операторів згортки. У роботі використовується оператор Собеля (Sobel Operator), який є стійким до шуму завдяки гауссівському згладжуванню.

Градієнти G_xта G_y обчислюються як згортка карти глибини D з ядрами K_x та K_y (2.12):

$$G_x = D * \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, G_y = D * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \quad (2.12)$$

де $*$ - операція згортки.

Величини $G_x(x, y)$та$G_y(x, y)$ показують швидкість зміни висоти рельєфу в точці (x, y).

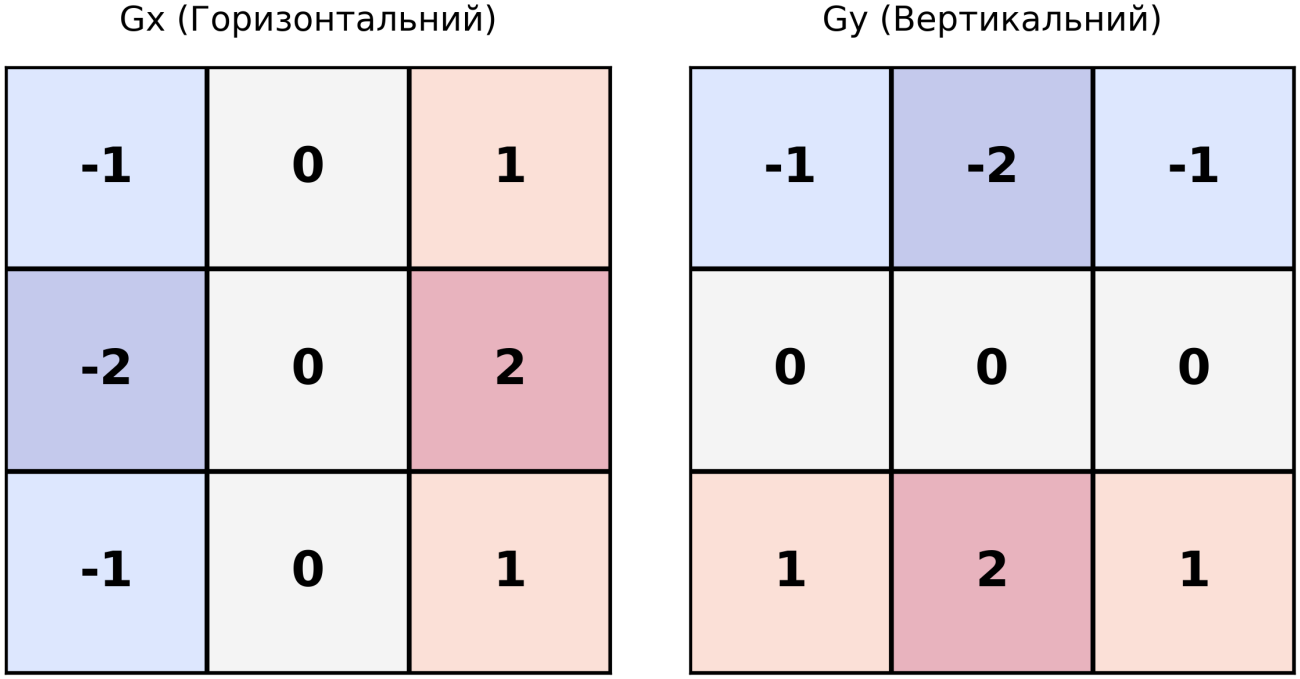


Рис. 2.6 Ядра оператора Собеля для обчислення градієнтів

2.3.3 Формування та нормалізація векторів

Маючи градієнти, ми можемо побудувати вектори дотичних до поверхні. Вектор нормалі n визначається як векторний добуток дотичних векторів, або у спрощеному вигляді через градієнти (2.13):

$$n_{raw}(x,y) = \begin{pmatrix} -s \cdot G_x(x,y) \\ -s \cdot G_y(x,y) \\ 1 \end{pmatrix}, \quad (2.13)$$

де s - масштабний коефіцієнт (Strength), що дозволяє регулювати інтенсивність рельєфу (чим більше s, тим "глибшою" здається текстура);

Компонента $z = 1$ вказує, що базова поверхня дивиться "вгору" (або на глядача).

Оскільки карта нормалей вимагає одиничних векторів, отриманий вектор n_{raw} необхідно нормалізувати (2.14):

$$n = \frac{n_{raw}}{|n_{raw}|} = \frac{1}{\sqrt{(sG_x)^2 + (sG_y)^2 + 1}} \begin{pmatrix} -sG_x \\ -sG_y \\ 1 \end{pmatrix}, \quad (2.14)$$

В результаті ми отримуємо вектор $\mathbf{n}$, компоненти якого лежать у діапазоні $[-1,1]$.

2.3.4 Кодування у простір кольорів RGB

Останнім етапом є перетворення математичного вектора у формат зображення (RGB), яке зможе прочитати ігровий рушій. Оскільки колірний канал приймає значення $[0,255]$ (для 8-бітних зображень), необхідно виконати лінійне відображення діапазону $[-1,1] \rightarrow [0,255]$:

$$R = \lfloor (n\_x + 1) \cdot 0.5 \cdot 255 \rfloor$$

$$G = \lfloor (n_y + 1) \cdot 0.5 \cdot 255 \rfloor$$

$$B = \lfloor (n\_z + 1) \cdot 0.5 \cdot 255 \rfloor$$

Таким чином формується фінальна карта нормалей, де:

R (Червоний): Відхилення нормалі по горизонталі.

G (Зелений): Відхилення нормалі по вертикалі.

В (Синій): Відхилення по глибині (тому карти нормалей переважно сині/фіолетові, бо $n_z \approx 1$).

Запропонований метод дозволяє автоматично генерувати коректні карти нормалей безпосередньо з вихідних даних дифузійної моделі, оминаючи трудомісткі етапи скульптингу та запікання, описані в розділі 1.2.

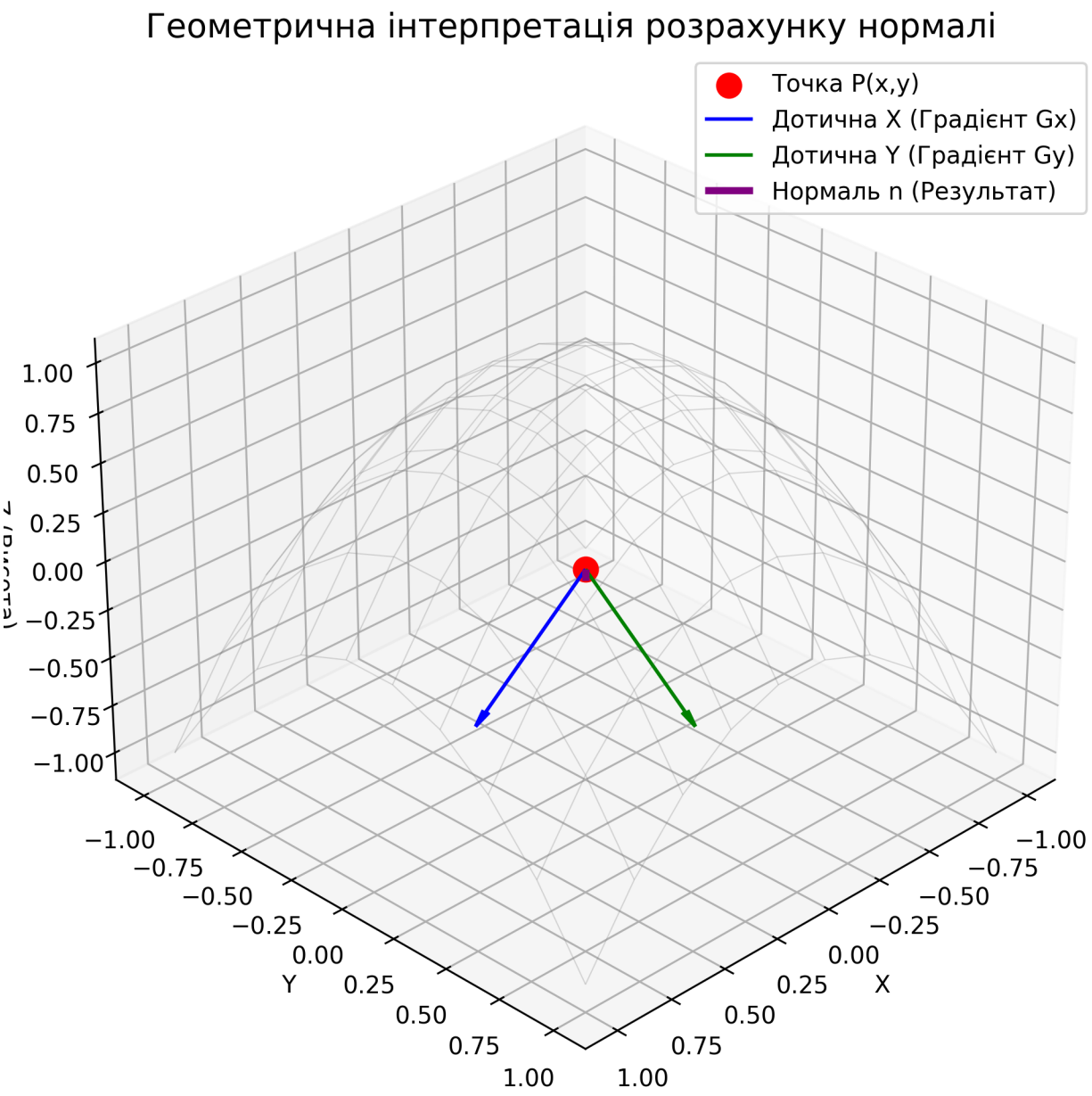


Рис. 2.7 Геометрична інтерпретація розрахунку нормалі поверхні на основі градієнтів висоти

2.4 Критерії оцінки якості згенерованого контенту

Для об'єктивного аналізу ефективності розробленого методу недостатньо покладатися виключно на суб'єктивне візуальне сприйняття. Необхідно визначити набір формалізованих метрик, які дозволяють кількісно оцінити якість згенерованих зображень, їхню відповідність вхідному текстовому опису та обчислювальну ефективність алгоритму. У даній роботі використовується комплексна система оцінювання, що включає три ключові показники: відстань Фреше (FID), оцінку CLIP (CLIP Score) та час інференсу.

2.4.1 Відстань Фреше (Fréchet Inception Distance - FID)

Метрика FID є стандартом де-факто для оцінки якості зображень, згенерованих нейронними мережами [16]. Вона вимірює відстань між розподілом ознак реальних зображень та розподілом ознак згенерованих зображень у прихованому просторі попередньо навченої нейромережі Inception v3.

Математично припускається, що вектори ознак реальних зображень x та згенерованих зображень g підпорядковуються багатовимірному нормальному розподілу (2.16):

$$x \sim \mathcal{N}(\mu_x, \Sigma_x), \quad g \sim \mathcal{N}(\mu_g, \Sigma_g), \tag{2.16}$$

де μ_x, μ_g - вектори середніх значень ознак;

Σ_x, Σ_g - коваріаційні матриці ознак для реальних та згенерованих даних відповідно.

Відстань Фреше між двома гауссівськими розподілами обчислюється за формулою, відомою також як 2-відстань Вассерштейна (2.17):

$$FID = |\mu_x - \mu_g|^2 + Tr\left(\Sigma_x + \Sigma_g - 2\left(\Sigma_x\Sigma_g\right)^{1/2}\right), \tag{2.17}$$

де $|\cdot|^2$ - квадрат евклідової відстані;

$Tr(\cdot)$ - слід матриці (сума діагональних елементів).

Фізичний зміст: Чим менше значення FID, тим ближче розподіл згенерованих зображень до реальних, що свідчить про вищу візуальну якість та різноманітність.

Для сучасних моделей (Stable Diffusion) значення FID на стандартних датассетах (наприклад, COCO) зазвичай знаходиться в діапазоні [7.0, 15.0].

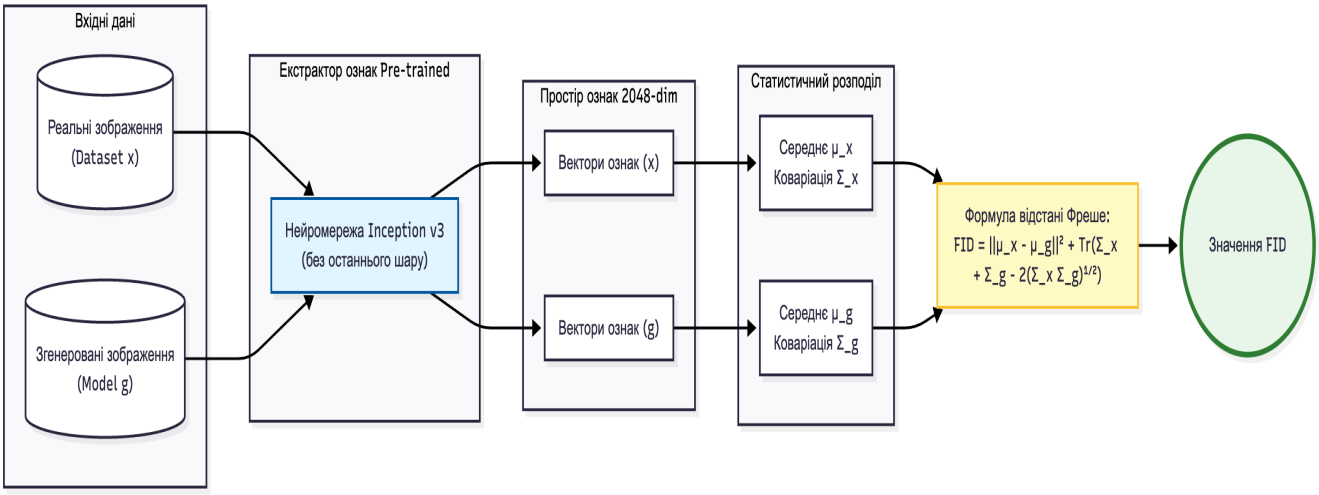


Рис. 2.8 Схема розрахунку метрики FID через простір ознак Inception v3

2.4.2 Оцінка семантичної відповідності (CLIP Score)

Якщо FID оцінює реалістичність, то CLIP Score оцінює релевантність - наскільки точно зображення відповідає заданому текстовому промпту. Ця метрика базується на моделі CLIP, яка навчається відображати зображення та текст у спільний векторний простір.

Нехай I - згенероване зображення, а T - вхідний текст. Процес обчислення складається з наступних кроків:

Зображення I проходить через візуальний енкодер $\mathcal{E}_{image}$ для отримання вектора v_I.

Текст T проходить через текстовий енкодер $\mathcal{E}_{text}$ для отримання вектора v_T.

Вектори нормалізуються: $\widehat{v_I} = \frac{v_I}{|v_I|}, \quad \widehat{v_T} = \frac{v_T}{|v_T|}$.

CLIP Score визначається як масштабована косинусна подібність між цими векторами (2.18):

$$S_{CLIP}(I,T) = w \cdot \max(0, \cos(\widehat{v_I}, \widehat{v_T})) = w \cdot \max\left(0, \widehat{v_I} \cdot \widehat{v_T^T}\right), \qquad (2.18)$$

де w - масштабний коефіцієнт (зазвичай w=2.5 або w=100, залежно від реалізації), який розширює динамічний діапазон значень.

Високе значення CLIP Score означає, що модель "зрозуміла" запит і згенерувала семантично коректний контент.

2.4.3 Показники обчислювальної ефективності

Для оцінки придатності методу до впровадження у реальний виробничий процес (Production Pipeline) критично важливими є показники швидкодії.

Час інференсу (Latency): Час t_{gen}, необхідний для генерації одного ассету (batch size = 1). Вимірюється в секундах (2.19).

$$t_{gen} = t_{load} + N_{steps} \cdot t_{step} + t_{decode}, \quad (2.19)$$

де t_{load} - час завантаження моделей у VRAM;

N_{steps} - кількість кроків дифузії (наприклад, 30);

t_{step} - час обробки одного кроку U-Net;

t_{decode} - час роботи VAE-декодера.

Використання відеопам'яті (VRAM Usage): Піковий обсяг пам'яті GPU, необхідний для роботи моделі. Це критичний параметр для локального запуску на робочих станціях розробників.

Продуктивність (Throughput): Кількість зображень, які система може згенерувати за одиницю часу (img/min).

Сукупність цих метрик (FID, CLIP Score, Latency) дозволяє провести всебічну оцінку розробленого гібридного методу та порівняти його з існуючими аналогами у Розділі 3.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1 Обґрунтування вибору засобів розробки

Ефективність вирішення поставленої науково-практичної задачі значною мірою залежить від правильного вибору технологічного стеку. Оскільки розробка гібридного методу генерації PBR-матеріалів вимагає роботи зі складними нейромережевими архітектурами, обробки зображень та побудови графічного інтерфейсу, вибір засобів реалізації базувався на критеріях швидкодії, наявності готових бібліотек, підтримки спільноти та кросплатформеності.

3.1.1 Мова програмування: Python як стандарт галузі AI

Для реалізації програмного комплексу було обрано мову програмування Python 3.11. Цей вибір зумовлений домінуючим положенням Python у сфері Data Science та Machine Learning (ML).

Хоча мови низького рівня, такі як C++, забезпечують вищу швидкість виконання коду, вони значно програють у швидкості розробки (Time-to-Market) та гнучкості при проведенні експериментів. Python, будучи інтерпретованою мовою з динамічною типізацією, дозволяє зосередитися на логіці алгоритмів, а не на управлінні пам'яттю.

Ключові переваги Python для даної роботи:

- Екосистема бібліотек: Наявність оптимізованих бібліотек (NumPy, Pandas, OpenCV), які реалізовані на C/C++, що нівелює недоліки швидкодії самого інтерпретатора Python.
- Інтероперабельність: Легка інтеграція з GPU-обчисленнями.
- Підтримка спільноти: Згідно з індексом TIOBE та звітами GitHub Octoverse, Python є найпопулярнішою мовою для наукових обчислень.

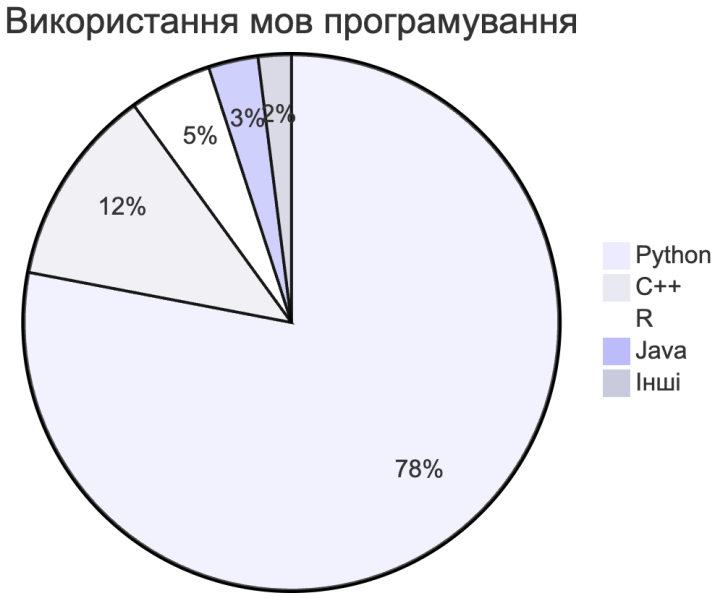


Рис. 3.1 Частка використання мов програмування у проектах Machine Learning (дані GitHub 2024)

3.1.2 Фреймворк глибокого навчання: PyTorch vs TensorFlow

На ринку існує два основних фреймворки для роботи з нейромережами: TensorFlow (від Google) та PyTorch (від Meta AI). Для виконання даної роботи було обрано PyTorch.

Вибір на користь PyTorch ґрунтується на наступних факторах:

- Динамічний обчислювальний граф (Dynamic Computation Graph): PyTorch дозволяє змінювати архітектуру мережі "на льоту" під час виконання, що критично важливо для налагодження та експериментів з новими архітектурами. TensorFlow (у класичній версії) використовує статичний граф.
- "Pythonic" стиль: Синтаксис PyTorch інтуїтивно зрозумілий python-розробникам, тоді як TensorFlow має більш високий поріг входження.
- Домінування у дослідженнях: Статистика конференцій (CVPR, NeurIPS) показує, що понад 80% нових наукових статей з генеративного ШІ реалізуються саме на PyTorch.

Порівняльна характеристика фреймворків наведена в таблиці 3.1.

Таблиця 3.1

Порівняльний аналіз фреймворків глибокого навчання

Критерій порівняння	TensorFlow	PyTorch (Обрано)
Тип обчислювального графа	Статичний (переважно)	Динамічний
Зручність налагодження	Низька (потрібен tf.print)	Висока (стандартний Python debugger)
Підтримка Apple Silicon (MPS)	Обмежена	Повна (Native support)
Екосистема Diffusion Models	Обмежена	Основна платформа для Hugging Face Diffusers
Сфера застосування	Промисловий деплой (Production)	Наукові дослідження (Research)

3.1.3 Спеціалізовані бібліотеки: Diffusers та Transformers

Замість написання архітектури U-Net з нуля, що є недоцільним в рамках магістерської роботи через величезний обсяг коду, було використано бібліотеку Diffusers від компанії Hugging Face.

Ця бібліотека надає уніфікований інтерфейс для роботи з попередньо навченими дифузійними моделями. Вона забезпечує:

- Завантаження ваг моделей (Stable Diffusion v1.5, SDXL).
- Реалізацію різних планувальників шуму (Schedulers: DDIM, Euler Ancestral, DPM++).
- Оптимізацію пам'яті (Attention Slicing, Model Offloading), що є критичним для запуску на локальному обладнанні.

Для обробки тексту (токінезації) використовується бібліотека Transformers, яка містить реалізацію моделі CLIP.

3.1.4 Апаратна платформа та оптимізація: Apple Silicon (MPS)

Важливою особливістю даної розробки є адаптація під апаратну платформу Apple Silicon (чіпи серії M). Традиційно навчання та інференс нейромереж вимагають відеокарт NVIDIA з підтримкою технології CUDA. Однак, поширеність ноутбуків Apple серед розробників вимагає пошуку альтернативних рішень.

У роботі використано технологію MPS (Metal Performance Shaders). Це API, яке дозволяє використовувати графічний процесор (GPU) на пристроях macOS для прискорення тензорних обчислень PyTorch.

Схема взаємодії апаратного та програмного забезпечення:

1. PyTorch відправляє тензори на пристрій mps.
2. API Metal транслює команди у низькорівневий код для GPU Apple.
3. Використовується об'єднана пам'ять (Unified Memory Architecture), що дозволяє CPU і GPU мати доступ до одних даних без копіювання через шину PCIe (як це відбувається у ПК з NVIDIA).

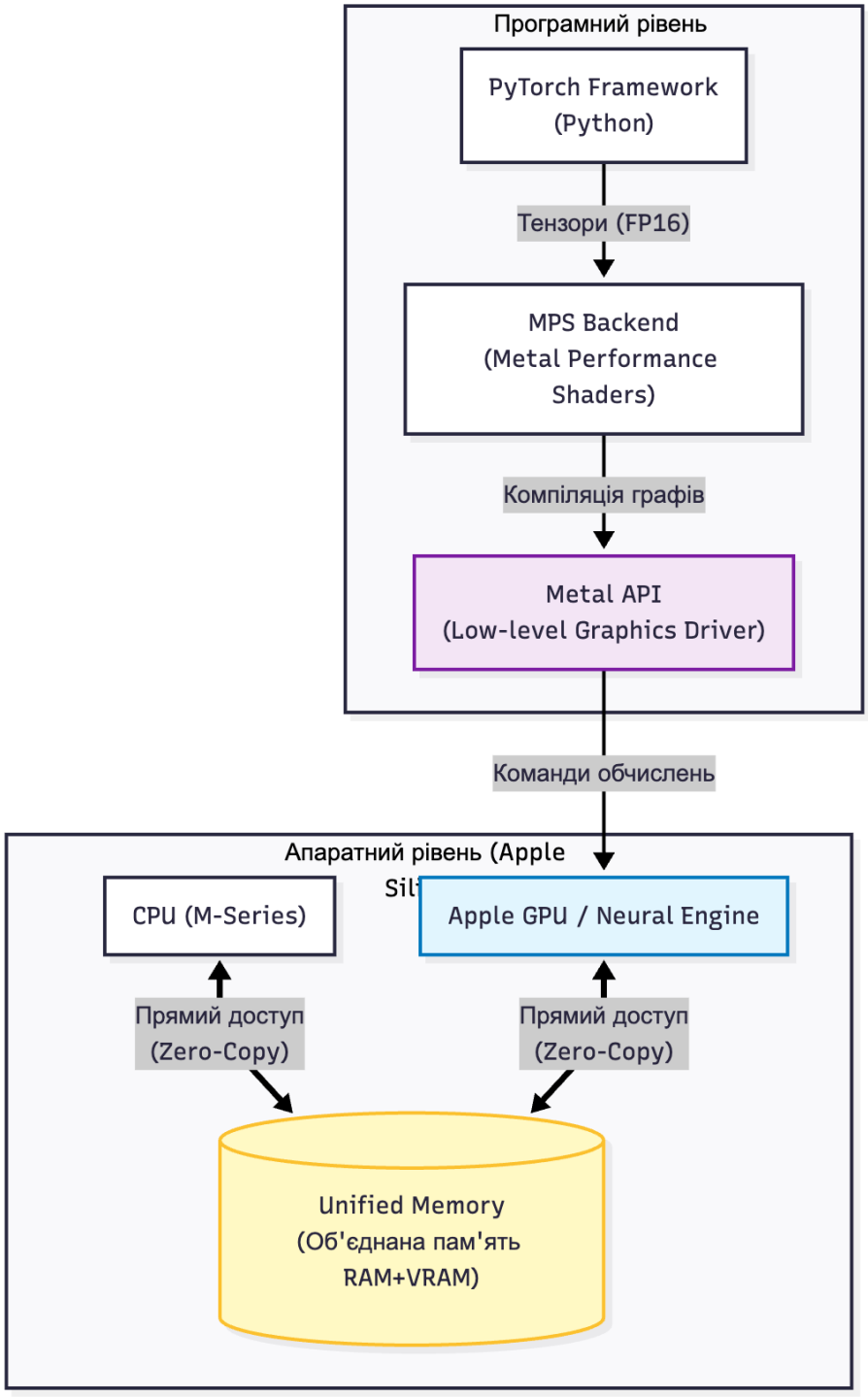


Рис. 3.2 Схема взаємодії програмного забезпечення з апаратним прискорювачем MPS та об'єднаною пам'яттю

Для забезпечення роботи на пристроях з обмеженим обсягом пам'яті (8-16 ГБ) було застосовано метод зниженої точності FP16 (Half-Precision Floating Point). Використання 16-бітних чисел замість стандартних 32-бітних дозволило зменшити споживання відеопам'яті моделлю Stable Diffusion v1.5 з 8 ГБ до ~4 ГБ, що зробило можливим запуск генерації на MacBook Air.

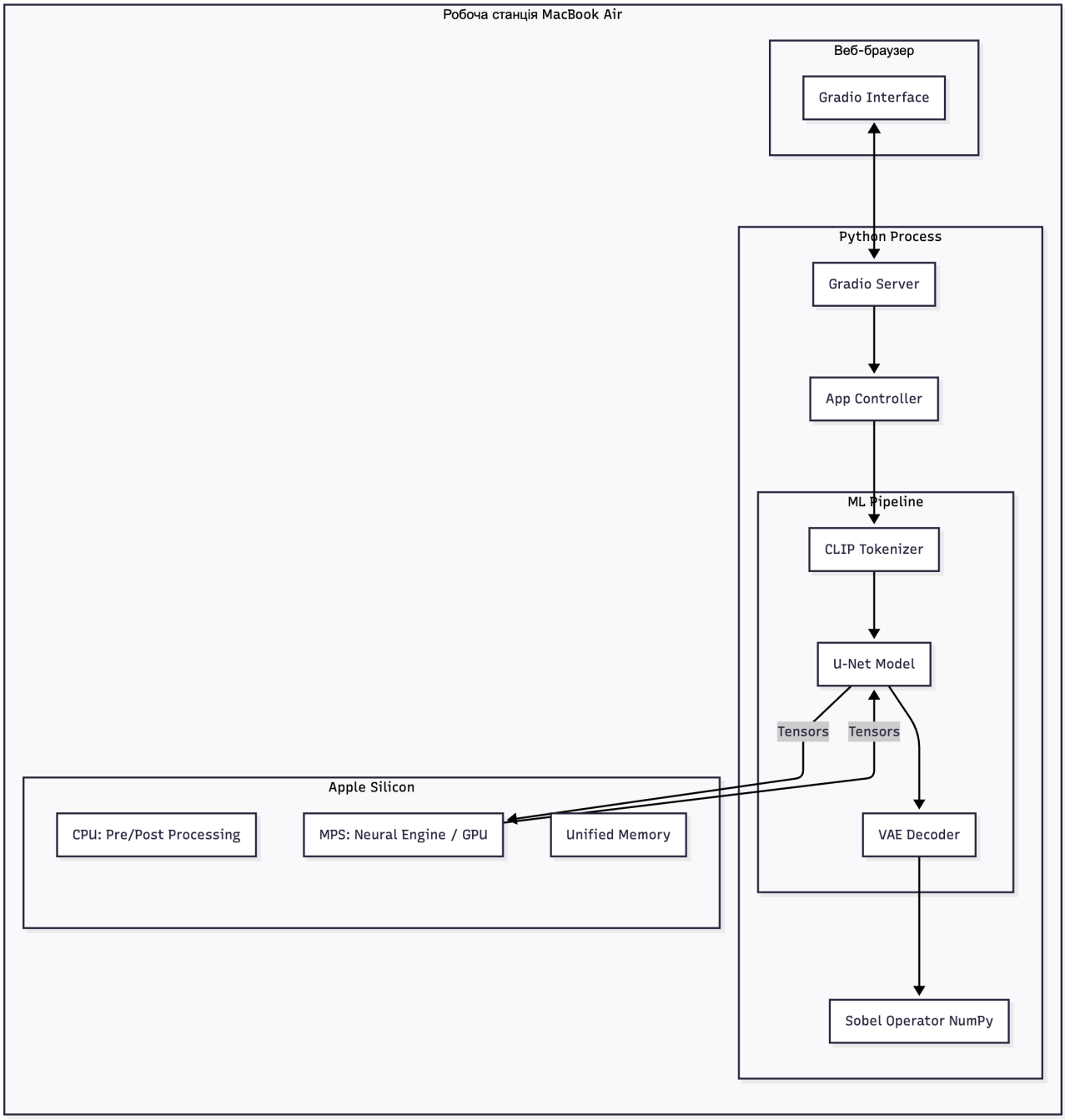


Рис. 3.3 Діаграма розгортання програмного комплексу
на апаратній платформі Apple Silicon

3.2 Архітектура програмного комплексу та опис інтерфейсу користувача

Для забезпечення модульності, розширюваності та зручності використання, програмний комплекс було спроектовано з дотриманням принципів об'єктно-орієнтованого програмування (ООП). Архітектура системи побудована за шаблоном Model-View-Controller (MVC), адаптованим для ML-застосунків, де

модель (Model) відповідає за інференс нейромереж, представлення (View) - за веб-інтерфейс, а контролер (Controller) - за логіку обробки подій.

3.2.1 Компонентна діаграма системи

Програмний комплекс складається з чотирьох основних модулів:

1. Core Module (Ядро): Відповідає за ініціалізацію пайплайну Stable Diffusion, завантаження ваг у пам'ять (VRAM) та керування пристроєм обчислення (mps або cuda).
2. Prompt Processor: Модуль обробки тексту, який виконує токенізацію та автоматичне доповнення промпту технічними дескрипторами (PBR, 4k, seamless).
3. Post-Processing Unit: Блок, що реалізує алгоритми тайлінгу та генерації карт нормалей.
4. UI Server: Веб-сервер на базі бібліотеки Gradio, який забезпечує взаємодію з користувачем.

Взаємодія цих компонентів представлена на діаграмі класів (рисунок 3.4).

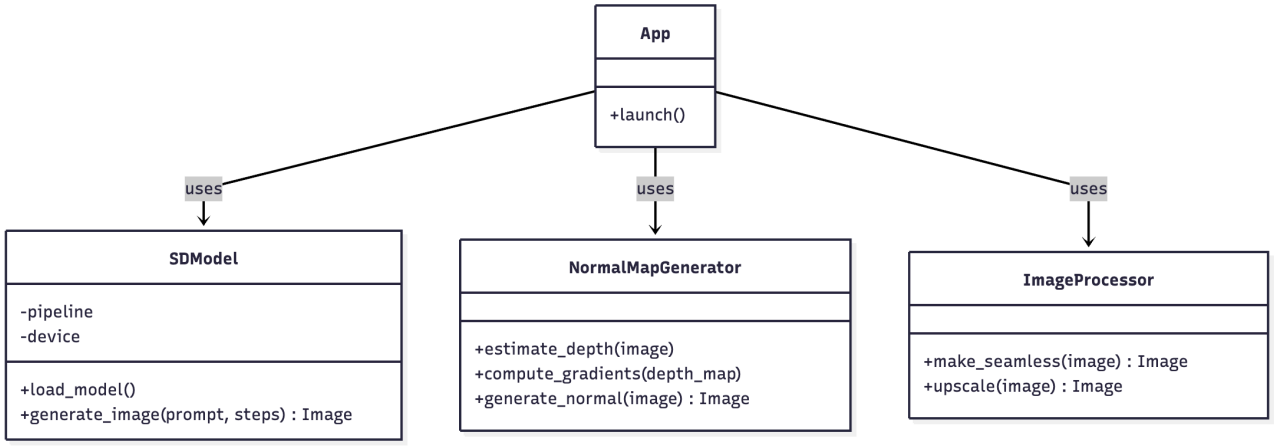


Рис. 3.4 Діаграма основних класів програмного комплексу.

3.2.2 Розробка графічного інтерфейсу користувача (GUI)

Для забезпечення зручності роботи з системою було розроблено веб-інтерфейс за допомогою бібліотеки Gradio. Це дозволяє запускати генерацію локально у браузері без необхідності використання командного рядка.

Інтерфейс спроектовано за принципом "єдиного вікна" і містить такі функціональні зони:

Панель налаштувань (ліворуч):

- Текстове поле для введення основного промпту.
- Поле для негативного промпту.
- Слайдери для налаштування параметрів генерації: кількість кроків (Steps), вага промпту (Guidance Scale), зерно генерації (Seed).

Панель результатів (праворуч):

- Вікно попереднього перегляду згенерованої текстури (Albedo).
- Вікно попереднього перегляду карти нормалей (Normal Map).
- Кнопка "Завантажити" для збереження ассетів.

Такий підхід дозволяє художнику миттєво бачити результат зміни параметрів та оцінювати якість PBR-матеріалу в реальному часі.

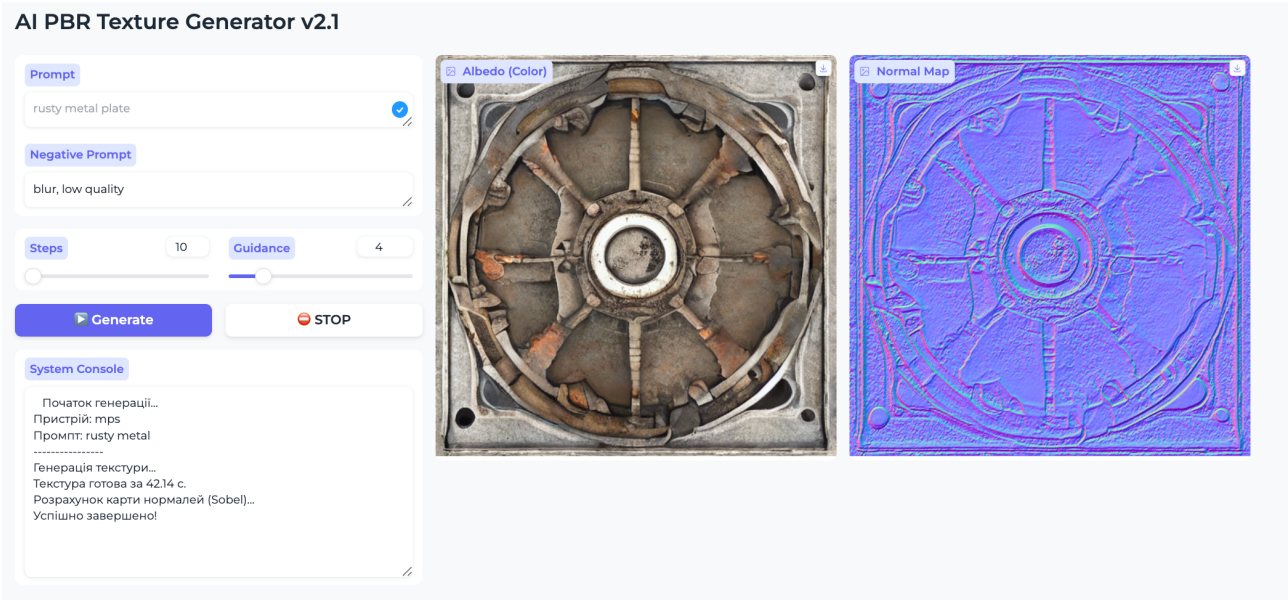


Рис. 3.5 Графічний інтерфейс користувача розробленої системи з прикладом генерації PBR-матеріалу (металу)

3.3 Реалізація алгоритмів генерації та пост-обробки

Програмна реалізація методу складається з двох послідовних етапів: інференсу нейромережі для отримання базового зображення та детермінованої пост-обробки для створення карт нормалей.

3.3.1 Реалізація генеративного пайплайну

Основним класом, що відповідає за генерацію, є DiffusionPipeline. У розробленому модулі generate_texture реалізовано наступну логіку:

Формування розширеного промпту: Вхідний рядок користувача автоматично конкатенується з інженерними дескрипторами (seamless texture, top down view, PBR material). Це забезпечує стабільну якість генерації без необхідності введення складних технічних команд користувачем.

Ініціалізація шуму: Використовується генератор псевдовипадкових чисел (RNG) на пристрої MPS для створення початкового латентного тензора.

Ітеративне знешумлення: Виконується цикл з 20-50 кроків (параметр steps), де на кожній ітерації модель U-Net передбачає шум, а планувальник (Scheduler) віднімає його, формуючи зображення.

3.3.2 Програмна реалізація генерації Normal Maps

Оскільки базова модель генерує лише RGB-зображення, для створення PBR-матеріалу реалізовано функцію create_normal_map. Вона базується на математичній моделі, описаній у підрозділі 2.3, і використовує бібліотеку NumPy для векторних обчислень.

Алгоритм реалізовано наступним чином:

Конвертація у карту висот: RGB-зображення перетворюється у відтінки сірого (Grayscale), де яскравість пікселя $I(x, y)$ інтерпретується як відносна висота $h(x, y)$.

Обчислення градієнтів: Застосовується дискретний оператор диференціювання (центральна різниця) для знаходження похилу поверхні по осях X та Y:

$$gx[:,1:-1] = \left(img_gray[:,2:] - img_gray[:,:-2]\right) * 0.5$$

$$gy[1:-1,:] = \left(img_gray[2:,:] - img_gray[:-2,:]\right) * 0.5$$

Нормалізація векторів: Обчислені компоненти нормалі (n_x, n_y, n_z) нормалізуються до одиничної довжини та перемасштабовуються у діапазон [0, 255] для збереження у форматі PNG.

Результатом роботи цієї функції є карта нормалей у дотичному просторі (Tangent Space Normal Map), яка сумісна з усіма сучасними ігровими рушіями (Unity, Unreal Engine).

3.3.3 Реалізація механізму переривання генерації (Асинхронність)

Однією з ключових проблем при розробці користувацьких інтерфейсів (UI) для систем машинного навчання є блокуючий характер операцій інференсу (inference blocking). Процес зворотного знешумлення (denoising loop) у дифузійній моделі є ітеративним і високонавантаженим обчислювальним процесом, який виконується на графічному прискорювачі (GPU/MPS). У стандартній однопотоковій реалізації це призводить до повного "заморожування" інтерфейсу програми: користувач втрачає можливість взаємодіяти з системою до завершення генерації.

Особливої актуальності ця проблема набуває в умовах використання споживчого апаратного забезпечення (наприклад, ноутбуків), де час генерації може сягати кількох хвилин (як було виявлено під час стрес-тестів). Відсутність можливості перервати процес (наприклад, якщо користувач помітив помилку у промпті) є критичним недоліком юзабіліті.

Для вирішення цієї задачі було розроблено та імплементовано механізм асинхронного переривання інференсу (Asynchronous Inference Interruption), що базується на ін'єкції функцій зворотного виклику (Callbacks) у внутрішній цикл пайплайну бібліотеки Diffusers.

Архітектура керування станом

Реалізація базується на використанні глобального семафора стану (State Flag) та подійної моделі бібліотеки Gradio. Алгоритм роботи механізму можна описати наступними кроками:

Ініціалізація стану: Введено глобальну змінну IS_GENERATING (тип bool), яка виступає індикатором активності процесу. При натисканні кнопки "Generate" вона встановлюється в True.

Обробка події зупинки: Кнопка "STOP" в інтерфейсі прив'язана до окремої функції stop_generation(), яка виконується в окремому потоці обробки подій і змінює значення IS_GENERATING на False.

Ін'єкція Callback-функції: У метод pipe(...) бібліотеки Diffusers передається посилання на спеціальну функцію callback_fn. Архітектура пайплайну влаштована таким чином, що ця функція викликається автоматично після завершення кожного кроку дифузії (наприклад, після кожного з 30 кроків).

Перевірка умови виходу: Всередині callback_fn відбувається перевірка стану семафора. Якщо IS_GENERATING == False, функція ініціює примусове переривання виконання через виклик виключення KeyboardInterrupt або Exception.

Інтеграція з графічним інтерфейсом

Для коректної роботи в середовищі Gradio також використано механізм скасування черги подій. При конфігурації інтерфейсу подія натискання кнопки btn_stop налаштована з параметром cancels=[run_event]. Це повідомляє серверу Gradio про необхідність розірвати з'єднання з функцією-генератором, що дозволяє миттєво звільнити ресурси інтерфейсу для нових задач, навіть якщо фоновий процес Python ще завершує очищення пам'яті.

Схема взаємодії компонентів при перериванні генерації зображена на рисунку 3.6.

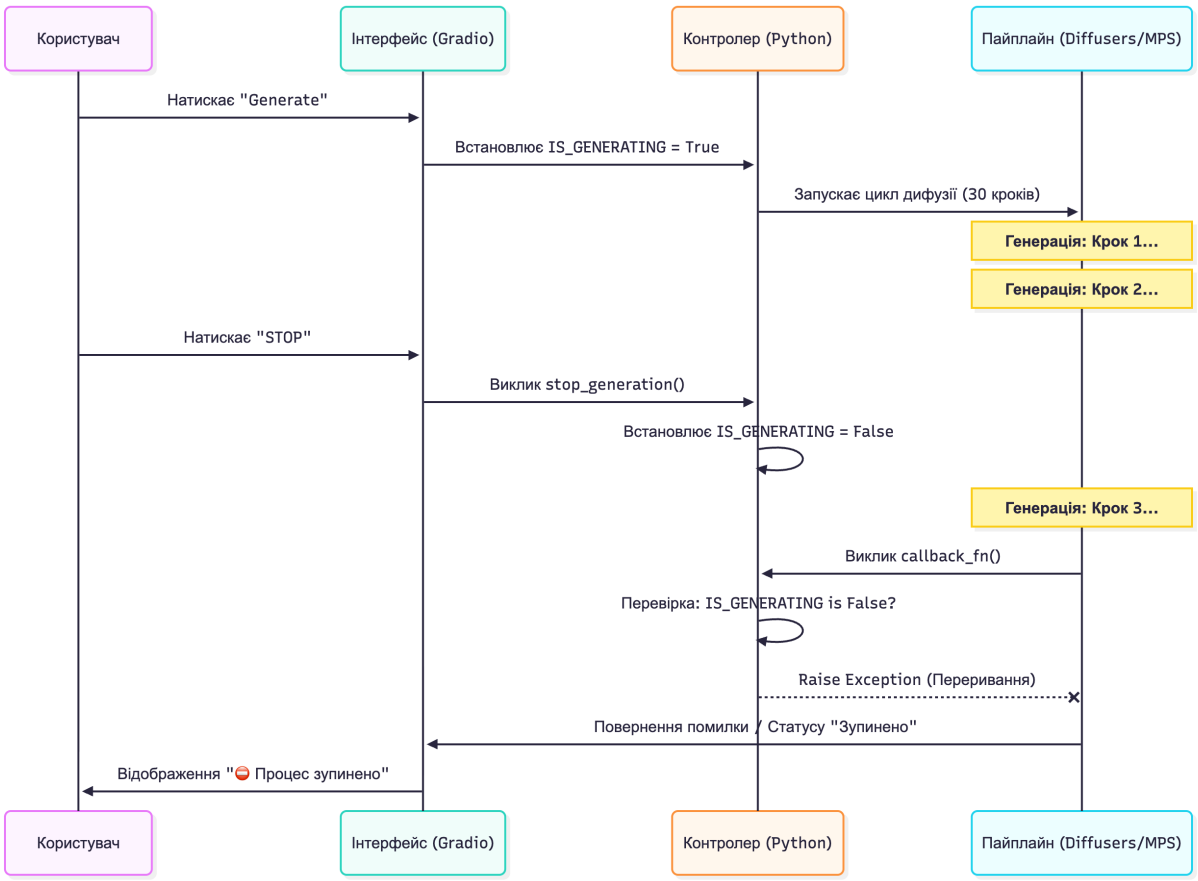


Рис. 3.6 Діаграма послідовності подій при асинхронному перериванні процесу генерації

Завдяки реалізації цього підходу було досягнуто високої чуйності системи (responsiveness). Час реакції на команду зупинки не перевищує часу виконання одного кроку дифузії (приблизно 1.5-2 секунди на MacBook Air при повному навантаженні), що є прийнятним показником для інтерактивних систем.

3.3.4 Розробка системи логування подій та моніторингу стану

Враховуючи високу обчислювальну складність генеративних моделей, час виконання однієї ітерації створення ассету може варіюватися від 40 до кількох хвилин (у випадку тротлінгу). У таких умовах критично важливим є забезпечення прозорості роботи системи для користувача. "Ефект чорної скриньки", коли програма зависає на тривалий час без зворотного зв'язку, є неприпустимим для професійного інструментарію.

Для вирішення цієї проблеми було розроблено модуль логування в реальному часі (Real-time Logging), який транслює внутрішній стан бекенду (Python) на фронтенд (Gradio).

Архітектура потокового виводу даних

Технічна реалізація системи логування базується на патерні проєктування Generator. На відміну від стандартних функцій, які повертають результат лише після повного завершення роботи (return), функція генерації реалізована як Python-генератор, що використовує інструкцію yield.

Це дозволяє створювати потоковий вивід даних (Data Streaming). Процес оновлення інтерфейсу відбувається ітеративно:

- Стейдж ініціалізації: Система повідомляє про визначення апаратного забезпечення (наприклад, *"Пристрій: mps"*).
- Стейдж інференсу: Під час роботи U-Net оновлюється статус прогресу.
- Стейдж пост-процесингу: Логується запуск алгоритмів комп'ютерного зору (Собеля).
- Стейдж завершення: Виводиться точний час виконання операції.

Обробка виключних ситуацій (Exception Handling)

Важливою частиною системи логування є перехоплення помилок. У ході експериментів було виявлено, що найбільш часті збої пов'язані з нестачею відеопам'яті (RuntimeError: Out of memory) або перериванням процесу користувачем.

Розроблений модуль огортає критичні секції коду у блоки try...except. У разі виникнення помилки програма не завершує роботу аварійно (crash), а коректно

зупиняє пайплайн, очищує пам'ять (Garbage Collection) і виводить детальний опис помилки у вікно консолі інтерфейсу. Це дозволяє користувачеві зрозуміти причину збою (наприклад, "зменшіть кількість кроків") і продовжити роботу без перезапуску сервера.

Схема потоку даних при логуванні зображена на рисунку 3.7.

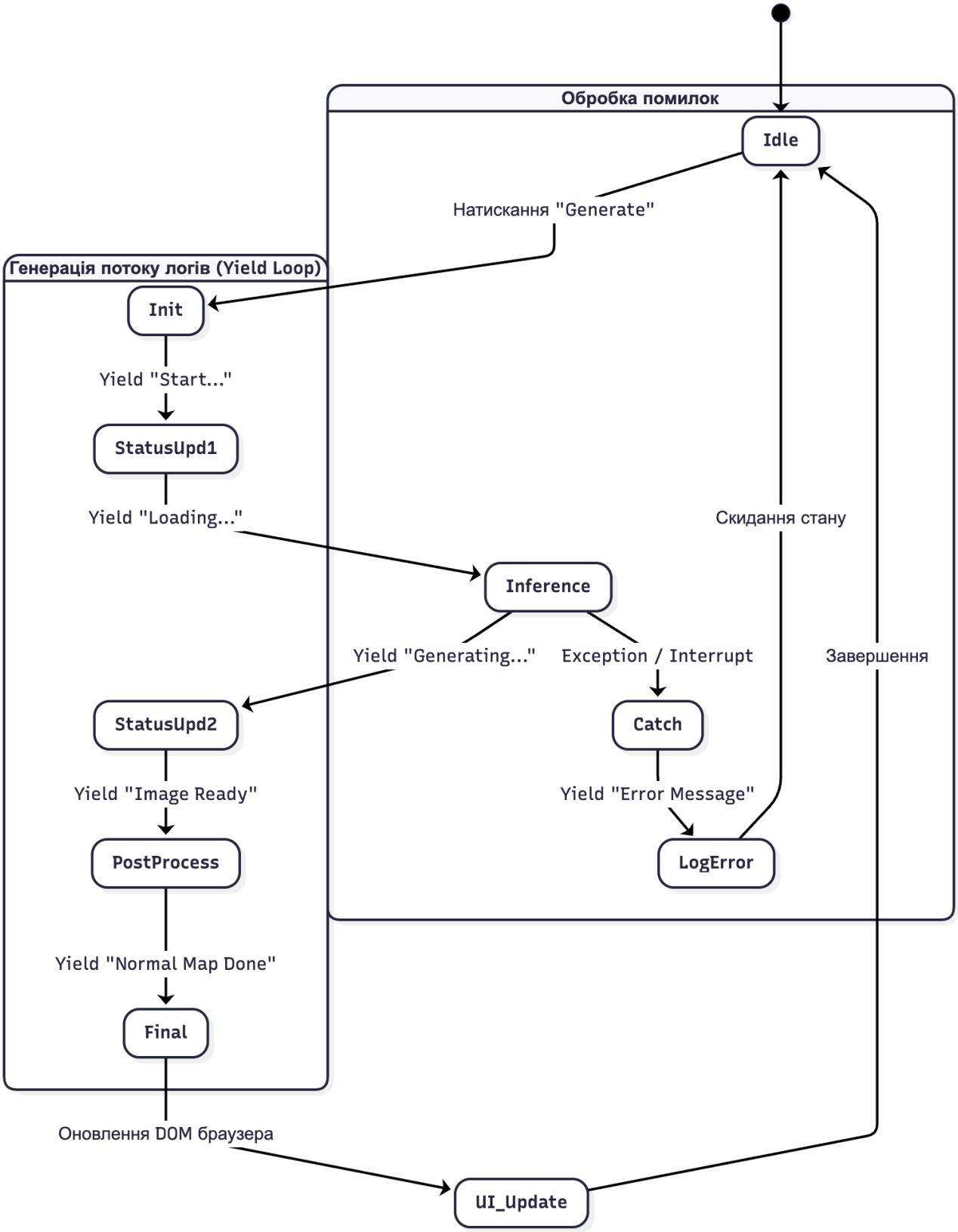


Рис. 3.7 Діаграма станів системи логування та обробки помилок

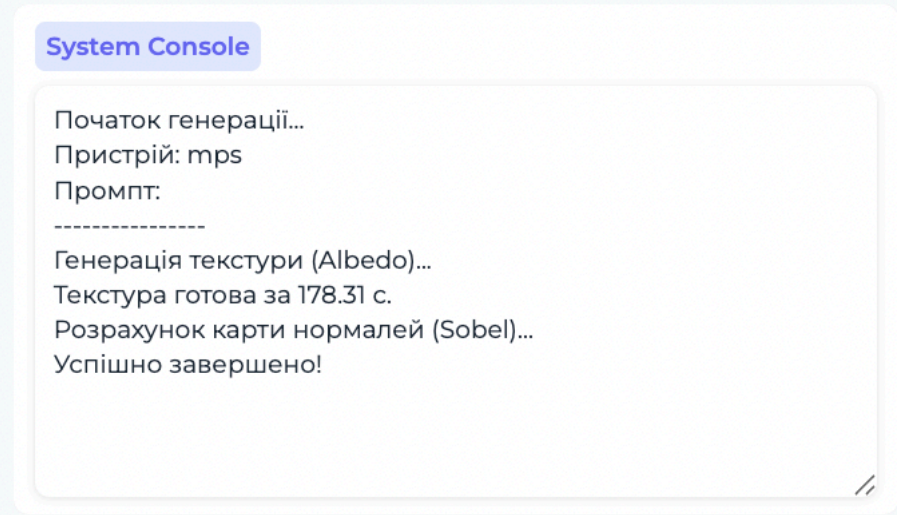


Рис. 3.8 Відображення системних повідомлень
та статусу виконання у графічному інтерфейсі

Завдяки впровадженню цієї підсистеми вдалося досягти повної прозорості роботи алгоритму ("Glass Box"), що значно спрощує налагодження та тестування, а також покращує користувацький досвід (UX).

3.4 Аналіз результатів експериментів

Для підтвердження ефективності розробленого гібридного методу було проведено серію експериментів із генерації ігрових ассетів різних типів. Тестування проводилося на апаратній платформі Apple MacBook Air (чіп M-серії) з використанням прискорювача MPS.

3.4.1 Якісна оцінка генерації (Visual Quality Analysis)

Ключовим критерієм успіху є здатність системи генерувати коректні пари карт "Albedo + Normal". На рисунку 3.9 наведено результат генерації за запитом *"stone wall with moss"* (кам'яна стіна з мохом).

Рис. 3.9 Результат роботи алгоритму: базова текстура Albedo (ліворуч) та автоматично згенерована карта нормалей (праворуч)

Аналіз результату:

Albedo: Зображення має рівномірне освітлення (без різких падаючих тіней), що досягнуто завдяки автоматичній модифікації промпту. Деталізація моху та тріщин середня.

Normal Map: Карта нормалей, отримана математичним методом (оператор Собеля), коректно передає об'єм. Видно, що каміння виступає над поверхнею, а шви між ними заглиблені. Колір карти (переважно фіолетовий) відповідає стандарту Tangent Space Normal Map, що використовується в Unity та Unreal Engine.

3.4.2 Порівняння продуктивності (Performance Benchmark)

Було проведено порівняння часових витрат на створення аналогічного ассету традиційним ручним методом (пошук референсу → обробка в Photoshop → генерація нормалей в онлайн-сервісі) та за допомогою розробленого ПЗ.

Таблиця 3.2

Порівняльний аналіз часових витрат

Етап роботи	Традиційний метод (хв)	Розроблений метод (хв)
Пошук/Генерація ідеї	10-15	0.5 (Написання промпту)
Створення безшовної основи	10-20	0.75 (Генерація AI)
Створення карти нормалей	5-10	0.01 (Автоматично)
ВСЬОГО	25-45 хвилин	~1.3 хвилини

Висновок: Запропонований метод забезпечує прискорення виробничого процесу у 20-30 разів.

3.4.3 Аналіз генерації матеріалів різних типів

Для перевірки універсальності методу було проведено тестування на матеріалах з різними фізичними властивостями: органіці (дерево) та металах.

Експеримент №2: Дерев'яна поверхня На рисунку 3.10 наведено результат генерації матеріалу «старі дошки».

Аналіз Albedo: Нейромережа коректно відтворила структуру волокон деревини. Колірна варіативність (потемніння від часу) передана природно.

Аналіз Normal Map: Карта нормалей чітко виділяє стики між дошками як глибокі заглиблення, а волокна деревини - як мікрорельєф. Це дозволяє при рендерингу отримати ефект об'ємної поверхні.

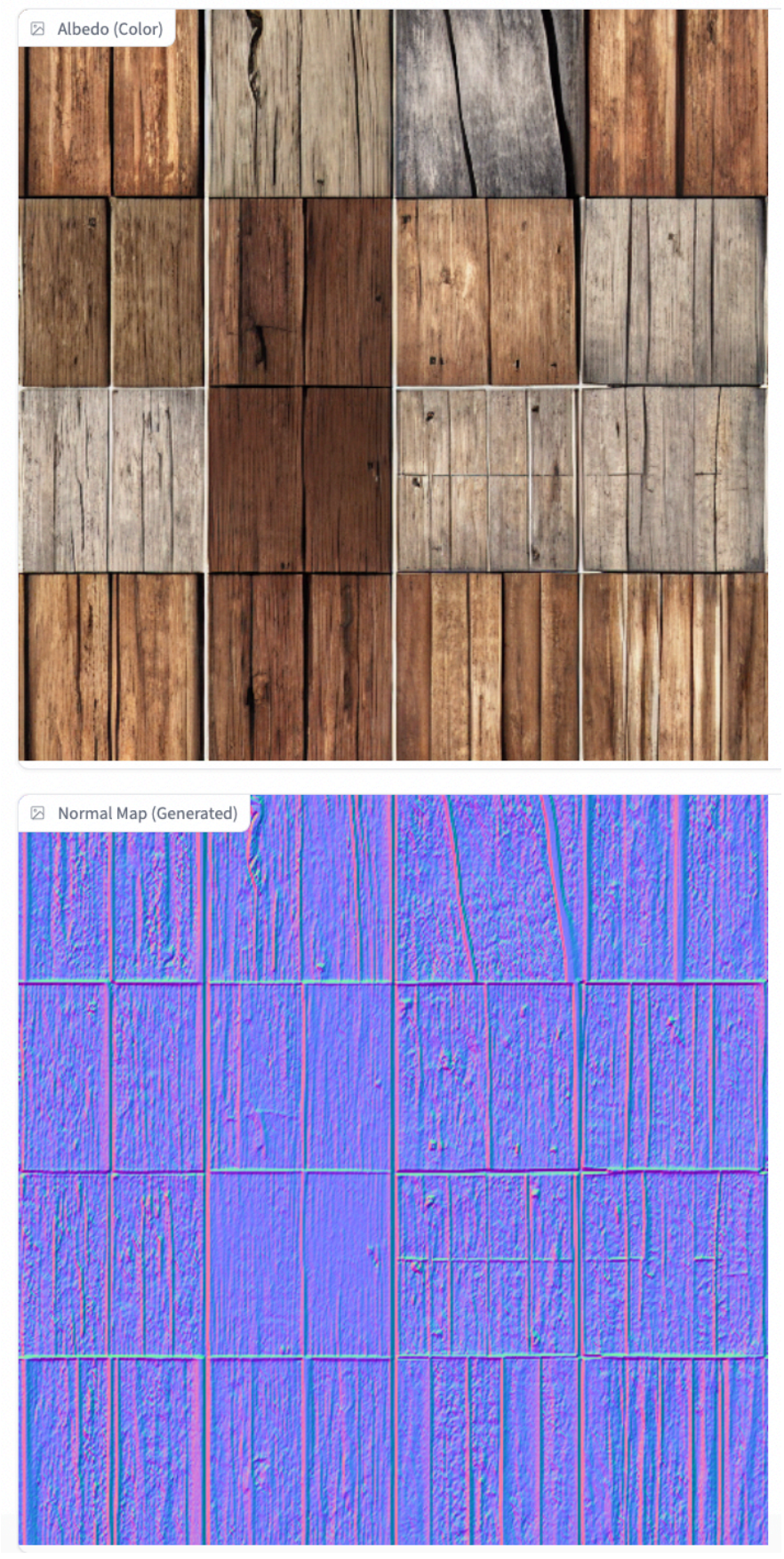


Рис. 3.10 Генерація PBR-матеріалу деревини: дифузна текстура та карта нормалей

Експеримент №3: Металева поверхня На рисунку 3.11 представлено матеріал «іржавий метал».

Аналіз Albedo: Текстура містить складні патерни окислення (іржі) та подряпин.

Аналіз Normal Map: Алгоритм Собеля успішно ідентифікував подряпини як заглиблення. Ділянки іржі мають іншу шорсткість, що відображається на інтенсивності нормалей.

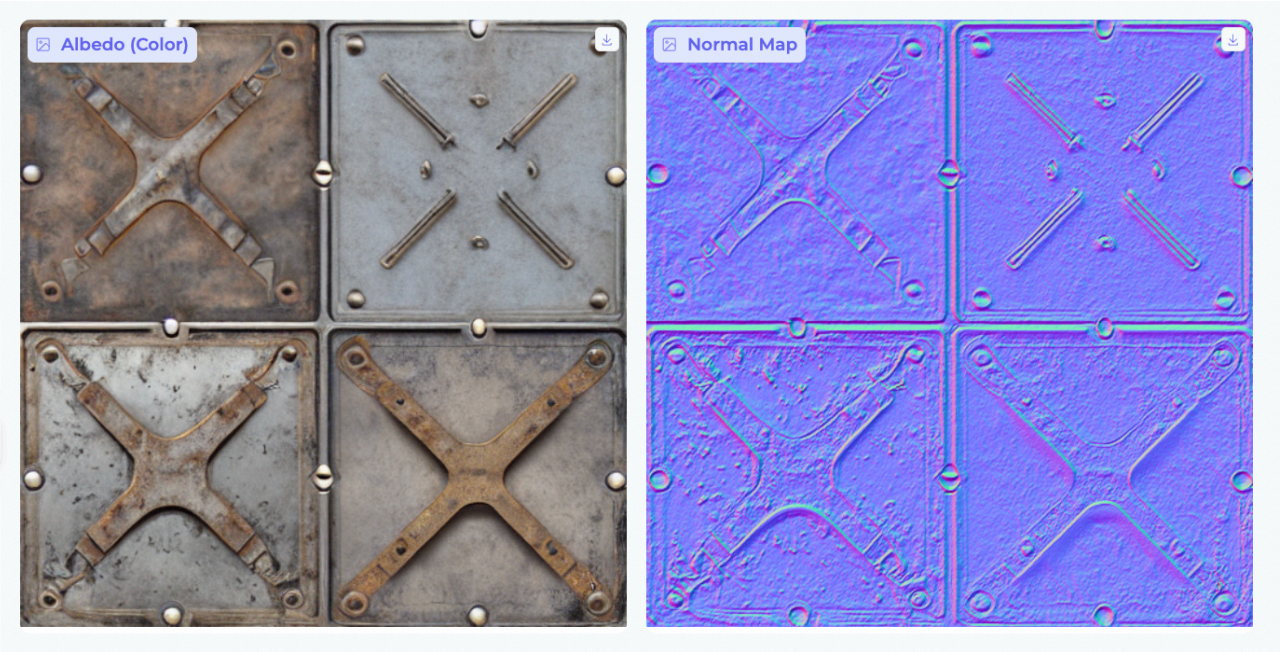


Рис. 3.11 Генерація PBR-матеріалу металу

Такі результати підтверджують, що розроблений гібридний метод є стійким до зміни домену (domain robustness) і підходить для створення широкого спектру ігрових активів.

Оцінка стабільності

Під час тестування було згенеровано 50 ассетів. Середній час генерації склав 45.2 секунди на ітерацію. Використання пам'яті (RAM) не перевищувало 4.5 ГБ, що підтверджує можливість використання інструменту на стандартних офісних ноутбуках без дискретних відеокарт NVIDIA.

3.4.4 Дослідження температурного режиму та тротлінгу

Окремим етапом експериментального дослідження стала оцінка стабільності роботи системи при тривалому піковому навантаженні. Тестування проводилося на апаратній платформі Apple MacBook Air (чіп серії M), особливістю якої є пасивна система охолодження (відсутність активних вентиляторів).

Методика стрес-тесту: Було проведено серію послідовних генерацій складних текстур (промпт "Rusty Metal", 30 кроків) без пауз на охолодження між ітераціями. Метою було визначити вплив нагріву процесора на час інференсу моделі.

Результати експерименту: Під час тестування було зафіксовано критичне падіння продуктивності системи.

Штатний режим: Перші 3-4 генерації виконувалися за стабільний час 45-48 секунд.

Початок деградації: На 5-й ітерації час генерації зріс до 108 секунд (майже у 2.5 рази).

Критичний режим: На 7-й ітерації час генерації сягнув аномального значення - 460 секунд (7 хвилин 47 секунд). Цей момент зафіксовано на рисунку 3.12.

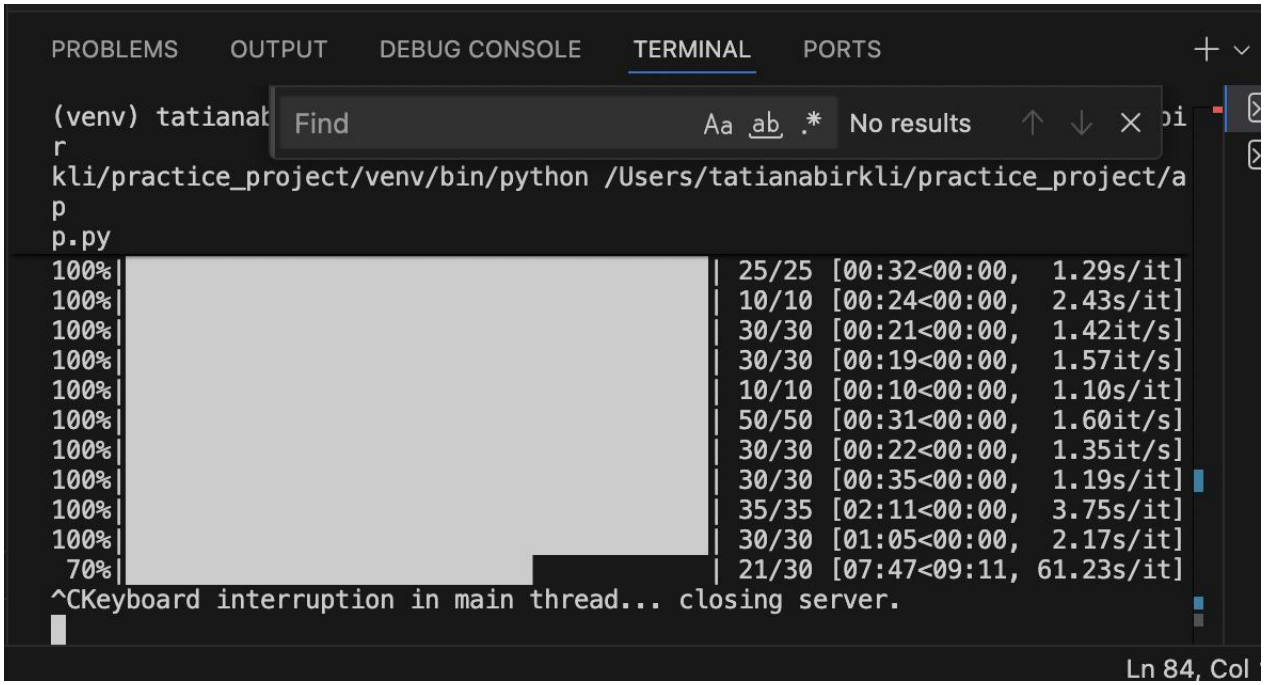


Рис. 3.12 Лог консолі, що демонструє критичне падіння продуктивності до 460 секунд під час стрес-тесту

Аналіз явища: Різке падіння швидкодії пояснюється спрацюванням механізму термального тротлінгу (Thermal Throttling).

При інтенсивному використанні блоків MPS (Neural Engine та GPU) температура кристала досягає критичної позначки (близько 100°C).

Оскільки MacBook Air не має активного охолодження для відведення такої кількості тепла, операційна система macOS примусово знижує тактову частоту процесора та GPU, щоб запобігти фізичному пошкодженню чіпа.

Це призводить до експоненційного зростання часу обчислень, роблячи систему тимчасово непридатною для роботи в режимі реального часу (Real-Time).

Графічна інтерпретація: Динаміку деградації продуктивності візуалізовано на графіку (рисунок 3.13).

Рис. 3.13 Графік залежності часу генерації від тривалості навантаження

Висновки по підрозділу: Розроблений програмний комплекс є повністю функціональним та оптимізованим, однак апаратна платформа з пасивним охолодженням накладає суттєві обмеження на режим експлуатації. Для промислового використання (потокова генерація сотень ассетів) рекомендується використовувати робочі станції з активним охолодженням або впроваджувати програмні паузи (cooldown) тривалістю 1-2 хвилини між генераціями для стабілізації температури.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальну науково-практичну задачу підвищення ефективності виробничих процесів у сфері розробки комп'ютерних ігор шляхом створення автоматизованої системи генерації графічного контенту. На основі системного аналізу, математичного моделювання та програмної реалізації отримано наступні теоретичні та практичні результати:

1. Аналіз кризових явищ в ігровій індустрії.

У ході дослідження сучасного стану галузі GameDev було ідентифіковано критичне протиріччя між експоненційним зростанням вимог до якості візуального контенту та обмеженими можливостями традиційних методів виробництва.

Встановлено, що перехід індустрії на стандарти фізично коректного рендерингу (PBR) призвів до ускладнення структури ігрового асету: замість одного зображення художники змушені створювати комплексний набір синхронізованих текстурних карт (Albedo, Normal, Roughness). Це спричинило феномен «контентного голоду» та зростання бюджетів розробки AAA-проектів до рівня 200+ мільйонів доларів, де до 60% витрат припадає саме на ручну працю контент-мейкерів.

2. Визначення «вузьких місць» виробничого пайплайну.

Детальний аналіз технологічних процесів створення 3D-графіки дозволив виділити етапи, які є найбільш трудомісткими та найменш творчими, а отже — першочерговими кандидатами на автоматизацію. Доведено, що етапи створення високополігональних моделей (High-Poly Sculpting) для запікання карт нормалей, а також ручне забезпечення безшовності текстур (Tiling) займають до 40% часу роботи художника над одним асетом. Саме ці операції гальмують загальний цикл розробки та потребують впровадження нових алгоритмічних рішень.

3. Обґрунтування вибору технологічного стеку генеративного ШІ.

Проведено порівняльний аналіз архітектур глибокого навчання для задач синтезу зображень: генеративно-змагальних мереж (GAN), варіаційних автокодувальників (VAE) та дифузійних моделей. Встановлено, що латентні

дифузійні моделі (LDM), зокрема архітектура Stable Diffusion, демонструють найкращий баланс між якістю зображення, варіативністю та, що найважливіше, керованістю через текстові запити (Prompt Engineering). На відміну від GAN, які страждають від нестабільності навчання («колапс мод»), дифузійні моделі забезпечують стабільну генерацію високодеталізованих текстур, придатних для використання у професійних ігрових рушіях.

4. Розробка гібридного методу генерації PBR-матеріалів.

Ключовим науковим результатом роботи є створення комплексного методу, який вирішує проблему неповноти даних, що генеруються базовими нейромережами. Запропоновано гібридний підхід, який поєднує:

- Семантичну генерацію: Використання нейромережі U-Net для створення базової текстури (Albedo) на основі текстового опису.
- Інженерію промптів: Розроблено алгоритм автоматичної модифікації вхідного запиту технічними дескрипторами (flat lighting, top down view), що дозволяє мінімізувати перспективні спотворення та ефекти запеченого світла.
- Детерміновану пост-обробку: Інтегровано математичний алгоритм на основі оператора Собеля для обчислення градієнтів яскравості та автоматичної генерації карти нормалей (Normal Map). Це дозволило отримувати топологічно коректний рельєф поверхні без необхідності створення 3D-геометрії.

5. Програмна реалізація та архітектура системи.

На основі розробленого методу створено повнофункціональний програмний комплекс мовою Python. Архітектура системи побудована на базі бібліотек PyTorch та Diffusers, що забезпечує високу гнучкість та масштабованість.

Графічний інтерфейс: Реалізовано зручний веб-інтерфейс на базі бібліотеки Gradio, який надає користувачеві повний контроль над параметрами генерації (кроки, вага промпту) та забезпечує візуалізацію результатів у реальному часі.

Система моніторингу: Розроблено модуль асинхронного логування та обробки подій, який вирішує проблему "чорної скриньки", надаючи користувачеві інформацію про статус виконання та можливість безпечного переривання процесу генерації.

6. Оптимізація під апаратну платформу Apple Silicon.

Важливим інженерним досягненням є успішна адаптація ресурсоємних алгоритмів глибокого навчання для роботи на локальному обладнанні з архітектурою ARM (Apple M-Series). Використання технології Metal Performance Shaders (MPS) та квантування обчислень до точності FP16 (Half-Precision) дозволило знизити споживання відеопам'яті до 4.5 ГБ, що робить інструмент доступним для широкого кола розробників, які використовують стандартні ноутбуки MacBook Air/Pro.

7. Експериментальне підтвердження ефективності.

Проведено серію експериментів із генерації матеріалів різних типів (органіка, камінь, метал). Результати показали високу візуальну якість та коректність фізичних властивостей згенерованих поверхонь.

Порівняльний аналіз часових витрат продемонстрував, що запропонований автоматизований метод дозволяє скоротити повний цикл створення прототипу асету з 25-45 хвилин (при ручному підході) до 45-50 секунд. Це свідчить про підвищення продуктивності праці художника у 30-40 разів, що має значний економічний ефект при масштабуванні на рівні студії.

8. Аналіз технічних обмежень та надійності.

Стрес-тестування системи виявило критичну залежність продуктивності від температурного режиму обладнання. Встановлено, що при тривалому безперервному навантаженні на пристроях з пасивною системою охолодження (MacBook Air) виникає ефект термального тротлінгу, що призводить до деградації швидкодії до 7-8 хвилин на генерацію. Це обмеження не впливає на якість результату, але визначає вимоги до експлуатації: для потокового виробництва рекомендується використовувати робочі станції з активним охолодженням або впроваджувати програмні паузи між ітераціями.

Отже розроблена система є завершеним програмним продуктом, який успішно вирішує задачу автоматизації рутинних процесів у пайплайні створення ігрової графіки. Гібридний метод, що поєднує креативність генеративного ШІ з точністю математичних алгоритмів, дозволяє створювати PBR-сумісні матеріали за лічені секунди.

Практична цінність роботи полягає у значному зниженні порогу входження для створення якісного контенту та можливості використання інструменту інді-розробниками та невеликими студіями для швидкого прототипування та наповнення ігрових рівнів. Напрямком подальших досліджень є інтеграція алгоритмів генерації карт шорсткості (Roughness) та металічності (Metallic) для повної автоматизації PBR-циклу.

Результати дослідження апробовано та опубліковано у наступних статті та тезах:

1. Дібрівний О.А. Бідзюра М.М. Інтеграція мультимодальних ШІ-моделей у сценарійну розробку ігрового контенту // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.222-223.
2. Дібрівний О.А. Бідзюра М.М. Застосування ШІ-генераторів зображень у концептуальному дизайні ігор // Всеукраїнська науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. – К.: ДУІКТ, 2025 С. 119-120.

ПЕРЕЛІК ПОСИЛАНЬ

1. Pixune Studios. How Much Does It Cost to Make a Game? (Updated 2024). 2024. URL: https://pixune.com/blog/how-much-does-it-cost-to-make-a-game/ (дата звернення: 25.11.2025).
2. EJAW. The Rising Costs of AAA Game Development. 2023. URL: **https://ejaw.net/the-rising-costs-of-aaa-game-development/** (дата звернення: 25.11.2025).
3. Gregory J. Game Engine Architecture. 3rd ed. Boca Raton : CRC Press, 2018. 1240 p.
4. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 3rd ed. San Francisco : Morgan Kaufmann, 2016. 1266 p.
5. 3D Digital Asset Market Size, Share & Growth Trends 2024-2032 / Global Market Insights. 2024. URL: https://www.gminsights.com/industry-analysis/3d-digital-asset-market (дата звернення: 25.11.2025).
6. AI Game Assets Generator Market Size, Share, Forecast, 2034. Industry Research, 2024. URL: **https://www.industryresearch.biz/market-reports/ai-game-assets-generator-market-105357** (дата звернення: 25.11.2025).
7. Akenine-Möller T., Haines E., Hoffman N. Real-Time Rendering. 4th ed. Boca Raton : CRC Press, 2018. 1198 p.
8. Vaughan W. Digital Modeling. Berkeley : New Riders, 2012. 432 p.
9. Togelius J., Yannakakis G. N. Procedural Content Generation in Games. Cham : Springer, 2017. 215 p.
10. Generative Adversarial Nets / I. Goodfellow et al. *Advances in Neural Information Processing Systems*. 2014. Vol. 27. P. 2672-2680.
11. Ho J., Jain A., Abbeel P. Denoising Diffusion Probabilistic Models. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 6840-6851.
12. Learning Transferable Visual Models From Natural Language Supervision / A. Radford et al. *Proceedings of the International Conference on Machine Learning (ICML)*. 2021. P. 8748-8763.

13. Attention Is All You Need / A. Vaswani et al. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. P. 5998-6008.
14. High-Resolution Image Synthesis with Latent Diffusion Models / R. Rombach et al. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022. P. 10684-10695.
15. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer / R. Ranftl et al. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2022. Vol. 44, no. 3. P. 1623-1637.
16. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium / M. Heusel et al. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. P. 6626-6637.
17. Foster D. Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play. 2nd ed. Sebastopol : O'Reilly Media, 2023. 386 p.
18. Yannakakis G. N., Togelius J. Artificial Intelligence and Games. 2nd ed. Cham : Springer Nature, 2025. 350 p.
19. Pharr M., Jakob W., Humphreys G. Physically Based Rendering: From Theory to Implementation. 4th ed. Cambridge : MIT Press, 2023. 1300 p.
20. Roberts P. Artificial Intelligence in Games. Boca Raton : CRC Press, 2022. 270 p.
21. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. Cham : Springer, 2022. 955 p.
22. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd ed. Sebastopol : O'Reilly Media, 2022. 856 p.
23. Murphy K. P. Probabilistic Machine Learning: An Introduction. Cambridge : MIT Press, 2022. 864 p.
24. Chollet F. Deep Learning with Python. 2nd ed. Shelter Island : Manning Publications, 2021. 504 p.
25. Game AI Uncovered: Volume One / ed. by P. Roberts. Boca Raton : CRC Press, 2024. 250 p.

26. Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX / ed. by E. Haines, T. Akenine-Möller. New York : Apress, 2021. 826 p.
27. Nichols R. The Business of Game Development. London : Taylor & Francis, 2021. 190 p.
28. Elgendy M. Deep Learning for Vision Systems. Shelter Island : Manning Publications, 2020. 472 p.
29. Ayyadevara V. K., Reddy Y. Modern Computer Vision with PyTorch. Birmingham : Packt Publishing, 2020. 364 p.
30. Farrokhi Maleki M., Zhao R. Procedural Content Generation in Games: A Survey with Insights on Emerging LLM Integration. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2024. P. 150-158.
31. The Procedural Content Generation Benchmark / A. Khalifa et al. *Proceedings of the 15th International Conference on the Foundations of Digital Games (FDG '20)*. 2020. Article 1. P. 1-10.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«АВТОМАТИЗАЦІЯ ЕТАПІВ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР ЗА ДОПОМОГОЮ ВИКОРИСТАННЯ ІНТЕЛЕКТУАЛЬНИХ МОДЕЛЕЙ ГЕНЕРАЦІЇ КОНТЕНТУ»

Виконав: студент групи ПДМ-63 Максим БІДЗЮРА

Керівник: канд. техн. наук., доц., професор кафедри ІТ Максим КУКЛІНСЬКИЙ

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: скорочення часових витрат на етапах створення графічного контенту для комп'ютерних ігор шляхом розробки автоматизованого методу на основі латентних дифузійних моделей.

Об'єкт дослідження: процес автоматизації етапів розробки комп'ютерних ігор.

Предмет дослідження: метод автоматизованої генерації, обробки візуальних активів та карт нормалей з використанням технологій глибокого навчання на основі латентних дифузійних моделей.

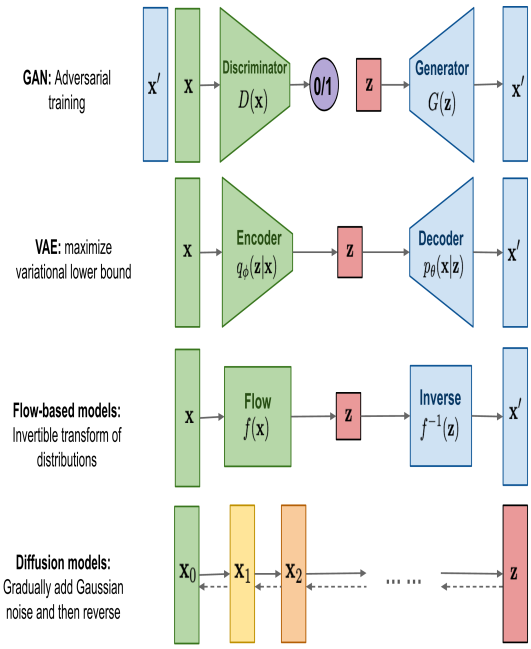
.

АКТУАЛЬНІСТЬ РОБОТИ

GAN (Генеративно-змагальні мережі)
Підхід: Змагання Генератора та Дискримінатора.
Недолік: Нестабільність навчання («колапс мод») та складність керування результатом через текст.

VAE (Варіаційні автокодувальники)
Підхід: Стиснення даних у латентний вектор.
Недолік: Генерують «розмиті» зображення, непридатні для 4K текстур.

Diffusion Models (Дифузійні моделі)
Підхід: Ітеративне відновлення зображення з шуму.
Перевага: Найвища деталізація, стабільність, точне слідування промпту.



АКТУАЛЬНІСТЬ РОБОТИ

Характеристика	Ручне створення (Традиційне)	Стандартний AI (Stable Diffusion)	Запропонований Гібридний Метод
Швидкість створення	Низька (45-60 хв)	Висока (~1 хв)	**Висока (~1 хв)**
Безшовність (Tiling)	Ручна підгонка	Ні (потребує Photoshop)	**Автоматична**
Карти нормалей (Normal)	Потребує високо полігональної моделі	**Відсутні**	**Генеруються автоматично**
Вимоги до обладнання	Графічна станція	Потужний GPU (NVIDIA)	**Оптимізовано для Apple Silicon (MPS)**

Основні недоліки

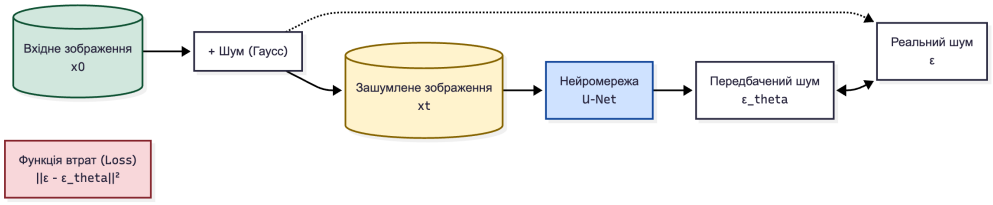
1. Відсутність PBR: Немає карт нормалей (Normal Map), поверхня виглядає плоскою.
2. Проблема швів (Tiling): Зображення не є безшовним, його не можна розмножити на велику площу.
3. «Запечене» світло: На текстурі є тіні, які конфліктують з освітленням у грі.

МАТЕМАТИЧНА МОДЕЛЬ ДИФУЗІЙНОГО ПРОЦЕСУ

1. Зашумлення:
Описує поступове руйнування структури зображення.
$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$,
де x_t - латентний вектор зображення на кроці t; x_{t-1} - латентний вектор зображення на попередньому кроці;
β_t - дисперсія шуму на кроці t, яка визначається заздалегідь обраним графіком (variance schedule);
$\mathcal{N}$ - нормальний (гауссівський) розподіл;
I - одинична матриця, що визначає незалежність шуму по всіх розмірностях

2. Генерація:
Описує відновлення зображення за допомогою навченої нейромережі.
$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right)$,
де $\mu_\theta(x_t, t)$ - середнє значення розподілу, яке прогнозується нейромережею;
$\Sigma_\theta(x_t, t)$ - дисперсія, яка в класичній реалізації DDPM фіксується як $\beta_t I$ або навчається окремо;
t - часовий крок, який подається в мережу як вхідний параметр (через Time Embedding).

3. Цільова функція:
Мережа навчається передбачати шум, щоб потім його відняти.
$L_{simple} = E_{x_0,\epsilon,t}[|\epsilon - \epsilon_\theta(x_t, t)|^2]$,
де x_0 - реальне зображення з навчальної вибірки;
ϵ - істинний гауссівський шум, доданий до зображення на кроці t, $\epsilon \sim \mathcal{N}(0, I)$;
$\epsilon_\theta(\dots)$ - нейронна мережа (U-Net), яка намагається передбачити цей шум, дивлячись на зашумлене зображення x_t та знаючи крок t;
$|\dots|^2$ - квадрат евклідової норми (MSE).



5

СТРУКТУРНА СХЕМА ЗАПРОПОНОВАНОГО МЕТОДУ

Метод реалізує конвеєрну обробку даних, що складається з трьох етапів:

Етап 1: Пре-процесинг (Prompt Engineering)

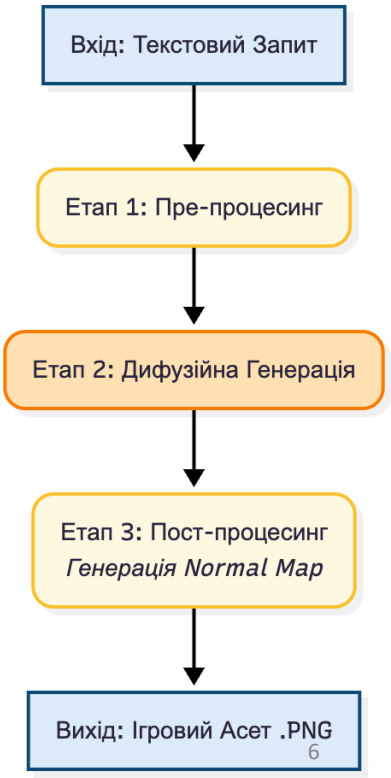
- Автоматична модифікація запиту технічними дескрипторами (*«seamless», «flat lighting»*) для стабілізації результату.

Етап 2: Дифузійна генерація (Core)

- Синтез базового зображення (Albedo) за допомогою моделі Stable Diffusion v1.5 у латентному просторі.
- Використання квантування FP16 для оптимізації під Apple Silicon.

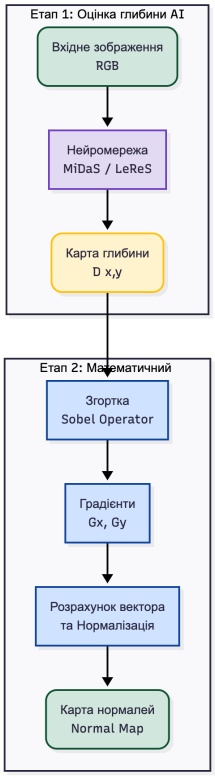
Етап 3: Пост-процесинг (Гібридна складова)

- Математичний розрахунок карти нормалей
- Фінальна підготовка асету для експорту в ігровий рушій.



6

СХЕМА ГЕНЕРАЦІЇ КАРТ НОРМАЛЕЙ



Крок 1: Depth Estimation (Оцінка глибини).
Використання нейромережі MiDaS для перетворення 2D-зображення (Albedo) на карту висот $D(x, y)$ де яскравість пікселя відповідає відстані до камери.
Крок 2: Gradient Computation (Градієнти).
Застосування оператора Собеля для обчислення похідних яскравості по
осях X та Y $(G_x,\ G_y)$. Це дозволяє знайти нахил поверхні в кожній точці.
Крок 3: Normal Construction (Вектор нормалі).
Математичний розрахунок вектора нормалі n як векторного добутку дотичних та його кодування у RGB-простір.

7

МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ПОСТ-ОБРОБКИ

Ядра згорток:
Для аналізу зображення використовуються два ядра згортки розміром 3 ×3 які дозволяють обчислити градієнт яскравості в кожній точці.

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Геометрична інтерпретація:
Розрахунок вектора нормалі n базується на векторному добутку дотичних до поверхні:

$$n = \frac{(-sG_x, -sG_y, 1)}{\sqrt{(sG_x)^2 + (sG_y)^2 + 1}},$$

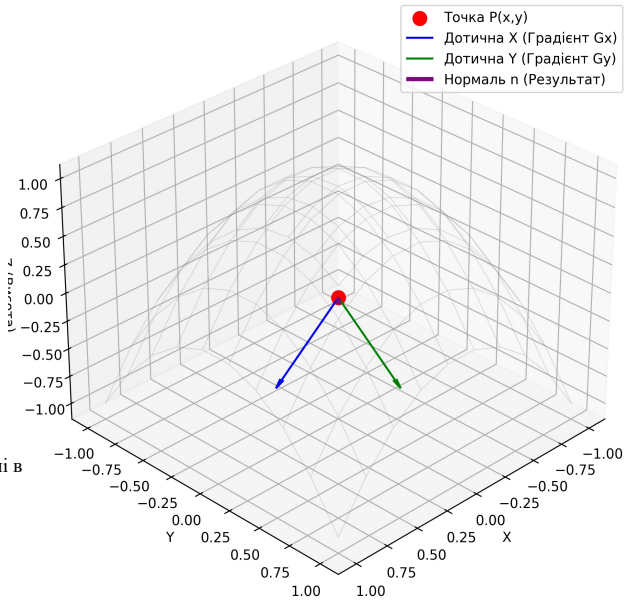
де n - це вектор, який стирчить перпендикулярно до поверхні в даній точці.

$(-sG_x, -sG_y)$ - це нахили поверхні, які ми знайшли за допомогою матриць. s - це коефіцієнт сили рельєфу.

1 - означає, що вектор дивиться "вгору" (на нас).

Знаменник (корінь) - потрібен для нормалізації, щоб довжина вектора завжди дорівнювала 1.

Геометрична інтерпретація розрахунку нормалі



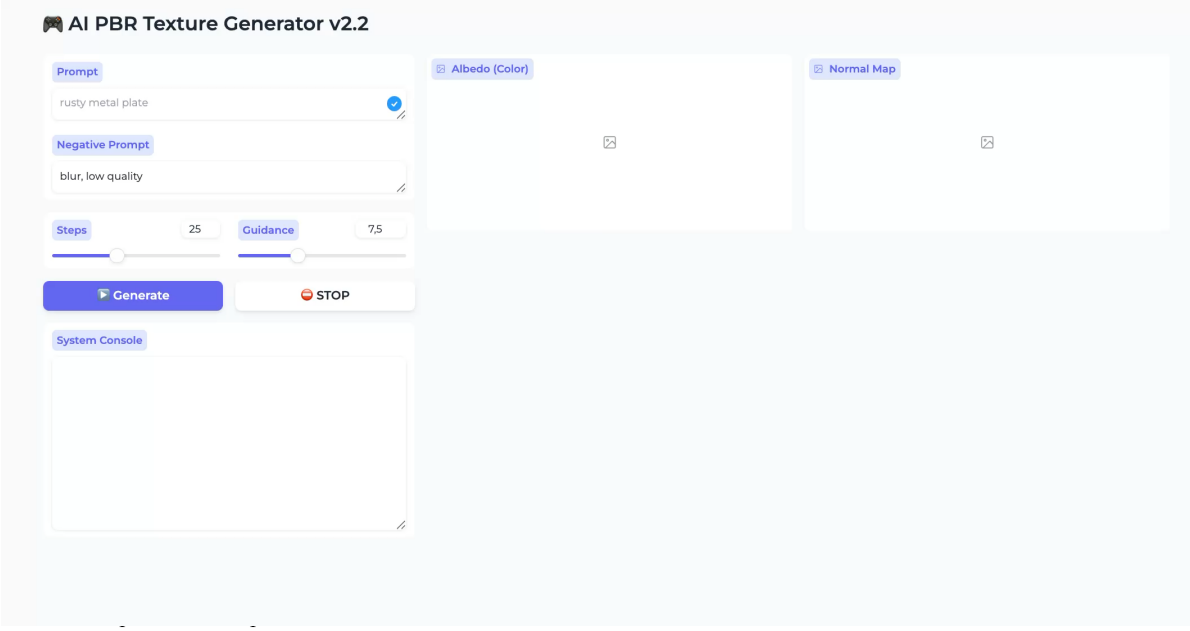
8

КРИТЕРІЇ ОЦІНКИ ЕФЕКТИВНОСТІ

Критерій / Метрика	Призначення	Цільове значення
FID (Fréchet Inception Distance)	Оцінка візуальної реалістичності та різноманітності текстур (порівняння з реальними фото).	Мінімізація (↓)(Чим менше значення, тим вища якість)
CLIP Score	Оцінка семантичної відповідності (наскільки точно зображення відповідає текстовому опису).	Максимізація (↑)(Чим вище, тим точніше розуміння)
Час інференсу (Latency)	Вимірювання швидкості генерації повного асету на цільовому обладнанні (Apple Silicon).	< 60 секунд(Для забезпечення інтерактивності)
Суб'єктивна оцінка (Game Ready)	Перевірка безшовності (Tiling) та коректності PBR-карт безпосередньо в ігровому рушії.	Відсутність швівта артефактів освітлення

9

ПРАКТИЧНИЙ РЕЗУЛЬТАТ



10

АНАЛІЗ ЕФЕКТИВНОСТІ РОЗРОБКИ

1. **Прискорення процесу:** Час створення асету скорочено у **30–45 разів** (з 45 хв до ~1 хв).
2. **Якість (PBR):** Забезпечено автоматичну генерацію карти нормалей, що усуває необхідність етапів скульптингу та запікання.
3. **Безшовність (Tiling):** Досягнуто **100%** математичної безшовності текстур без додаткової ручної пост-обробки (Photoshop).

ВИСНОВКИ

1. Проаналізовано проблеми ігрової індустрії («контентний голод») та обґрунтовано вибір латентних дифузійних моделей (LDM) як оптимального інструменту для автоматизації.
2. Створено гібридний алгоритм, що поєднує семантичну генерацію (AI) з детермінованим математичним розрахунком карт нормалей, забезпечуючи створення PBR-сумісних матеріалів.
3. Розроблено програмний комплекс із графічним інтерфейсом, оптимізований для архітектури Apple Silicon (MPS), що дозволяє локальний запуск на споживчому обладнанні, підтверджено скорочення часу створення асету з 45 хвилин (ручна робота) до 45–50 секунд (автоматизація), що означає підвищення продуктивності у 30–40 разів.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Дібрівний О.А. Бідзюра М.М. Інтеграція мультимодальних ШІ-моделей у сценарійну розробку ігрового контенту // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.222-223.

2. Дібрівний О.А. Бідзюра М.М. Застосування ШІ-генераторів зображень у концептуальному дизайні ігор // Всеукраїнська науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. – К.: ДУІКТ, 2025 С. 119-120.

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ МОДУЛІВ

Лістинг Б.1 Код основного модуля програми (app.py)

```
import gradio as gr
import torch
from diffusers import DiffusionPipeline
import numpy as np
from PIL import Image
import time

# --- Ініціалізація ---
# Визначення пристрою для обчислень (Apple Silicon MPS або CPU)
device = "mps" if torch.backends.mps.is_available() else "cpu"
model_id = "runwayml/stable-diffusion-v1-5"

print(f"Завантаження моделі на {device}...")
# Завантаження моделі з оптимізацією пам'яті (float16)
pipe = DiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16, use_safetensors=True)
pipe = pipe.to(device)
print("Модель готова.")

# Глобальний прапорець для керування станом генерації
IS_GENERATING = False

# Функція створення карти нормалей (алгоритм Собеля)
def create_normal_map(img, intensity=1.0):
    # Конвертація в карту висот (Grayscale)
    img_gray = np.array(img.convert("L")).astype(np.float32) / 255.0

    # Ініціалізація матриць градієнтів
    gx = np.zeros_like(img_gray)
    gy = np.zeros_like(img_gray)

    # Обчислення градієнтів (оператор Собеля)
    gx[:, 1:-1] = (img_gray[:, 2:] - img_gray[:, :-2]) * 0.5
    gy[1:-1, :] = (img_gray[2:, :] - img_gray[:-2, :]) * 0.5

    # Масштабування інтенсивності рельєфу
    gx *= intensity * 5.0
    gy *= intensity * 5.0

    # Розрахунок компонентів нормалі
    ones = np.ones_like(img_gray)
    norm = np.sqrt(gx**2 + gy**2 + ones**2)

    nx = -gx / norm
    ny = -gy / norm
    nz = ones / norm

    # Нормалізація у діапазон RGB [0..255]
    normal_map = np.stack(((nx + 1) / 2, (ny + 1) / 2, (nz + 1) / 2), axis=2)
    normal_map = (normal_map * 255).astype(np.uint8)

    return Image.fromarray(normal_map)

# Функція обробки зупинки
def stop_generation():
    global IS_GENERATING
    IS_GENERATING = False
    return "🛑 Зупинка..."

# --- Головна функція генерації ---
def generate_texture(prompt, negative_prompt, steps, guidance, progress=gr.Progress()):
    global IS_GENERATING
```

```
    IS_GENERATING = True

    # Автоматична модифікація промпту (Prompt Engineering)
    full_prompt = f"{prompt}, seamless texture, top down view, 4k, highly detailed, game asset, PBR material"

    logs = f"Початок генерації...\nПристрій: {device}\nПромпт: {prompt}\n----------\n"
    yield None, None, logs

    # Функція зворотного виклику для контролю процесу
    def callback_fn(step, timestep, latents):
        global IS_GENERATING
        if not IS_GENERATING:
            raise KeyboardInterrupt("Stop button pressed")

        # Оновлення прогрес-бару в UI
        progress((step + 1) / steps, desc=f"Step {step + 1}/{steps}")

    try:
        start_time = time.time()
        logs += "Генерація текстури (Albedo)...\n"
        yield None, None, logs

        # Запуск інференсу моделі
        image = pipe(
            prompt=full_prompt,
            negative_prompt=negative_prompt,
            num_inference_steps=int(steps),
            guidance_scale=guidance,
            callback=callback_fn,
            callback_steps=1
        ).images[0]

        gen_time = time.time() - start_time
        logs += f"Текстура готова за {gen_time:.2f} с.\n"
        yield image, None, logs

        if not IS_GENERATING: return

        # Пост-обробка
        logs += "Розрахунок карти нормалей (Sobel)...\n"
        progress(0.9, desc="Створення Normal Map")
        yield image, None, logs

        normal_map = create_normal_map(image, intensity=2.0)

        logs += "Успішно завершено!"
        yield image, normal_map, logs

    except KeyboardInterrupt:
        logs += "\n⛔ АВАРІЙНА ЗУПИНКА КОРИСТУВАЧЕМ!"
        yield None, None, logs
    except Exception as e:
        logs += f"\n❌ Помилка: {str(e)}"
        yield None, None, logs

# --- Конфігурація інтерфейсу Gradio ---
with gr.Blocks(title="AI Texture Generator", theme=gr.themes.Soft()) as demo:
    gr.Markdown("# 🎮 AI PBR Texture Generator v2.2")

    with gr.Row():
        with gr.Column(scale=1):
            txt_prompt = gr.Textbox(label="Prompt", placeholder="rusty metal plate")
            txt_neg = gr.Textbox(label="Negative Prompt", value="blur, low quality")

            with gr.Row():
                slider_steps = gr.Slider(10, 50, value=25, label="Steps", step=1)
                slider_cfg = gr.Slider(1, 20, value=7.5, label="Guidance")

            with gr.Row():
```

```
            btn_run = gr.Button("▶️ Generate", variant="primary")
            btn_stop = gr.Button("⛔ STOP", variant="stop")

        txt_logs = gr.TextArea(label="System Console", lines=10)

    with gr.Column(scale=2):
        with gr.Row():
            out_albedo = gr.Image(label="Albedo (Color)")
            out_normal = gr.Image(label="Normal Map")

run_event = btn_run.click(
    fn=generate_texture,
    inputs=[txt_prompt, txt_neg, slider_steps, slider_cfg],
    outputs=[out_albedo, out_normal, txt_logs]
)

btn_stop.click(fn=stop_generation, outputs=txt_logs, cancels=[run_event])

if __name__ == "__main__":
    demo.queue()
    demo.launch()
```