Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Генерація PBR-текстур на основі зображень у 3D-графіці методами машинного навчання
(тема)

Виконав:
здобувач другого року навчання,
групи ДСм-24-1
Ніколайчук Д. А.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Науки про дані (Data Science)
(повна назва спеціалізації)
Керівник доц. Вітько О. В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Лариса ЧАЛА
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Науки про дані (Data Science)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Ніколайчуку Денису Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Генерація PBR-текстур на основі зображень у 3D-графіці методами машинного навчання

затверджена наказом університету від 24 листопада 20 25 р. № 1057Ст

2. Термін подання студентом роботи до екзаменаційної комісії 10 грудня 20 25 р.

3. Вихідні дані до роботи Науково-технічні публікації, дані інтернет-джерел та відомих наукових проєктів щодо розробки моделей машинного навчання з використанням нейронних мереж для генерації зображень, а також застосунків, що використовують моделі машинного навчання, Python documentation, TensorFlow documentation

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Проектування та вибір архітектури аддону
3) Вибір технологій та засобів розробки
4) Опис програмної реалізації

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	24.11.2025	виконано
2	Аналіз предметної галузі	24.11.2025	виконано
3	Аналіз існуючих аналогів	24.11.2025 – 25.11.2025	виконано
4	Вибір архітектури аддону	25.11.2025 – 26.11.2025	виконано
5	Підготовка набору навчальних та тестових даних	26.11.2025	виконано
6	Аналіз зібраних даних	26.11.2025	виконано
7	Створення способу роботи з зібраними даними	27.11.2025 – 28.11.2025	виконано
8	Проєктування моделі нейронної мережі	29.11.2025 – 30.11.2025	виконано
9	Навчання спроєктованої моделі	1.12.2025	виконано
10	Збереження навченої моделі	1.12.2025	виконано
11	Розробка аддону	2.12.2025 – 6.12.2025	виконано
12	Тестування розробленого аддону	06.12.2025	виконано
13	Написання пояснювальної записки	06.12.2025 – 08.12.2025	виконано
14	Перевірка на академічний плагіат	10.12.2025	виконано
15	Нормоконтроль	11.12.2025	виконано
16	Попередній захист	16.12.2025	виконано
17	Захист перед ЕК	17.12.2025	

Дата видачі завдання 24 листопада 20 25 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Вітько О. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 93 с., 29 рис., 2 табл., 1 дод., 22 джерела.

БІБЛІОТЕКИ, ГЕНЕРАЦІЯ ЗОБРАЖЕНЬ, МАШИННЕ НАВЧАННЯ, МИСТЕЦТВО, ТЕКСТУРУВАННЯ, 3D-ГРАФІКА.

Об’єкт дослідження – методи генерації PBR-текстур з одного вхідного зображення в 3D-графіці.

Мета роботи – закріплення, поглиблення теоретичних та практичних знань, здобутих у процесі навчання, а також дослідження та практична реалізація сучасних методів машинного навчання (ML) для автоматизованого синтезу фізично коректних текстур (зокрема, карт нормалей, шорсткості та металічності).

Методи дослідження – в рамках виконання кваліфікаційної роботи було проведено теоретичний аналіз, що містить дослідження літератури та огляд технологій, а також емпіричний аналіз під час якого було зібрано дані, проведено необхідні експерименти та їх аналіз

Темою кваліфікаційної роботи є автоматизована генерація набору PBR-текстур (normal, roughness, displacement та ін.) з використанням бібліотек машинного навчання, зокрема TensorFlow/Keras та мови Python. У результаті виконання було проведено дослідження як теоретичних основ PBR-пайплайну, так і практичної реалізації моделі та її інтеграції.

Розглядався загальний підхід до задачі перетворення зображення на зображення, що дозволяє нейронній мережі виводити приховані матеріальні властивості поверхні з візуальних підказок на одному фото. Була розроблена та натренована модель ML, а також створено аддон для Blender 3D для демонстрації практичного застосування.

ABSTRACT

Master’s thesis contains: 93 pp., 29 fig., 2 tabl., 1 ann., 22 references.

IMAGE CLASSIFICATION, LIBRARIES, MACHINE LEARNING, MODELS, NEURAL NETWORKS, TENSORFLOW.

The object of the study is the methods for generating PBR textures from a single input image in 3D graphics.

The purpose of the work is to consolidate and deepen the theoretical and practical knowledge acquired during the study process, as well as to research and practically implement modern machine learning (ML) methods for the automated synthesis of physically based textures (specifically normal, roughness, and metalness maps).

Research methods – within the framework of the qualification paper, a theoretical analysis was conducted, which included a literature review and a technology overview, as well as an empirical analysis involving data collection, the execution of necessary experiments, and their subsequent analysis.

The subject of the qualification paper is the automated generation of a set of PBR textures (Normal, Roughness, Displacement, etc.) using machine learning libraries, specifically TensorFlow/Keras, and the Python programming language. As a result, a study was conducted on both the theoretical foundations of the PBR pipeline and the practical implementation of the model and its integration.

The general approach to the Image to Image Translation task was examined, which allows the neural network to infer latent material surface properties from visual cues in a single photograph. An ML model was developed and trained, and a Blender 3D addon was created to demonstrate the practical application.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 8
Вступ 9
1 Аналіз предметної галузі 11
1.1 Опис предметної галузі 11
1.2 Аналіз існуючих аналогічних систем 15
1.2.1 Застосунок для створення PBR текстур – Substance Sampler 16
1.2.2 Застосунок для створення PBR текстур – Materialize 17
1.2.3 Застосунок для створення PBR текстур – CrazyBump 19
1.2.4 Застосунок для створення PBR текстур – PixPlant 20
1.3 Постановка задачі 23
2 Вибір моделей та методів вирішення задачі генерації текстур 24
2.1 Аналіз цифрових зображень 24
2.2 Попередня обробка зображень у системах комп'ютерного зору 26
2.3 Обробка зображень високої роздільної здатності 27
2.4 Аналіз штучних нейромереж 29
2.5 Вибір архітектури нейронної мережі 31
2.6 Метрики оцінки навчання 34
3 Проектування та вибір архітектури аддону 37
3.1 Функціональні вимоги до аддону 37
3.2 Вибір архітектури аддону 38
3.3 Проєктування архітектури аддону 43
3.4 Підсумок кінцевої архітектури аддону 45
4 Вибір технологій та засобів розробки 47
4.1 Мова програмування Python 49
4.2 Бібліотека Tensor Flow 50
4.3 Сервіс Google Colab 52
4.4 Інструменти Blender API 54
4.5 Інструменти доповнення NumPy 55

4.6 Застосунок для роботи з 3D графікою – Blender 3D 56
4.7 Хмарне сховище Google Drive 58
5 Програмна реалізація та експериментальні дослідження 61
5.1 Вибір набору даних 61
5.2 Тренування, збереження і перевірка моделей машинного навчання 62
5.3 Створення аддону для Blender 74
5.3.1 Ініціалізація середовища та імпорт залежностей 77
5.3.2 Логіка обробки даних 78
5.3.2 Інтеграція з Blender API 79
5.4 Огляд роботи застосунку 80
Висновки 83
Перелік джерел посилання 85
Додаток А Приклади програмної реалізації основного функціоналу 88
Додаток Б Відомість кваліфікаційної роботи 93

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПГ – предметна галузь;

ШІ – штучний інтелект;

AI – Artificial Intelligence – штучний інтелект;

API – Application Programming Interface – інтерфейс програмування додатків;

CNNs – Convolutional Neural Networks – згорткові нейронні мережі;

CPU – Central Processing Unit – центральний процесор;

DNNs – Deep Neural Networks – глибинні нейронні мережі;

GPU – Graphics Processing Unit – графічний процесор;

NLP – Natural Language Processing – обробка природної мови;

ONNX – Open Neural Network Exchange – відкритий обмін нейронними мережами, формат збереження моделей штучного інтелекту;

PBR – Phisically Based Rendering – рендеринг оснований на фізиці;

ReLU – Rectified Linear Unit – випрямлена лінійна функція;

RGB – Red Green Blue – кольорова модель в якій кольори формуються шляхом змішування червоного зеленого та синього в різних пропорціях.

ВСТУП

Одним із найцікавіших творчих напрямів комп'ютерних наук є комп'ютерна графіка. Існує велика кількість розгалужень даної дисципліни таких як векторна чи растрова графіка, проте, на мою думку одним з найбільш цікавих є – 3D-графіка. Імітація тривимірного простору на основі фізичної бази завжди була важливим етапом розвитку комп'ютерних технологій та значно покращила можливості людини творити. Дана сфера дотична як до технічних спеціальностей таких як інженерія чи медицина, так і креативної сфери, від комерції та маркетингу до кінематографа.

В даній роботі основну увагу буде зосереджено саме на креативному напрямку. В свій час кінострічка «Трон» стала родоначальником 3D-графіки основаної на полігональному моделюванні. Дана стрічка була проривною на свій час та дарувала глядачам незабутній досвід. Від цього моменту з розвитком комп'ютерного заліза та можливостей, візуальні ефекти створені на основі 3D-графіки стрімко входять в конвеєр більшості студій. А сама сфера стає однією з найбільших та найбажаніших галузей індустрії [1].

Паралельно кінематографу також розвивається індустрія виробництва 3D-відеоігор. Основана на тих же принципах, графіка в відеоіграх стає таким же проривом в комп'ютерній сфері, даруючи гравцям нові враження.

До сьогоднішнього дня, розвиток технологій створення даного виду 3D-графіки є стрімким та не зупиняється ні на мить. Такі застосунки як Blender 3D, дали можливість великій кількості людей увійти у дану сферу. Широкий інструментарій та активне використання застосунку великими студіями, а також факт того що даний застосунок є безплатним та має відкритий початковий код. Як результат популярність сфери 3D-графіки сильно виросла за останній час, як і попит на сучасні інструменти її створення.

Паралельно комп'ютерній графіці стрімко розвивається і інший напрямок комп'ютерних наук – машинне навчання. З ростом

обчислювальних потужностей – алгоритми основані на штучному інтелекті почали демонструвати все більш виняткові результати. Особливо цікавими є різні генеративні нейронні мережі, що можуть створювати як зображення, так і 3D-моделі [2]. Проте навіть попри стрімкий розвиток моделей генераторів, ручна робота художника все ще залишається цінною та більш якісною тому слід розглядати дані технології не як заміну, а як допоміжний інструмент, що буде виконувати рутинні завдання.

В ході виконання кваліфікаційної роботи було вирішено створити саме такий допоміжний інструмент, що використовує переваги алгоритмів машинного навчання, задля полегшення життя художника, а не для його заміни. Суть розроблюваного інструменту полягає в спрощенні процесу фотограмметрії текстур за допомогою генеративних нейронних мереж. Буде створено розширення до програмного застосунку Blender 3D, що з вхідного зображення – карти кольору, генеруватиме текстури PBR пайплайну, а саме карту шорсткості (roughness map) та карту нормалей (normal map). Даний застосунок значно полегшить створення власних високоякісних текстур задля використання їх в 3D-графіці.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис предметної галузі

Індустрія 3D-графіки набула неабиякої популярності з початку 2000-х років. Якщо на початку це була професія неймовірно важка та дорога в опануванні, то зараз будь-хто з більш менш потужним комп’ютером може завантажити Blender 3D та почати створювати тривимірне мистецтво. Від твердотільного моделювання для ігор та фільмів, до симуляцій та моушен-графіки.

Разом з популяризацією 3D-графіки, розвиток алгоритмів машинного навчання створив нові можливості в безлічі сфер в тому числі і в сфері 3-х вимірної творчості. Один з прикладів інтеграції алгоритмів машинного навчання для застосування у 3D-графіці стали так звані «денойзери», що зменшують час рендерингу фізично коректних рушіїв, адже замість збільшення кількості семплів для зменшення шуму, денойзер дає змогу відрендерити зашумлене зображення та позбутися небажаних шумів за допомогою нейронної мережі.

Це гарний приклад використання алгоритмів машинного навчання для полегшення життя художника. Проте є і приклади негативні, такі як популярні зараз генератори, що можуть створювати як зображення, так і 3D-моделі. Якість таких моделей наразі сумнівна, як і легальний аспект, адже дані мережі натреновані на безлічі робіт реальних людей і не можуть бути захищені авторським правом. Використання алгоритмів машинного навчання, може дійсно полегшити життя художника, проте не повинно витіснити його замінюючи унікальність та тяжку працю, на галюцинації штучного інтелекту.

Зазначивши це слід пояснити, як же це стосується теми кваліфікаційної роботи. Текстури є невід’ємною частиною роботи в багатьох напрямках 3D-графіки. Текстурування – це процес надання 3D-

об’єкту реалістичності. Один з найважливіших етапів роботи 3D-художника. Проте якісні фотореалістичні текстури не беруться ні звідки. Процес створення текстур за PBR-пайплайном досить довгий та вимагає якісного та високовартісного обладнання.

З розвитком технологій кожен художник має камеру високої роздільності у себе в кишені. Проте цього не достатньо для запису всієї інформації необхідної для створення якісної текстури. Необхідно також мати інформацію карти нормалей, що додасть об’єму та деталей яких немає на 3D-об’єкті, для оптимізації та підвищеного рівня реалізму. Карта шорсткості або ж roughness map – карта яку складно записати, адже фотограмметрія блискучих об’єктів досить важкий процес який потребує великої кількості ручної роботи як на етапі запису інформації, так і на етапі створення фінального асету чи текстури.

Дана карта впливає на те наскільки блискучим буде об’єкт у 3D-просторі, тобто вказує рушію на те чи є поверхня шороховатою та розсіює світло, чи ж навпаки відбиває його. Карта амбієнтної оклюзії дозволить додати локальні тіні на деталях текстури додаючи об’єму. На рисунку 1.1 зображено карту нормалей, карту кольору та карту шорсткості окремо.

Рисунок 1.1 – Приклад накладання PBR текстур на 3D-об’єкт

Слід також зазначити як інформація з зображень використовується рушієм 3D-рендеринга. Зважаючи на тип текстури значення записані в зображенні можуть зчитуватися в одноканальному режимі з набором значень від 0 до 1. Наприклад, карта металічності, де чим ближче значення до одиниці, тим металічніше виглядає об’єкт (хоча коректним для такої карти є саме бінарне значення – 0 або 1 так як об’єкт не може бути напівметалічним, проте з деяким рівнем художньої інтерпретації іноді потрібно використовувати проміжне значення). В прикладі ж з картою шорсткості, що також записує інформацію в одному каналі, вже прийнято використовувати повний діапазон від 0 до 1 де 0 – глянцевий об’єкт, а 1 – матовий.

Інший приклад інформації, записаної в текстури, це інформація про колір, чи карта нормалей. Кольорова інформація зазвичай записана в триканальному RGB-форматі. Прийнято використовувати різні кольорові профілі такі як ACEScg для візуальних ефектів, чи sRGB в відео ігровій індустрії. Дана текстура найбільш зрозуміла з візуального представлення, адже як вона виглядає в 2D-просторі так вона і накладатиметься, на 3D-об’єкт.

Найскладнішою ж в розумінні є карта нормалей. Дана карта є однією з найважливіших складових набору текстур у робочому процесі PBR і відіграє критичну роль у створенні ілюзії високо деталізованої геометрії без фактичного збільшення кількості полігонів моделі. Її основне призначення – модифікація напрямку нормалей поверхні, які використовуються системою рендерингу для обчислення освітлення та тіней. Замість того щоб мати єдину нормаль на великому полігоні, карта нормалей надає унікальні векторні дані для кожного текселя. Це дозволяє імітувати дрібні виступи, западини та зморшки, такі як пори шкіри, заклепки на металі чи злами на камені, що робить поверхню візуально значно більш складною та реалістичною, ніж її спрощена базова сітка (рисунок 1.2).

Рисунок 1.2 – Використання карти нормалей для оптимізації

Карта нормалей являє собою спеціальний тип текстури, де кольорові канали RGB інтерпретуються як компоненти (X, Y, Z) вектора нормалі. Типовий синій (або фіолетовий) відтінок більшості карт нормалей відповідає вектору, що вказує прямо «назовні» від поверхні (Z-координата), що є нормою для пласкої, незміненої поверхні. Відхилення від цього середнього значення, закодовані змінами в червоному (X) та зеленому (Y) каналах, вказують на імітовані висоти та низини. Цей принцип дозволяє досягти високої візуальної якості з мінімальними обчислювальними витратами на етапі рендерингу, оскільки складні обчислення геометрії замінюються на відносно швидке зчитування та застосування текстурних даних. Таким чином, карта нормалей є ключовим елементом оптимізації у 3D-графіці, особливо для застосувань в задачах рендерингу в реальному часі, таких як відеоігри [3]. На рисунку 1.3 продемонстровано, як рушій рендерингу сприймає інформацію з карти нормалей.

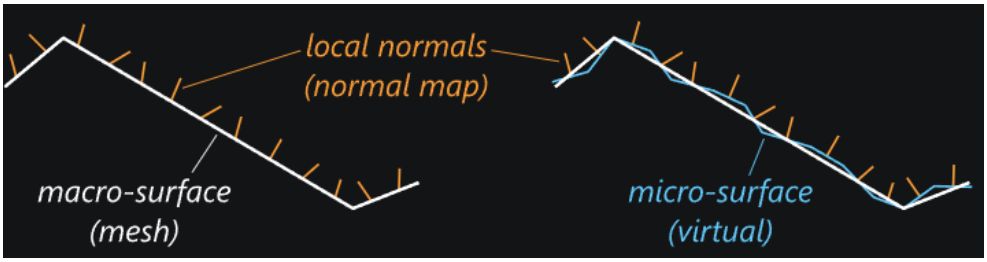


Рисунок 1.3 – Візуалізація роботи карти нормалей

Таким чином, PBR-пайплайн являє собою сучасний, науково обґрунтований підхід до створення та рендерингу 3D-матеріалів, який став індустріальним стандартом завдяки своїй здатності досягати винятkового фотовізуального реалізму. На відміну від застарілих методів, PBR ґрунтується на фізичних принципах збереження енергії та законах розсіювання світла, використовуючи стандартизований набір текстур. Цей пайплайн забезпечує постійну та передбачувану візуальну якість матеріалу незалежно від умов освітлення, що робить його незамінним у високоякісній 3D-графіці, від кінематографа до відеоігор. Методи машинного навчання, що розглядаються у даній роботі, спрямовані на автоматизацію та оптимізацію етапу генерації цих критично важливих PBR-текстур, що суттєво прискорює робочий процес 3D-художників та розробників.

1.2 Аналіз існуючих аналогічних систем

Перед початком розробки та проектування власної системи генерації PBR-текстур на основі зображень необхідним кроком є ґрунтовний аналіз існуючих комерційних та наукових рішень. Цей розділ присвячений критичному огляду сучасних підходів до вирішення завдання текстурування, зокрема тих, що використовують методи машинного навчання. Метою аналізу є ідентифікація переваг та недоліків провідних аналогів, оцінка їхньої ефективності, швидкості та якості згенерованих текстур. Результати цього огляду послужать фундаментом для визначення ключових вимог, вибору оптимальної архітектури нейронної мережі та формулювання інноваційних аспектів власної системи, що забезпечить її конкурентоспроможність та технологічну доцільність.

1.2.1 Застосунок для створення PBR текстур – Substance Sampler

Substance Sampler (рисунок 1.4) є ключовою професійною програмою в індустрії 3D-графіки, спеціально розробленою для швидкого та ефективного створення високоякісних PBR-матеріалів на основі одного вхідного зображення (фотографії чи скану). Його головна перевага полягає у гібридній методології, яка поєднує класичні алгоритми обробки зображень з потужними інструментами машинного навчання. Ядром функціоналу є ШІ-інструменти, такі як «delighter», що автоматично видаляє спрямоване освітлення та тіні з фотографії, щоб виділити чистий колір основи, та інтелектуальний механізм material extraction, який аналізує відбиття та структуру для точного виведення карт normal, roughness, metalness, та displacement. Таким чином, sampler перетворює неструктуровані візуальні дані на повний набір фізично коректних параметрів, готових до використання у будь-якому PBR-рендерері.

Рисунок 1.4 – Сплеш-скрін застосунку Substance Sampler

Substance Sampler має низку значних переваг, які зробили його індустріальним еталоном: він забезпечує високу якість і точність

згенерованих текстур, мінімізуючи артефакти, а його ШІ-функції значно прискорюють робочий процес, що є критично важливим для комерційних проектів. Функція автоматичного створення безшовних текстур також високо цінується. Водночас, з точки зору наукового дослідження та розробки аналогічних систем, існують і важливі недоліки. sampler є пропрієтарним комерційним продуктом, що унеможливлює глибокий аналіз або модифікацію його внутрішніх алгоритмів машинного навчання, які ефективно працюють як «чорний ящик». Крім того, хоча sampler є потужним, він все ще може мати обмеження у створенні карт нормалей або шорсткості для дуже складних або неоднорідних матеріалів, залишаючи простір для вдосконалення через розробку спеціалізованих дослідницьких моделей, орієнтованих на конкретні аспекти PBR-параметризації [4].

1.2.2 Застосунок для створення PBR текстур – Materialize

Materialize (рисунок 1.5) – це безкоштовний інструмент для генерації текстурних карт, який є популярною альтернативою комерційним рішенням у незалежних розробників та освітніх установах. На відміну від гібридних систем на кшталт substance sampler, materialize повністю покладається на класичні алгоритмічні методи обробки зображень. Він дозволяє користувачеві імпортувати базове зображення і, використовуючи контраст, виявлення країв та різні фільтри, ітеративно генерувати допоміжні PBR-карти: height map, з якої потім виводяться normal map та displacement map, а також карти metallic, smoothness (яку можна інвертувати в roughness) та ambient occlusion. Програма надає повний візуальний контроль над кожним етапом генерації, дозволяючи користувачеві вручну налаштовувати такі параметри, як інтенсивність глибини, деталізація та згладжування, перед експортом готового набору текстур.

Рисунок 1.5 – Логотип програмного застосунку Materialize

Головною перевагою Materialize є його доступність та прозорість робочого процесу, що ідеально підходить для навчальних цілей та для досліджень, де потрібно розуміти кожен крок алгоритмічної трансформації. Він також чудово підходить для генерації базових карт висот, з яких можна створювати безшовні тайлінгові текстури. Однак його основний недолік полягає у відсутності інтелектуальних ШІ-компонентів, що критично обмежує його ефективність порівняно з сучасними аналогами. Оскільки Materialize не вміє автоматично «видаляти» освітлення, якість вихідних карт нормалей та кольору сильно залежить від рівномірності освітлення вхідної фотографії. Це вимагає від користувача додаткових маніпуляцій у зовнішніх редакторах і, як правило, призводить до менш реалістичних результатів на складних матеріалах, оскільки алгоритмічна екстракція глибини з контрасту часто поступається точності, яку забезпечує машинне навчання [5]. Однак даний застосунок роками був вибором більшості початківців у даній сфері і при коректній експлуатації може видавати дуже гарні результати.

1.2.3 Застосунок для створення PBR текстур – CrazyBump

CrazyBump – це спеціалізований програмний інструмент, який історично був одним із піонерів у сфері швидкої генерації карт нормалей та карт зміщення з базових фотографій або текстур (рисунок 1.6). Як і Materialize, він повністю базується на класичних алгоритмічних методах обробки зображень, а не на машинному навчанні. Його інтерфейс орієнтований на візуальний зворотний зв’язок у реальному часі, дозволяючи користувачеві маніпулювати параметрами контрасту, деталізації та глибини, щоб отримати максимально виразну карту нормалей. Хоча програма може генерувати й інші PBR-карти, як-от карти оклюзії та блиску, її основна спеціалізація та перевага завжди полягає в ефективному виведенні високодеталізованої карти нормалей шляхом аналізу країв та мікроконтрасту у вхідному зображенні.

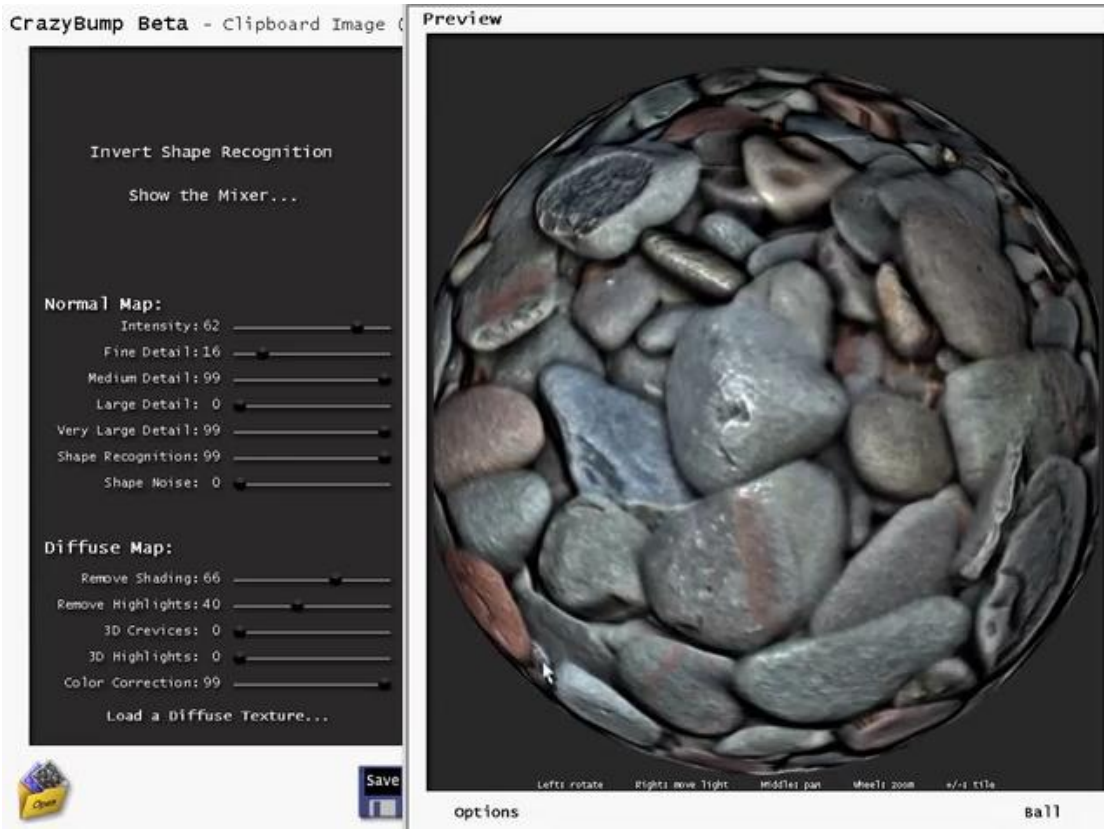


Рисунок 1.6 – Інтерфейс програмного застосунку CrazyBump

Ключова перевага CrazyBump полягає в його спеціалізації та простоті використання. Він забезпечує дуже швидкий і ефективний спосіб отримання високодеталізованих карт нормалей з мінімальними зусиллями, що ідеально підходить для швидкого прототипування або для художників, яким потрібен максимальний контроль над дрібною геометрією. Його візуальний, орієнтований на користувача робочий процес часто дозволяє досягти кращих результатів, ніж у більш універсальних алгоритмічних інструментів. Однак, його головним недоліком є вузька спеціалізація та застарілість. Він не є повноцінним PBR-інструментом і часто вимагає додаткових маніпуляцій з іншими картами (наприклад, Roughness чи Metalness), які генеруються менш якісно. Найважливіше – відсутність компонентів машинного навчання означає, що, як і Materialize, він не може автоматично усувати освітлення з вхідного зображення, що призводить до можливих артефактів тіней, «запечених» у карті нормалей, знижуючи її фізичну коректність у динамічному PBR-середовищі [6].

1.2.4 Застосунок для створення PBR текстур – PixPlant

PixPlant – це спеціалізоване програмне забезпечення, орієнтоване на створення високоякісних безшовних текстур та генерацію повного набору PBR-карт з одного вихідного зображення (рисунок 1.7). Його основною функцією є не лише створення допоміжних карт, але й забезпечення ідеальної безшовності текстури за допомогою алгоритмів клонування та змішування. Програма використовує традиційні методи обробки зображень та аналізу світла для виведення параметрів матеріалу. PixPlant пропонує потужний інструментарій для вирівнювання світла (хоча і не на основі ШІ, як у Sampler), а також інтерактивні 3D-вікна перегляду, що дозволяють художнику бачити, як згенеровані карти виглядають на 3D-моделі під різними умовами освітлення. Це робить його надійним інструментом для

архітектурної візуалізації та створення ігрових ассетів, де повторюваність текстури є критично важливою.

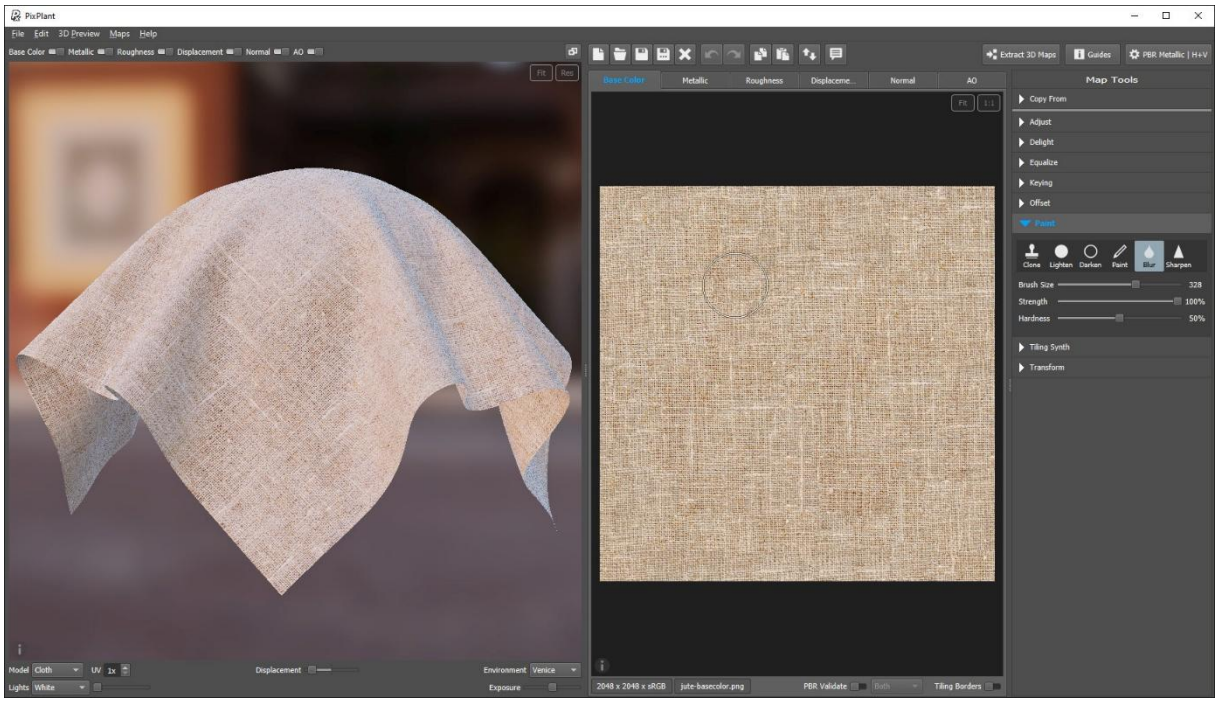


Рисунок 1.7 – Інтерфейс програмного застосунку PixPlant

Основною перевагою PixPlant, яка виділяє його серед інших алгоритмічних аналогів, є його неперевершена функція безшовності. Він дуже ефективно автоматизує процес створення плитки, мінімізуючи видимі шви та повторювані патерни, що є складним завданням для ручної обробки або базових алгоритмів. Крім того, він забезпечує повний набір PBR-карт і має відносно інтуїтивно зрозумілий робочий процес. Однак, головний недолік PixPlant, як і в інших класичних інструментів, полягає у відсутності сучасних можливостей машинного навчання. Його методи «вирівнювання світла» є алгоритмічними і часто не можуть досягти такої ж точності у виділенні чистого кольору, як це роблять ШІ-моделі. Це призводить до того, що карти нормалей та шорсткості, виведені на основі зображень з нерівномірним освітленням, можуть містити небажані артефакти або бути

фізично менш коректними у складних PBR-сценах порівняно з результатами, отриманими за допомогою глибинного навчання [7].

В таблиці 1.1 можна побачити порівняння переваг та недоліків кожного застосунку у зручному та зрозумілому вигляді.

Таблиця 1.1 – Порівняння переваг та недоліків

	Окремий Застосунок	Повний обсяг текстур	Зручний інтерфейс	Швидка робота
Substance Sampler	+	+	-	-
Materialize	+	+	-	-
CrazyBump	+	-	-	+
PixPlant	+	+	+	+
Кваліфікаційна робота	-	-	+	+

Завершивши критичний аналіз існуючих аналогічних систем, включаючи оцінку їхніх ключових переваг та недоліків, було отримано чітке уявлення про архітектурні та функціональні потреби системи, що розробляється. Узагальнення результатів цього огляду дозволяє не лише сформулювати науково обґрунтовану постановку задачі, але й визначити конкретні функціональні вимоги до майбутнього застосунку. Таким чином, отримані дані слугують основою для розробки рішення, яке зможе ефективно подолати обмеження, властиві існуючим алгоритмічним та комерційним аналогам, особливо в контексті генерації фізично коректних PBR-текстур методами машинного навчання.

1.3 Постановка задачі

Метою даної кваліфікаційної роботи є розробка моделей штучної нейронної мережі для генерації зображень та реалізація її можливостей в аддоні для програмного рішення Blender 3D.

Необхідно провести дослідження та аналіз типів моделей ШНМ, що задовольняють умовам виконання кваліфікаційної роботи та реалізувати обрану модель на практиці.

Розроблений аддон повинен мати можливість завантажувати зображення обране користувачем, застосовувати модель нейронної мережі та виводити результат.

Для досягнення поставленої мети необхідно виконати наступні дії:

- проаналізувати предметну галузь;
- проаналізувати наявні аналоги;
- підготувати набір навчальних та тестових даних;
- провести аналіз зібраних даних;
- створити зручний спосіб працювати з зібраними даними;
- спроєктувати модель нейронної мережі;
- виконати навчання спроєктованої моделі;
- зберегти навчену модель;
- розробити аддон;
- протестувати коректність роботи розробленого аддону.

Отже, маючи на меті створення аддону для генерації PBR текстур методами машинного навчання, необхідно закласти фундамент для створення системи, що значно пришвидшить ро1боту багатьох користувачів, що працюють з фотореалістичними текстурами, сценами, чи об’єктами.

2 ВИБІР МОДЕЛЕЙ ТА МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ ГЕНЕРАЦІЇ ТЕКСТУР

2.1 Аналіз цифрових зображень

Для людини зображення складається з набору сигналів нервової системи, переданих від ока до мозку в результаті реакції даного органу зору на фотони світла, що відбиваючись від об’єктів зовнішнього середовища, в кінці кінців потрапляють до нашого ока.

Розглянемо, як людина змогла відтворити людське око, створивши інструмент, що дозволяє симулювати даний орган зору та зберігати інформацію, що подорожує простором у вигляді фотонів.

З появою фотоплівки, людина вперше створила пристрій, здібний записувати вищезгадану інформацію у вигляді зображення. Робота традиційної фотоплівки ґрунтується на явищі фотохімії і є аналогом функції сітківки людського ока.

З появою перших обчислювальних машин почалася епоха цифровізації. Цифрове зображення кардинально відрізняється від свого аналогового чи плівкового попередника, будучи за своєю суттю матрицею (сіткою) числових значень. Ця сітка складається з дискретних елементів – пікселів. Кожен піксель відповідає за запис і відображення певного кольору або відтінку в конкретній точці зображення. З точки зору комп’ютерної технології, двовимірне зображення представляється як двовимірний масив (або матриця), де кожна клітинка містить вектор даних, що визначає колір пікселя. Коли ми розглядаємо кольорове зображення, цей вектор розширюється, формуючи так звані колірні канали. Найпоширенішим і базовим форматом для відображення на екранах є модель RGB (red, green, blue), яка використовує адитивний синтез кольорів. У цій моделі кожен піксель має три незалежні числові значення, що відповідають інтенсивності червоного, зеленого та синього кольорів. Якщо,

наприклад, на кожен канал виділяється 8 біт (що є стандартом), то інтенсивність кожного кольору варіюється від 0 до 255. Поєднання цих трьох значень дає можливість відтворити понад 16 мільйонів відтінків, які сприймає людське око.

Базовий принцип представлення цифрового зображення полягає у його дискретизації та квантуванні. Дискретизація перетворює безперервне світлове поле реального світу на дискретну сітку пікселів, визначаючи його просторову роздільну здатність (наприклад, 1920 x 1080). Квантування ж визначає глибину кольору – кількість унікальних значень, які може набувати інтенсивність кольору в кожному каналі. Зображення зберігається у пам’яті комп’ютера як послідовність цих числових значень. Таким чином, для комп’ютера кольорове зображення W x H є тривимірним тензором (або масивом) розмірності W x H x 3, де останній вимір (3) відповідає колірним каналам RGB. Хоча існують й інші формати, такі як CMYK (для друку) або HSV/HSL (для маніпуляцій з відтінком та насиченістю), саме RGB залишається основою для відображення. Сучасні формати, як-от PNG, можуть додавати четвертий канал – alpha, який використовується для кодування рівня прозорості, проте це не змінює базового матричного принципу кодування кольору.

Саме таке матричне представлення візуальної інформації стає фундаментом для застосування методів машинного навчання та комп’ютерного зору. Оскільки для обчислювальної системи зображення є впорядкованим багатовимірним масивом числових даних, це відкриває можливість використання складних математичних перетворень, таких як операція згортки, для аналізу його вмісту. У контексті штучних нейронних мереж така структура вхідних даних дозволяє алгоритмам автоматично виявляти приховані закономірності, текстурні патерни та геометричні особливості, фактично імітуючи процеси обробки сигналів у зоровій корі головного мозку. Таким чином, перехід від фізичної природи світла до його математичної абстракції у вигляді тензорів є ключовою передумовою для

створення інтелектуальних систем, здатних не лише зберігати візуальні дані, але й інтерпретувати їх та синтезувати нові реалістичні зображення [8].

2.2 Попередня обробка зображень у системах комп'ютерного зору

Штучні нейронні мережі (ШНМ) являють собою потужний обчислювальний апарат, здатний виявляти приховані закономірності та синтезувати нову інформацію шляхом ієрархічної обробки сигналів у багатошаровій структурі. Проте, з математичної точки зору, нейромережа оперує виключно числовими тензорами, виконуючи над ними послідовність матричних операцій та нелінійних перетворень. Така архітектурна особливість накладає суворі вимоги до статистичних властивостей вхідних даних.

Попередня обробка зображень є фундаментальним етапом у конвеєрі машинного навчання, що виступає необхідною сполучною ланкою між сирими візуальними даними та вхідним шаром нейронної мережі.

Оскільки вихідні цифрові зображення, отримані з різних джерел, часто характеризуються гетерогенністю параметрів – варіативністю роздільної здатності, форматами кодування кольору та рівнями шуму – їх безпосереднє використання може призвести до нестабільності обчислювального процесу або зниження точності прогнозування.

Головна мета препроцесингу полягає у стандартизації вхідного потоку даних, приведенні їх до єдиного математичного формату, що дозволяє алгоритму зосередитися на виявленні значущих семантичних ознак, мінімізуючи вплив технічних артефактів та варіацій умов зйомки.

Однією з ключових процедур цього етапу є геометрична трансформація, насамперед масштабування. Архітектури глибоких згорткових мереж, такі як U-Net, зазвичай вимагають фіксованої розмірності вхідного тензора (наприклад, 256x256 або 512x512 пікселів) для коректного виконання матричних операцій. Процес зміни розміру часто

супроводжується інтерполяцією, яка дозволяє зберегти інформативність зображення при зменшенні кількості пікселів.

Крім того, на цьому етапі часто застосовуються методи аугментації даних – штучного збагачення навчальної вибірки шляхом випадкових поворотів, віддзеркалень чи афінних перетворень. Це не лише збільшує обсяг доступних даних для навчання, але й підвищує інваріантність моделі, роблячи її стійкою до змін орієнтації та масштабу об’єктів у реальних умовах експлуатації.

Критично важливим аспектом підготовки даних є нормалізація значень інтенсивності пікселів. У стандартному 8-бітному представленні значення яскравості кожного каналу знаходяться в діапазоні цілих чисел від 0 до 255.

Однак для ефективної роботи алгоритмів градієнтного спуску та функцій активації такі великі значення є не оптимальними. Тому виконується лінійне перетворення даних у діапазон з плаваючою комою [0, 1] або [-1, 1]. Така нормалізація забезпечує центрування даних навколо нуля та уніфікацію їх масштабу, що значно прискорює збіжність нейронної мережі під час навчання, запобігає проблемі затухання градієнтів та забезпечує рівномірний внесок кожного пікселя у формування ознак, незалежно від загальної яскравості сцени [9].

2.3 Обробка зображень високої роздільної здатності

Одним із суттєвих обмежень класичних архітектур згорткових нейронних мереж, таких як U-Net, є фіксована розмірність вхідного шару (зазвичай 256x256 або 512x512 пікселів), що обумовлено лімітами відеопам’яті (VRAM) та особливостями навчання.

При роботі з вхідними даними високої чіткості (2K, 4K або 8K) стандартний підхід передбачає глобальне зменшення масштабу до робочої

роздільної здатності моделі, що неминуче призводить до катастрофічної втрати високочастотних деталей та розмиття мікрорельєфу.

Для вирішення цієї проблеми у аддоні, що розробляється планується імплементувати стратегію тайлового інференсу. Цей метод передбачає алгоритмічну фрагментацію вихідного зображення високої роздільної здатності на матрицю менших зображень (тайлів), розмір яких відповідає вхідному тензору нейромережі.

Кожен фрагмент обробляється моделлю незалежно, після чого отримані результати реконструюються у єдину вихідну карту, зберігаючи оригінальну деталізацію джерела.

Критичним аспектом реалізації тайлового інференсу є усунення артефактів на межах суміжних фрагментів, які виникають через відсутність контекстної інформації у згорткових фільтрів на краях зображення (padding artifacts).

Для нівелювання цього ефекту застосовано метод «ковзного вікна з перекриттям». Суть методу полягає у тому, що кожен тайл вирізається з вихідного зображення із певним запасом (буферною зоною), який перекриває сусідні області.

Після проходження через нейромережу, для фінальної зшивки використовується лише центральна, «валідна» частина тайлу, де передбачення є найбільш точним, а периферійні області, що можуть містити спотворення, відкидаються.

Такий підхід гарантує повну візуальну когерентність та безшовність результуючої текстури високої роздільної здатності, дозволяючи отримувати PBR-карти рівня 4K навіть за допомогою моделі, навченої на зображеннях 256x256.

Порівнюючи метод тайлового інференсу з альтернативним підходом – використанням алгоритмів супер-роздільної здатності (AI Upscaling/Super-Resolution) – слід відзначити фундаментальну різницю у природі формування деталей.

Апскейлери, що працюють за принципом генерації нових пікселів на основі низькоякісного вхідного зображення, схильні до «генеративних галюцинацій» – створення правдоподібних, але фізично неіснуючих деталей, що може призводити до розсинхронізації між картою кольору та картою нормалей.

Натомість тайловий підхід забезпечує абсолютну відповідність згенерованої геометрії (нормалей) реальним пікселям вхідної фотографії, оскільки він не синтезує нову роздільну здатність, а опрацьовує наявну інформацію частинами.

Хоча тайловий інференс є більш обчислювально витратним через необхідність багаторазового запуску моделі, він є безальтернативним вибором для задач, де пріоритетом є точність відтворення фактури та збереження автентичності матеріалу, а не просто збільшення кількості пікселів.

2.4 Аналіз штучних нейромереж

Штучні нейронні мережі – це клас алгоритмів машинного навчання, архітектура та принцип дії яких натхненні біологічною організацією нервової системи мозку.

Базуючись на системі взаємопов’язаних обчислювальних елементів (штучних нейронів), організованих у шари, вони здатні апроксимувати складні нелінійні функції шляхом ітеративного налаштування внутрішніх вагових коефіцієнтів у процесі тренування. Ця здатність до самонавчання та узагальнення інформації дозволяє ШНМ ефективно вирішувати задачі, які важко формалізувати традиційними алгоритмічними методами, такі як розпізнавання образів, обробка природної мови чи генерація візуального контенту.

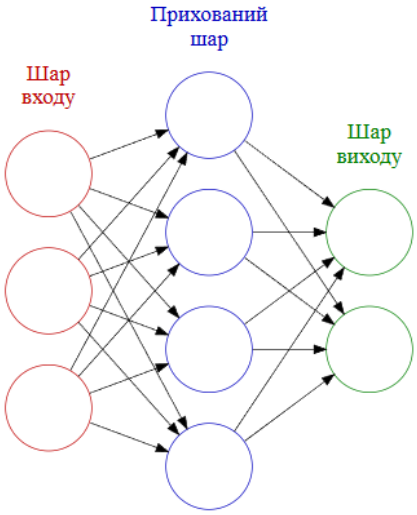


Рисунок 2.1 – Приклад візуалізації архітектури ШНМ

Моделі машинного навчання, особливо ті, що базуються на ШНМ набули широкого застосування в усіх сферах сучасних застосунків – від розважальних платформ та банківських систем до навігації та медицини. Причиною такої тотальної популярності є їхня висока ефективність в аналізі та синтезі даних. Наприклад, у сфері 3D-графіки, ШНМ дозволяють перетворити простий процес (як-от алгоритмічне виведення карт нормалей з карт висоти) на складну інтелектуальну генерацію повного набору PBR-карт з одного єдиного фото, враховуючи освітлення та приховані фізичні властивості матеріалу. Це значно прискорює створення високоякісного контенту та знижує вимоги до ручної праці 3D-художника.

Раніше розробка та впровадження складних моделей на основі ШНМ були прерогативою великих корпорацій через необхідність значних обчислювальних потужностей та спеціалізованого штату інженерів. Внесення навіть незначних змін вимагало значних ресурсів, що стримувало розвиток ринку незалежних розробників. Ситуація кардинально змінилася, коли такі компанії, як Google та Microsoft, інвестували у створення доступного інструментарію (TensorFlow, PyTorch, Google Colab). Це дозволило будь-кому, маючи мінімальні витрати часу та ресурсів,

створювати та тренувати власні моделі ШНМ, що відкрило шлях для швидкого розвитку нішевих рішень, зокрема у сфері автоматизації 3D-текстурування.

У порівнянні з класичними детермінованими алгоритмами або традиційними методами комп'ютерного зору, застосування глибоких нейронних мереж забезпечує якісно вищий рівень автоматизації та точності при відновленні об'ємних характеристик поверхні з одного зображення. Головною перевагою ШНМ є здатність вирішувати некоректно поставлені задачі шляхом семантичного аналізу контексту, ефективно розрізняючи візуальні імітації рельєфу та реальну геометрію, тоді як класичні фільтри часто помилково інтерпретують тіні або колірні патерни як зміни висоти. Втім, суттєвим недоліком нейромережевого підходу залишається висока обчислювальна складність та критична залежність результату від репрезентативності навчальної вибірки, на відміну від алгоритмічних методів, які є менш ресурсоємними та більш передбачуваними, проте вимагають значного ручного налаштування параметрів для кожного окремого випадку.

2.5 Вибір архітектури нейронної мережі

Для розв'язання завдання перетворення зображення на зображення яким є генерація набору PBR-текстур з одного вхідного зображення, найбільш доцільним вибором є архітектура U-Net (рисунок 2.2). Це спеціалізований тип згорткової нейронної мережі, який ідеально підходить для завдань, де вихідне зображення повинно мати той самий розмір і зберігати локальну просторову кореляцію, що й вхідне. U-Net була розроблена для біомедичної сегментації, але її структура енкодера-декодера з високоцінними з'єднаннями пропуску виявилася надзвичайно ефективною для синтезу текстур, оскільки вона дозволяє моделі одночасно розуміти глобальний контекст матеріалу (наприклад, цегла це чи дерево) та точно

відтворювати локальні деталі (гострі грані чи мікрорельєф). Це важливо, адже карта нормалей вимагає високої деталізації піксельного рівня, тоді як карта шорсткості залежить від загального контексту.

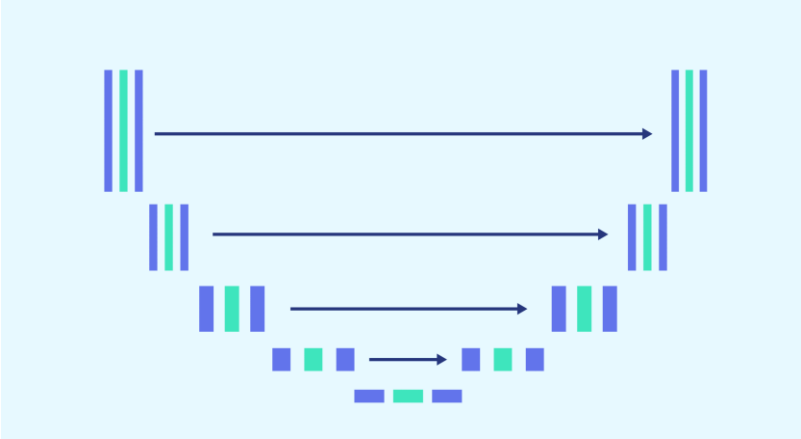


Рисунок 2.2 – Схема архітектури U-Net

Ключова особливість архітектури U-Net полягає у її симетричній V-подібній структурі, що включає два основні шляхи. Шлях стиснення (енкодер) функціонально подібний до традиційної згорткової нейронної мережі. Він складається з послідовних згорткових шарів та шарів пулінгу (max-pooling), які поступово зменшують просторовий розмір зображення і водночас збільшують кількість карт особливостей (глибину). Цей етап відповідає за витягування ієрархічних ознак – від простих країв і кутів на початкових шарах до високоабстрактних, семантичних ознак на глибинних шарах. У контексті генерації PBR-текстур, енкодер вчиться розпізнавати не просто колірні плями, а визначати тип матеріалу та його загальні властивості, необхідні для формування базового шаблону шорсткості або основних контурів зміщення. На виході енкодера ми отримуємо компактне представлення, що містить глибокий контекстний зміст вхідного зображення.

Другим і не менш важливим є шлях розширення – декодер, який відповідає за відновлення роздільної здатності зображення до його

початкового розміру. Декодер використовує розгортку зі збільшенням кроку (transposed convolution, або upsampling) для поступового збільшення просторового розміру карт особливостей. Під час цього процесу модель синтезує вихідне зображення, використовуючи глибокий контекст, отриманий від кінця енкодера. Однак, саме тут проявляється геніальність U-Net: безпосереднє відновлення з низької роздільної здатності призвело б до розмитих і неякісних результатів. Щоб уникнути цього, U-Net використовує з'єднання пропуску – прямі конкатенації карт особливостей із відповідних етапів енкодера. Ці з'єднання передають деталізовану інформацію про низькорівневі ознаки (наприклад, точні координати країв) безпосередньо в декодер, дозволяючи йому генерувати вихідні PBR-карти з необхідною високою піксельною точністю та гостротою.

Таким чином, використання архітектури U-Net є оптимальним вибором, оскільки вона забезпечує два критичні аспекти: глобальне розуміння та локальну точність [10]. Кожен вихідний канал моделі навчається незалежно генерувати відповідну карту, використовуючи повний контекст вхідного зображення. Хоча в роботі можуть бути використані модифікації U-Net, такі як Pix2Pix (який додає дискримінатор для підвищення фотореалістичності виходів), базова U-подібна структура залишається ядром, оскільки вона гарантує, що модель ефективно вчиться, як інвертувати процес захоплення зображення, витягуючи з колірної інформації невидимі фізичні параметри, необхідні для реалістичного PBR-рендерингу. Дана архітектура ШНМ також має переваги над більш складними GAN моделями, а саме більш коректне відтворення деталей вхідного зображення на відміну від можливих генеративних галюцинацій. В таблиці 2.1 продемонстровано, як саме виглядає архітектура U-net моделі, для задачі перетворення зображення на зображення.

Таблиця 2.1 – Архітектура штучної нейронної мережі типу U-net

№	Блок/шар	Функція активації	Роль (енкодер/декодер)
1	Rescaling / Normalization	-	Підготовка даних
2	Conv2D Block (x2)	ReLU	Енкодер: стиснення (Блок 1)
3	MaxPooling2D	-	Зменшення розміру
4	Conv2D Block (x2)	ReLU	Енкодер: стиснення (Блок 2)
5	MaxPooling2D	-	Зменшення розміру
6	Conv2D Block (x2)	ReLU	Дно: глибокий контекст
7	UpSampling2D / Conv2DTranspose	ReLU	Декодер: розширення (Блок 1)
8	Concatenation (Skip)	-	З’єднання з енкодером 2
9	Conv2D Block (x2)	ReLU	Декодер: обробка з деталями
10	UpSampling2D / Conv2DTranspose	ReLU	Декодер: розширення (Блок 2)
11	Concatenation (Skip)	-	З’єднання з енкодером 1
12	Conv2D (1x1)	Sigmoid або Tanh	Фінальний вихід (генерація карт)

Наступним етапом в розробці ШНМ є вибір метриків оцінки навчання, що є критично важливим для якісного навчання моделей, запобігання локальних мінімумів та інших проблем з навчанням.

2.6 Метрики оцінки навчання

У рамках проєктування системи навчання нейронної мережі для задачі регресії зображень, як базові критерії оптимізації обрано класичні попіксельні метрики: середню абсолютну похибку (Mean Absolute Error, MAE/L1) та середньоквадратичну похибку (Mean Squared Error, MSE/L2).

Вибір цих функцій зумовлений їхньою диференційованістю, що є необхідною умовою для алгоритму зворотного поширення помилки, та здатністю забезпечувати стабільну збіжність на ранніх етапах навчання.

Зокрема, функція втрат L2, яка базується на квадратичній різниці інтенсивностей пікселів, є стандартним вибором для задач регресії, оскільки вона сильно штрафує великі відхилення, змушуючи модель уникати значних помилок у прогнозуванні глобальної геометрії об’єкта.

Водночас функція L1, яка мінімізує модуль різниці, вважається більш стійкою до викидів у даних і, згідно з літературними джерелами, меншою мірою схильна до створення артефактів розмиття порівняно з L2.

Проте, теоретичний аналіз задач відновлення зображень вказує на суттєве обмеження обох метрик: вони базуються на припущенні про незалежність кожного пікселя, ігноруючи локальні кореляції та просторову структуру зображення.

У випадку невизначеності, коли одному вхідному патерну може відповідати декілька ймовірних варіантів вихідної текстури (мультимодальний розподіл), мінімізація MSE математично призводить до усереднення всіх можливих варіантів. Це явище, відоме як «регресія до середнього», створює ризик того, що модель ігноруватиме високочастотні деталі, такі як пори, подряпини чи шум, генеруючи візуально пласкі та розмиті текстури, які мають низьку похибку MSE, але не відповідають вимогам перцептивного фотореалізму [11].

Для подолання обмежень попіксельних метрик та забезпечення високої перцептивної якості генерації, у методику дослідження закладено використання функції перцептивних втрат, також відомої як втрата по контенту.

Цей підхід базується на гіпотезі, що оцінка подібності зображень повинна відбуватися не у просторі інтенсивностей пікселів, а у багатовимірному просторі семантичних ознак.

Для реалізації цього планується використання попередньо навченої на масивному датасеті ImageNet згорткової нейронної мережі VGG-19, яка виступатиме в ролі незмінного екстрактора ознак.

Теоретична перевага цього методу полягає у порівнянні карт активації (feature maps), отриманих з різних шарів мережі VGG, що дозволяє оцінювати відповідність зображень на різних рівнях абстракції: від низькорівневих геометричних примітивів (кути, лінії, градієнти) до високорівневих текстурних патернів.

Очікується, що включення перцептивної складової у функцію втрат дозволить вирішити проблему спектрального зсуву, змушуючи генеративну модель відновлювати високочастотні компоненти зображення та зберігати чіткість країв, навіть якщо вони мають незначні просторові відхилення від еталону, які б сильно штрафувалися метриками L1/L2. Такий комплексний підхід до формування цільової функції теоретично повинен забезпечити баланс між точністю кольоропередачі та збереженням структурної цілісності мезоструктури матеріалу [14].

3 ПРОЕКТУВАННЯ ТА ВИБІР АРХІТЕКТУРИ АДДОНУ

3.1 Функціональні вимоги до аддону

Функціональні вимоги до проєкту мають забезпечити наступний можливості для кінцевого користувача:

– користувач повинен мати можливість завантажити (обрати) базове зображення (фотографію) з локального диска через стандартний інтерфейс Blender;

– користувач повинен мати можливість автоматично призначити завантажене зображення як base color (albedo) для нового або обраного матеріалу в Blender;

– користувач повинен мати можливість ініціювати процес генерації PBR-карт з обраного зображення, передаючи його як вхідні дані завантаженій ML-моделі;

– модель повинна генерувати PBR-текстури: normal map, roughness map;

– аддон повинен повідомляти користувача про успішне завершення генерації та призначення текстур;

– аддон повинен забезпечувати зручне видалення згенерованих текстур із пам’яті та матеріалу Blender;

– адміністратори проєкту мають мати доступ до тренування моделі.

Дані функціональні вимоги мають забезпечити задуманий функціонал для користувача та адміністраторів проєкту. Поставлені функціональні вимоги дозволяють зрозуміти, в якому напрямку слід рухатися на подальших етапах розробки.

3.2 Вибір архітектури аддону

Критично важливим етапом у розробці аддону Blender для генерації PBR-текстур є обґрунтований вибір архітектури системи. Некоректне архітектурне рішення може не лише ускладнити базову реалізацію головної функціональності – інтеграцію моделі машинного навчання та автоматичне підключення текстур – але й суттєво знизити якість кінцевого продукту, особливо швидкість виконання та стабільність. Зміна архітектури на пізніх етапах розробки завжди призводить до значних часових витрат, що є не бажаним в умовах обмеженого терміну кваліфікаційної роботи. Отже, до вибору архітектури слід підходити стратегічно, ретельно зважуючи вимоги до швидкодії, можливості локального виконання та перспективи подальшої інтеграції.

Вибір архітектури повинен бути підкріплений ретельним аналізом пріоритетів розробки. У даному випадку ключовими факторами є локальне виконання моделі, простота інтеграції в середовище Blender та необхідність мінімізації залежностей для кінцевого користувача. Наприклад, можна було б розглянути варіант використання зовнішнього хмарного API для виконання моделі, що підвищило б вимоги до інтернет-з’єднання та ускладнило б розгортання. Проте, з огляду на цілі роботи, що передбачають локальне виконання натренованої моделі, оптимальною є трирівнева архітектура, яка чітко розділяє інтерфейс користувача (GUI аддона), бізнес-логіку (попередня обробка та виклик моделі) та сам рівень даних (файл натренованої моделі).

Керуючись принципом достатньої функціональності на початковому етапі, слід зосередитися на ключовій перевазі: автоматичній генерації PBR-карт. Надлишкові функції, такі як, наприклад, розширена ручна корекція згенерованих карт усередині аддону або підтримка різних фреймворків ML, які не використовуються, повинні бути відкладені. На ранньому етапі головною цільовою аудиторією є 3D-художники, потреба яких – швидке та

якісне отримання фізично коректних текстур з фото. Таким чином, обрана трирівнева архітектура має забезпечити максимальну простоту розгортання та швидкість виконання прогнозу, залишаючи при цьому архітектурний простір для додавання більш складних функцій, як-от використання GPU-прискорення або інтеграція додаткових моделей, у майбутніх версіях системи (рисунок 3.1).

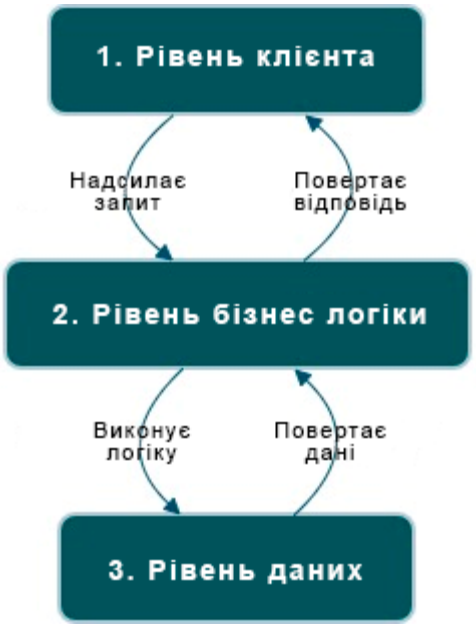


Рисунок 3.1 – Схема трирівневої архітектури

Ключова перевага обраної локальної трирівневої архітектури полягає у можливості незалежної розробки та вдосконалення моделі машинного навчання та самого аддона Blender. Натренована модель, що відповідає за генерацію PBR-текстур, може бути розроблена та верифікована окремо (наприклад, у середовищі Google Colab), а потім локально збережена у вигляді оптимізованого файлу, незалежного від основного коду аддона. Це забезпечує значну гнучкість для подальшого розвитку: модель можна перетреновувати з новим датасетом або замінювати на більш складну архітектуру (наприклад, з U-Net на cGAN), не вимагаючи при цьому повної переробки інтерфейсу користувача та логіки аддона. Це спрощує тестування

та гарантує, що покращення ML-ядра можуть бути швидко інтегровані в кінцевий продукт.

Застосунок (аддон Blender) відповідає виключно за інтерфейс, попередню обробку зображення, локальне виконання натренованої моделі та автоматичне підключення згенерованих карт. Залежно від вимог до продуктивності, може бути передбачено використання GPU-прискорення через спеціалізовані бібліотеки (наприклад, CUDA), або ж, як альтернативний шлях розвитку, створення клієнт-серверної архітектури, де модель виконуватиметься на потужному віддаленому сервері. Така гнучкість дозволяє безлічі різних систем (наприклад, інші 3D-редактори або окремі програми для текстурування) теоретично використовувати ту саму локально збережену модель. Ця схема повністю задовольняє потреби поточної кваліфікаційної роботи, гарантуючи модульність та перспективи для масштабування обох ключових компонентів проєкту.

Таким чином, вибір локальної трирівневої архітектури обґрунтований тим, що він є найбільш доцільним для розробки складних ML-моделей в умовах обмеженого часу. Він дозволяє зосередити зусилля на вдосконаленні саме якості генерації PBR-текстур, не відволікаючись на інтеграційні складнощі. Оскільки модель зберігається локально і викликається безпосередньо, немає залежності від зовнішніх API та інтернет-з’єднання, що підвищує стабільність та доступність застосунку для кінцевого користувача. Цей підхід пришвидшує розробку та гарантує, що навіть проста натренована модель може бути швидко та ефективно інтегрована у професійне 3D-середовище [13].

В ході розробки планується реалізувати архітектуру нейронної мережі (наприклад, на основі U-Net), призначену для перетворення одного вхідного зображення на набір PBR-карт. Тренування моделі відбуватиметься за допомогою обчислювальних ресурсів сервісу Google Colab та бібліотеки TensorFlow/Keras або PyTorch. Набір даних, а також фінальна натренована модель, зберігатимуться на Google Drive для

забезпечення зручного доступу та версійності. Модель буде експортовано у формат, придатний для локального виконання в середовищі Python.

Кінцевий застосунок буде створено як розширення (аддон) для Blender 3D з використанням мови Python. Аддон забезпечить графічний інтерфейс користувача для імпорту вхідного зображення та запуску процесу генерації. Головна перевага полягає у локальному виконанні моделі за допомогою Python-бібліотек, інтегрованих у Blender, таких як OpenCV або інших оптимізованих рішень. На даному етапі не планується створення віддаленого клієнт-серверного застосунку; однак, завдяки гнучкості мови Python, при необхідності додавання більш «важких» моделей (наприклад, для класифікації відео або складного 3D-сканування), архітектура дозволить перенести виконання прогнозу на окремий сервер (створюючи, наприклад, REST API на основі Flask або Django).

Розібравшись із загальною трирівневою архітектурою системи, перейдемо до деталізації застосунку, через який кінцевий користувач – 3D-художник – взаємодіятиме з моделлю машинного навчання. Для організації коду аддону буде використано популярну схему розробки програмного забезпечення MVC (рисунок 3.2).

Цей шаблон, хоча і є внутрішньою організацією коду, критично важливий для забезпечення модульності, повторного використання та зручності підтримки аддону. Схема MVC працює на принципі чіткого поділу відповідальності: model зберігає основний функціонал і бізнес-логіку; view формує інтерфейс користувача та відповідає за відображення; а controller зв’язує ці дві частини, обробляючи дії користувача та ініціюючи зміни в Моделі.

Застосунки, що використовують шаблон MVC, працюють за принципом розділення інтересів. модель містить усі правила, які керують даними та їхніми змінами – у нашому випадку це логіка попередньої обробки та виклик ML-моделі. view – це частина, яка є доступною користувачу для взаємодії (графічний інтерфейс аддона в Blender).

Controller служить посередником: він приймає команди від view (наприклад, натискання кнопки «Згенерувати»), інтерпретує їх, надає Моделі необхідні дані (шлях до зображення) і, отримавши результат, оновлює view. Така схема забезпечує високу безперебійність роботи та легкість внесення змін, оскільки критичні помилки в одній частині (наприклад, графічні проблеми у view) не зупиняють виконання основної логіки (генерація текстур у model).

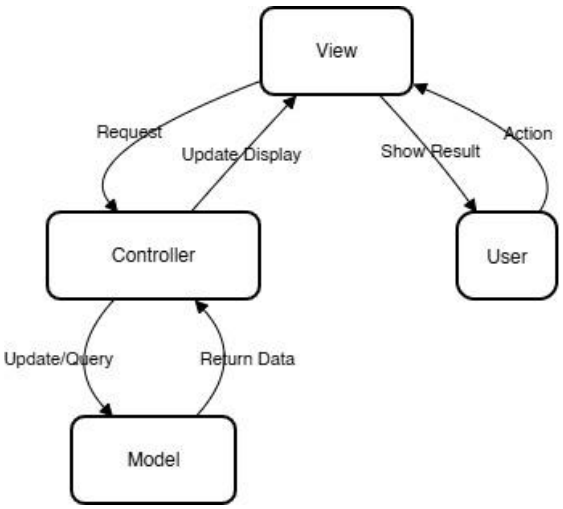


Рисунок 3.2 – Схема MVC

В розробленому проєкті, відповідно до шаблону MVC, ролі буде розподілено наступним чином:

– view (вид): це графічний інтерфейс користувача (GUI), реалізований засобами фреймворку Blender Python API (bpy). Він відповідає за поля для вибору зображення, кнопки ініціації генерації та відображення статусу процесу;

– controller (контролер): це основний скрипт аддона, який містить обробники подій (event handlers). Він приймає команди від view, верифікує вхідні дані та викликає відповідні методи в моделі;

– model (модель): це бізнес-логіка аддона, що включає функції попередньої обробки зображення, ініціацію завантаження локальної

натренованої ML-моделі та її виконання (inference), а також фінальну обробку згенерованих карт та їхнє автоматичне підключення до шейдера Blender [14].

3.3 Проєктування архітектури аддону

В системі, що розробляється, пріоритет віддається простоті взаємодії для будь-якої категорії користувачів, від професіоналів, що добре знайомі з принципами функціонування Blender 3D та подібних йому інструментів, до новачків-любителів, які ще не відчувають себе комфортно, працюючи з даним застосунком. Для досягнення даної мети планується побудувати доповнення, що добре інтегрується в інтерфейс Blender (рисунок 3.3) та не порушує логіку роботи даного інструменту.

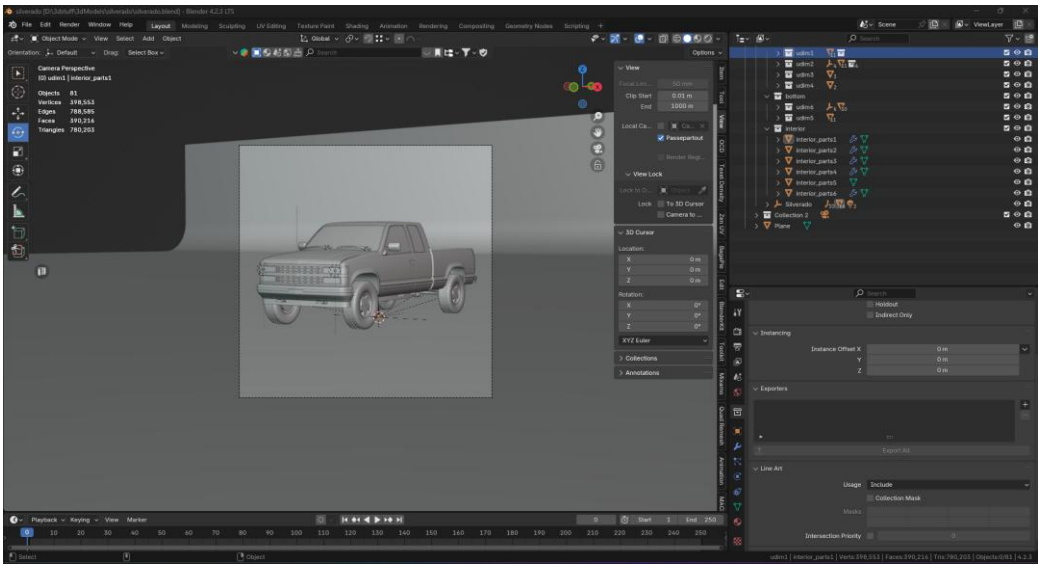


Рисунок 3.3 – Приклад інтерфейсу Blender 3D

Принцип роботи системи полягає в тому, що користувач, перебуваючи в середовищі Blender 3D, ініціює завантаження зображення через інтерфейс аддона. Важливо забезпечити максимальну зручність цієї дії, оскільки аддон має інтегруватися у звичний робочий процес 3D-

художника. Після вибору зображення, система одразу переходить до етапу генерації зображення (тобто PBR-карт). Процес виконується в фоновому режимі Python-скриптом. Ключовим моментом є завантаження моделі генератора (локально збереженого файлу ML-моделі) у пам’ять Blender для виконання прогнозу. Після успішної генерації, користувач одразу бачить збереження вихідного результату – автоматичне підключення згенерованих карт до активного матеріалу в Blender (рисунок 3.4).

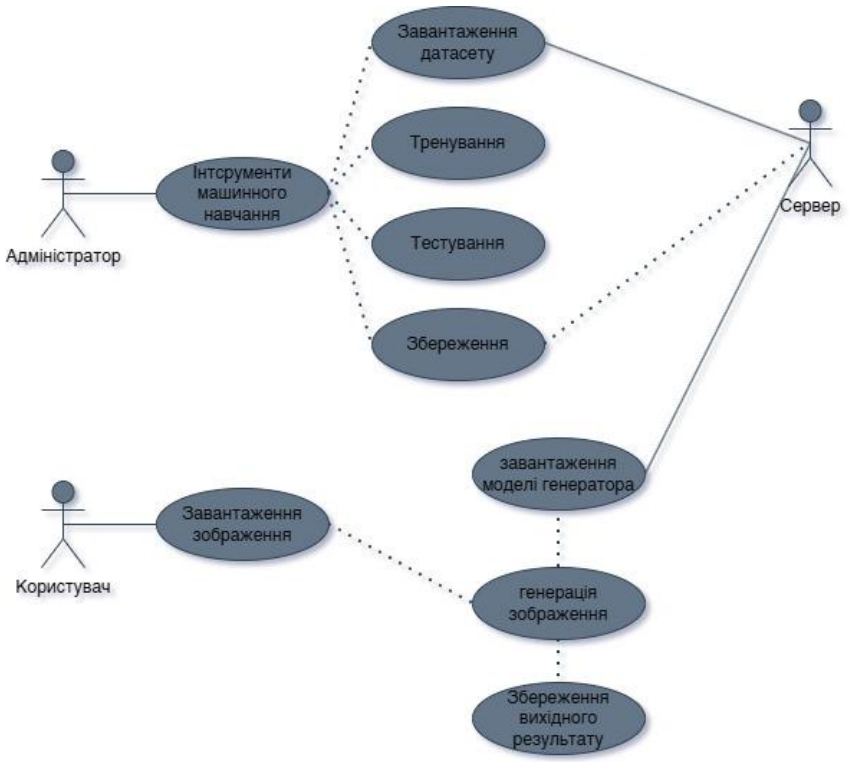


Рисунок 3.4 – UML-схема взаємодії користувача з системою

Таким чином, буде розроблено аддон, що є максимально простим та інтуїтивно зрозумілим для 3D-митців, забезпечуючи мінімальну кількість кроків між вхідним зображенням та готовим PBR-матеріалом. При цьому критично важливо врахувати формат та об’єм вхідних зображень. Оскільки модель машинного навчання має чіткі умови до вхідних даних (розмір, канали, нормалізація), зображення, обране користувачем, повинно пройти обов’язкову попередню обробку. Це дозволить забезпечити коректну

роботу моделі та не перевантажувати обчислювальні ресурси (CPU/GPU), що використовуються Blender.

Ці виключення та умови необхідно опрацювати в логіці аддона. Наприклад, якщо користувач обирає зображення, аддон має бути здатним коректно валідувати формат і, за потреби, вивести повідомлення про несумісність або автоматично конвертувати зображення до підтримуваного формату (наприклад, JPG або PNG). Також бізнес-логіка повинна включати функцію зміни розміру зображення (resizing) до необхідного вхідного розміру моделі (наприклад, 256x256 або 512x512 пікселів). Це забезпечує стабільність роботи системи, відсікаючи небажані формати та перевантаження. Проте, функціонал системи не обмежується лише кінцевим користувачем.

Адміністратори проєкту також повинні мати зручний доступ до необхідного функціоналу. Цей функціонал реалізується поза Blender (на сервері або в середовищі Google Colab). Він включає інструменти машинного навчання для тренування та тестування моделі, а також можливість завантаження датасету та збереження фінальної моделі. Така незалежність гарантує, що вдосконалення ядра генерації PBR-текстур може відбуватися паралельно з розвитком аддона, що є ключовою перевагою обраної трирівневої архітектури та дозволить експериментувати з різними підходами до вирішення даної задачі генерації текстур та дасть змогу швидко тестувати нові рішення.

3.4 Підсумок кінцевої архітектури аддону

Таким чином для створення програмного застосунку було обрано трирівневневу архітектуру як основу для проєкту, що забезпечує чітке розділення відповідальності між рівнем клієнта, рівнем бізнес-логіки та Рівнем даних.

Це архітектурне рішення гарантує незалежність розробки та можливість вдосконалення ML-моделі окремо від кінцевого застосунку. Для організації коду аддона був обраний шаблон Model-View-Controller, що підвищує модульність та спрощує інтеграцію інтерфейсу Blender з ядром обробки.

Візуалізація функціональних вимог системи була представлена у вигляді UML-діаграми варіантів використання, яка відобразила ключові сценарії взаємодії користувача та адміністратора.

Нарешті, було проведено глибоке дослідження та обрано архітектуру U-Net як оптимальну для задачі перетворення зображення на зображення.

Важливим елементом проєктування стало теоретичне обґрунтування вибору метрик оцінки якості генерації. Аналіз обмежень класичних попіксельних метрик, таких як MAE та MSE, виявив їхню схильність до усереднення результатів, що є неприйнятним для задач відновлення деталізованих текстур. У зв’язку з цим, для забезпечення високої перцептивної якості вихідних карт нормалей та шорсткості, було визначено стратегію оптимізації на основі функції перцептивних втрат.

Використання попередньо навченої мережі VGG-19 у ролі екстрактора ознак дозволяє оцінювати структурну подібність зображень на різних рівнях абстракції, що теоретично гарантує збереження високочастотних деталей та мінімізацію артефактів розмиття.

Таким чином, обрана архітектурна стратегія повністю задовольняє вимоги проєкту, гарантуючи його масштабованість, надійність та ефективність виконання.

4 ВИБІР ТЕХНОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ

Застосунок, що інтегрує функції машинного навчання для автоматизованої генерації PBR-текстур, повинен бути побудований з урахуванням специфічних вимог 3D-середовища. Зважаючи на функціональні та архітектурні вимоги, описані в попередніх розділах, необхідно вибрати набір технологій, що забезпечить безшовну інтеграцію в робочий процес Blender 3D. Оскільки проєкт складається з трьох ключових рівнів – візуальний інтерфейс користувача, ядро обробки та рівень моделі, вибір технологій має відповідати кожному з них: Python та Blender API є невід'ємними для розробки клієнтської частини, тоді як TensorFlow/Keras забезпечує функціонал для локального виконання натренованої моделі машинного навчання (рисунок 4.1).

Рисунок 4.1 – Розмежування back-end та front-end областей

Front-end доповнення Blender – це все, що користувач бачить і з чим взаємодіє безпосередньо у вікні програми: це спеціалізовані панелі (panels), меню, кнопки, поля для введення та інші елементи керування, створені за допомогою Blender Python API. Ця частина відповідає за презентацію даних,

налаштувань та запуск функцій. Розробник фронтенду використовує класи bpy.types.Panel для розміщення інтерфейсу у потрібному місці, наприклад, на N-панелі або у вкладці properties, та елементи layout для структуризації. Мета фронтенду – забезпечити інтуїтивно зрозумілий та органічний досвід користувача, максимально інтегрований у філософію Blender. Всі ці елементи інтерфейсу самі по собі не виконують складні операції з 3D-моделлю чи даними, вони лише збирають вхідні параметри від користувача (наприклад, значення масштабу, кут повороту, тип операції) і викликають основну логіку.

Back-end доповнення Blender – це його справжня функціональність, яка відповідає за обробку даних, маніпуляції з 3D-сценою та виконання складних обчислювальних завдань. В аддонах Blender цю роль виконують оператори, які є головним механізмом для виконання дій. Оператор – це клас, успадкований від bpy.types.Operator, який містить метод execute (). Саме в цьому методі знаходиться бізнес-логіка доповнення, написана на Python з використанням API Blender: створення нової геометрії, застосування модифікаторів, виконання складних математичних розрахунків, маніпуляції з матеріалами або анімацією. Back-end бере дані, отримані від клієнтської сторони, взаємодіє з внутрішньою структурою даних .blend-файлу та повертає результат, який змінює сцену або об’єкти. Таким чином, бекенд є обчислювальним двигуном, відокремленим від візуального інтерфейсу, що забезпечує швидкість та стабільність роботи.

Розмежування front-end і back-end є прикладом патерну «Розділення відповідальності». Хоча обидві частини написані однією мовою і знаходяться в одному файлі чи теці аддону, їхні ролі чітко розділені: front-end викликає back-end, передаючи йому необхідні властивості. Цей зв’язок відбувається через механізм імені оператора. Таке архітектурне рішення є критично важливим для створення великих, підтримуваних та надійних доповнень: якщо потрібно змінити лише вигляд кнопки, не потрібно торкатися складної логіки обчислень, і навпаки. Така структура гарантує,

що зміни в інтерфейсі не призведуть до неочікуваних помилок у ядрі функціональності, дозволяючи full-stack розробнику Blender ефективно масштабувати свій проєкт.

4.1 Мова програмування Python

Python – це високорівнева, інтерпретована мова програмування загального призначення, яка була створена Гвідо ван Россумом (Guido van Rossum) та вперше випущена у 1991 році. Його основна філософія, чітко сформульована у «Дзені Python», акцентує увагу на читабельності коду та простоті синтаксису, використовуючи значні відступи (індексацію) замість фігурних дужок, що суттєво полегшує його вивчення та підтримку. Завдяки своїй гнучкості, Python підтримує безліч парадигм, включаючи об'єктно-орієнтовану, імперативну та функціональну, що дозволяє застосовувати його для вирішення широкого спектру завдань: від простих скриптів автоматизації до створення складних корпоративних систем. Його кросплатформність (здатність працювати на Windows, macOS, Linux) та неймовірно велика та активна спільнота, яка підтримує безперервний розвиток величезної екосистеми сторонніх бібліотек, зробили Python де-факто стандартом у таких галузях, як веб розробка (Django, Flask), наукові обчислення (SciPy, NumPy) і, особливо, в науці про дані та штучному інтелекті, де він домінує завдяки своїй здатності швидко прототипувати та розгортати моделі.

У середовищі 3D-моделювання та візуалізації, Python слугує мовою для автоматизації та розширення функціоналу Blender. Вбудований інтерпретатор Python надає розробникам повний доступ до Blender Python API (bpy) – вичерпного набору модулів, які дозволяють програмно контролювати буквально кожен аспект сцени: маніпулювати сітками, створювати чи змінювати матеріали та текстури, керувати камерами й освітленням, а також налаштовувати інтерфейс користувача (UI) через

створення власних панелей та операторів. Оператори є ядром бекенду доповнення, де розміщується складна логіка, наприклад, алгоритми для генеративної процедурної побудови об’єктів або автоматичного риггінгу. Завдяки Python, 3D-артисти можуть автоматизувати монотонні та повторювані завдання (наприклад, груповий експорт об’єктів чи пакетне перейменування), інтегрувати Blender із зовнішніми інструментами та, що найважливіше, створювати високоспеціалізовані робочі процеси, які є критично важливими для ефективності у великих виробництвах анімації та відеоігор.

Python є абсолютно незамінним інструментом, що забезпечує всю необхідну інфраструктуру для машинного навчання та глибинного навчання. Роль Python у контексті задачі перетворення зображення на зображення багатогранна: він використовується для попередньої обробки величезних наборів даних текстур, для створення та конфігурації архітектури нейронної мережі через провідні бібліотеки, такі як PyTorch або TensorFlow/Keras. Python забезпечує керування складним тренувальним циклом та, нарешті, для тестування та візуалізації результатів тренування [15].

4.2 Бібліотека Tensor Flow

TensorFlow є однією з найпотужніших і найпоширеніших відкритих програмних бібліотек, розроблених компанією Google, і призначена вона для високопродуктивних числових обчислень, що особливо ефективно використовуються у сфері машинного навчання. В основі TensorFlow лежить концепція графів потоку даних, де обчислення представлені у вигляді спрямованого графа, а вузли цього графа виконують математичні операції над багатовимірними масивами даних, які називаються тензорами. Цей підхід дозволяє бібліотеці автоматично оптимізувати обчислювальний процес і забезпечує можливість розподіленого виконання на різних

пристроях. Для такого завдання, як генерація PBR-текстур, де обробляються великі обсяги графічних даних (зображення, представлені як тензори), TensorFlow надає необхідну швидкість та інструментарій для ефективного керування цими числовими потоками від вхідного зображення до згенерованих вихідних карт.

Ключова архітектурна перевага TensorFlow полягає в його масштабованості та кросплатформеності. Бібліотека розроблена таким чином, що одна й та сама модель може бути натренована на потужних кластерах із використанням спеціалізованих тензорних процесорів (TPU) у хмарі (наприклад, через Google Colab), а потім без змін розгорнута для виконання на стандартних CPU, GPU користувачів, або навіть на мобільних і вбудованих пристроях. Така гнучкість є критичною для проєкту, оскільки вона дозволяє проводити ресурсомістке тренування нейронної мережі U-Net у високопродуктивному хмарному середовищі, а потім безперешкодно інтегрувати та виконувати висновок вже на локальному комп'ютері користувача в середовищі аддона Blender. Здатність TensorFlow ефективно використовувати апаратне прискорення, наприклад GPU, є необхідною умовою для досягнення прийнятної швидкості генерації текстур у реальному часі.

Сучасна екосистема TensorFlow значною мірою зосереджена навколо його високо рівня API – Keras, що є потужною надбудовою, яка значно спрощує розробку, тестування та налагодження нейронних мереж, дозволяючи дослідникам і розробникам зосередитися на логіці моделі, а не на низькорівневих обчислювальних деталях. Keras надає готові будівельні блоки, такі як шари Conv2D (згортковий шар), Pooling та Conv2DTranspose (Розгорткова Згортка), які є основою для побудови складної архітектури U-Net. Завдяки Keras, побудова цільової моделі для генерації PBR-текстур, включаючи складні з'єднання пропуску та множинні вихідні гілки, стає послідовним і зрозумілим процесом. Це прискорює

прототипування та забезпечує стандартизований підхід до побудови надійної архітектури.

У контексті задачі перетворення зображення на зображення TensorFlow пропонує спеціалізовані інструменти для роботи з зображеннями. Він має потужні утиліти для попередньої обробки даних – автоматичне перетворення зображень у тензори, зміна розміру, нормалізація та розділення на навчальні й тестові вибірки. Ці інструменти життєво необхідні для забезпечення того, щоб вхідні дані відповідали суворим вимогам моделі U-Net. Крім того, TensorFlow забезпечує механізми для збереження та завантаження натренованої моделі у стандартизованому форматі, що є останнім і критичним кроком перед інтеграцією в аддон Blender. Це гарантує, що вся логіка, навчена на хмарних ресурсах, може бути швидко та ефективно викликана Python-скриптом аддона для здійснення генерації PBR-текстур безпосередньо на робочій станції користувача [16].

4.3 Сервіс Google Colab

Google Colaboratory (Colab) – це безкоштовний хмарний сервіс, розроблений Google, який надає середовище для роботи з ноутбуками Jupyter. Його основною перевагою є демократизація доступу до обчислювальних ресурсів, необхідних для роботи з великими даними та складними алгоритмами машинного навчання, що особливо актуально для незалежних дослідників і студентів. Фактично, Colab дозволяє запускати код Python у браузері, усуваючи необхідність у складних налаштуваннях локального середовища, встановленні залежностей та потужного обладнання. У контексті завдань, які вимагають інтенсивних обчислень, як-от тренування глибоких нейронних мереж, Colab стає незамінним інструментом, оскільки забезпечує інтеграцію з екосистемою Google та дозволяє безпосередньо використовувати такі бібліотеки, як TensorFlow та PyTorch.

Ключовим функціоналом, що робить Colab критично важливим для завдань машинного навчання, є доступ до прискорювачів апаратного забезпечення (GPU та TPU). Тренування складних архітектур, таких як U-Net для генерації PBR-текстур, є надзвичайно ресурсомістким і може займати тижні або навіть місяці на стандартних центральних процесорах. Використання виділених графічних процесорів, які доступні в середовищі Colab, значно скорочує час навчання. Ця можливість дозволяє експериментувати з різними гіперпараметрами, тестувати різні варіанти архітектури моделі та швидко ітерувати процес розробки. Таким чином, Colab вирішує одну з головних проблем, яка раніше обмежувала невеликі команди та освітні проєкти – брак обчислювальної потужності.

Colab також надає зручний механізм для управління даними та співпраці. Завдяки інтеграції з Google Drive, користувачі можуть легко монтувати своє хмарне сховище безпосередньо в середовище ноутбука, що дозволяє швидко завантажувати великі навчальні набори даних і зберігати результати тренування, включаючи файли натренованої моделі. Це значно спрощує логістику роботи з великими обсягами графічних даних, необхідних для навчання моделей перетворення зображення на зображення. Крім того, функція спільного доступу, аналогічна Google Docs, дозволяє кільком користувачам одночасно працювати над одним і тим самим ноутбуком, що полегшує командну роботу, спільне налагодження коду та обмін експериментальними результатами.

Незважаючи на всі переваги, Colab вимагає від користувача врахування певних особливостей. Сесії, які використовують безкоштовні GPU-ресурси, мають обмежену тривалість і можуть бути припинені після певного періоду неактивності або досягнення ліміту використання, що вимагає імплементації механізмів збереження контрольних точок під час тривалого навчання. Проте, цей сервіс залишається ідеальною платформою для виконання навчальних проєктів. Він забезпечує гнучке, потужне та доступне хмарне середовище, необхідне для успішного тренування та

валідації складних моделей машинного навчання перед їхнім фінальним розгортанням на локальних пристроях або серверах [17].

4.4 Інструменти Blender API

Blender Python API (bpy) є невід'ємним інструментом і ключовим компонентом клієнтського рівня проєкту. Цей API є повноцінною програмною оболонкою, яка дозволяє розробникам взаємодіяти з будь-яким елементом, даними чи функціоналом 3D-середовища Blender за допомогою скриптів на мові Python. У контексті створення аддона, bpy слугує мостом, який зв'язує складну логіку машинного навчання з візуальним світом 3D-графіки. Він забезпечує механізми для створення користувацького інтерфейсу – додавання нових панелей, кнопок та полів для завантаження зображення користувачем, що є першим етапом у ланцюжку генерації. Без bpy неможливе було б вбудувати функціонал аддона безпосередньо в існуючі вікна Blender, забезпечуючи інтуїтивне і нативне використання для 3D-художника.

Функціональність bpy значно виходить за межі лише розробки інтерфейсу; він також є контролером та диспетчером усього процесу обробки даних. Як тільки користувач натискає кнопку «Генерувати PBR-текстури», bpy викликає відповідний оператор, який інкапсулює бізнес-логіку аддона. Цей оператор використовує можливості bpy для доступу до даних сцени: ідентифікації активного об'єкта, отримання шляху до вхідного зображення та, що найважливіше, ініціації та запуску зовнішніх обчислювальних процесів. Саме через bpy скрипт Python отримує необхідні параметри та викликає функції, що взаємодіють з TensorFlow для виконання висновку натренованої моделі. Таким чином, API не лише приймає команди, але й керує потоком даних між 3D-сценою та обчислювальним ядром.

Незважаючи на свою потужність, робота з Blender API вимагає розуміння специфічної структури даних Blender та його циклу виконання. З

точки зору архітектури, bpy є частиною клієнтського рівня, але його функціонал пронизує бізнес-логіку і навіть частково рівень презентації (створення інтерфейсу). Вибір Python як мови програмування для API забезпечує високу сумісність з ключовими ML-бібліотеками, такими як NumPy та TensorFlow, що дозволяє уникнути складних міжмовних інтерфейсів. Таким чином, Blender Python API виступає як критична технологія, що забезпечує як візуальне, так і функціональне вбудовування системи генерації PBR-текстур у робочий процес 3D-художника [18].

4.5 Інструменти доповнення NumPy

NumPy є фундаментальною бібліотекою для наукових і числових обчислень у Python і відіграє критично важливу роль у будь-якому проєкті машинного навчання, включаючи генерацію PBR-текстур. Головною її концепцією є об’єкт ndarray (N-dimensional array) – ефективний багатовимірний масив. На відміну від стандартних списків Python, які є гнучкими, але повільними, масиви NumPy дозволяють зберігати дані одного типу (наприклад, числа з плаваючою комою) у неперервних блоках пам’яті. Це забезпечує значно вищу швидкість виконання обчислень, оскільки код NumPy оптимізовано і скомпільовано на нижчому рівні. Таким чином, NumPy забезпечує необхідну продуктивність для обробки великих обсягів даних, що є передумовою для роботи з зображеннями.

Критичне значення NumPy для даної роботи полягає у тому, як він представляє зображення. У комп’ютерній графіці та машинному навчанні, кольорове зображення є тривимірним тензором, а сіра карта, як-от roughness, є двовимірним масивом. Безпосередньо перед подачею в модель TensorFlow/Keras, усі вхідні дані мають бути конвертовані в об’єкти ndarray NumPy. Ця конвертація забезпечує стандартизований числовий формат, з яким може працювати нейронна мережа. Крім того, на етапі попередньої

обробки даних, функції NumPy надають швидкі та ефективні векторні операції: замість повільного ітеративного обходу пікселів, масиви NumPy дозволяють застосовувати математичні перетворення одразу до всього масиву.

NumPy також виступає як міст між логікою аддона Blender і ядром TensorFlow. Хоча TensorFlow використовує власні оптимізовані тензори для виконання моделі, вхідні та вихідні дані завжди проходять через NumPy. Коли аддон Blender, використовуючи bpy, зчитує вхідне зображення, він перетворює його піксельні дані у масив NumPy для подальшої обробки та передачі до TensorFlow. Аналогічно, після того, як модель U-Net завершує генерацію PBR-карт, вона повертає результат у вигляді тензорів, які негайно конвертуються назад у масиви NumPy. Саме ці масиви потім використовуються функціями bpy для створення нових файлів зображень і підключення їх до шейдерів Blender.

Завдяки своїй інтеграції з C/C++ кодом та ефективному використанню пам’яті, NumPy забезпечує не тільки швидкість, але й сумісність з іншими ключовими науковими бібліотеками, які часто використовуються у сфері обробки зображень та машинного навчання, такими як SciPy, Matplotlib та безпосередньо TensorFlow. Фактично, TensorFlow використовує синтаксис і методику NumPy для роботи з даними, що робить знання цієї бібліотеки обов’язковим для будь-якого розробника ML. У підсумку, NumPy забезпечує стабільну, швидку та стандартизовану основу для маніпулювання складними багатовимірними даними, які є критичними для успішного виконання нейронної мережі у даному проєкті генерації PBR-текстур [19].

4.6 Застосунок для роботи з 3D графікою – Blender 3D

Blender – це потужний, відкритий і кросплатформовий пакет для створення тривимірної комп’ютерної графіки. Він забезпечує повний

робочий процес для 3D-художників, включаючи моделювання, скульптінг, текстурування, анімацію, рендеринг і компонування (рисунок 4.2).

Рисунок 4.2 – Логотип та слоган застосунку Blender 3D

У контексті даного проєкту, Blender є цільовою платформою для розгортання та виконання кінцевого застосунку. Його ключова перевага полягає у вбудованій системі Python API (bpy), яка дозволяє розробникам створювати власні розширення – аддони. Ці аддони можуть інтегруватися безпосередньо в інтерфейс користувача, надаючи доступ до внутрішніх даних сцени, таких як геометрія, матеріали та графова система шейдерів. Таким чином, Blender виступає не просто як середовище для відображення, але як клієнтський рівень, що забезпечує необхідний інтерфейс і механізм для запуску, контролю та візуалізації результатів, згенерованих моделлю машинного навчання.

Історія Blender розпочалася у 1994 році як внутрішній програмний продукт голландської анімаційної студії NeoGeo, а його публічний реліз відбувся у 1998 році. Після банкрутства студії, у 2002 році, спільнота організувала безпрецедентну кампанію зі збору коштів, щоб викупити

вихідний код у інвесторів і випустити Blender як вільне програмне забезпечення з відкритим кодом. Цей перехід до моделі відкритого коду, під управлінням Blender Foundation, став поворотним моментом. Зараз Blender є одним із найвпливовіших інструментів у світі 3D-графіки, який регулярно оновлюється та вдосконалюється світовою спільнотою розробників і професіоналів. Його використання поширилося від незалежних художників і стартапів до великих студій, що працюють над кіно, візуальними ефектами та відеоіграми, підтверджуючи його статус як потужного, надійного та динамічно розвиваючогося програмного комплексу [20].

4.7 Хмарне сховище Google Drive

Google Drive – це хмарний сервіс зберігання даних і синхронізації файлів, розроблений компанією Google, який є невід'ємною частиною екосистеми Google Cloud. У контексті завдань машинного навчання, особливо при використанні сервісу Google Colab, Drive виконує критично важливу функцію централізованого та доступного сховища для великих обсягів даних. Навчання таких ресурсомістких моделей, як U-Net для генерації PBR-текстур, вимагає використання значних навчальних наборів даних, які можуть легко перевищувати доступний локальний простір на комп'ютері розробника. Drive дозволяє зберігати ці датасети в хмарі та легко монтувати їх безпосередньо у середовище виконання Colab. Це усуває необхідність постійно завантажувати гігабайти даних, значно прискорюючи робочий процес і забезпечуючи надійне резервне копіювання.

Крім зберігання вхідних навчальних даних, Google Drive слугує основним місцем для збереження та управління результатами роботи. Під час тривалого процесу тренування моделі, особливо коли використовуються безкоштовні сесії Colab з обмеженим часом, необхідно регулярно зберігати контрольні точки, а також фінальну натреновану модель у стандартизованому форматі (наприклад, TensorFlow SavedModel). Drive

забезпечує надійність цього процесу: натренована модель, яка є ядром усього застосунку, безпечно зберігається і може бути швидко завантажена в будь-який момент. На фінальному етапі проєкту цей файл натренованої моделі завантажується з Drive на локальний пристрій, щоб інтегрувати його у розроблений аддон Blender. Таким чином, Google Drive виступає як необхідний рівень даних для проєкту, забезпечуючи життєвий цикл моделі: від зберігання вхідних даних для тренування до фіксації готового, натренованого результату.

У результаті ґрунтовного аналізу було сформовано оптимальний технологічний стек, що повністю відповідає вимогам трирівневої архітектури проєкту. Вибір мови Python забезпечив як розробку клієнтського рівня через Blender Python API, так і функціональність обчислювального ядра, використовуючи бібліотеки TensorFlow/Keras для виконання моделі U-Net. Бібліотека NumPy була визначена як невід'ємна основа для швидкої та ефективної роботи з багатовимірними масивами даних. Нарешті, Google Colab і Google Drive були обрані як критично важливі хмарні сервіси для ресурсомісткого тренування моделі та надійного зберігання датасетів і фінальних результатів. Таким чином, обраний інструментарій гарантує високу продуктивність, модульність, кросплатформність і безшовну інтеграцію функціоналу машинного навчання безпосередньо в 3D-середовище Blender [21].

4.8 Бібліотека OpenCv

OpenCV (Open Source Computer Vision Library) – це високопродуктивна бібліотека з відкритим вихідним кодом, що стала де-факто стандартом у галузі комп'ютерного зору та обробки зображень. Вона надає розширений набір інструментарію для виконання широкого спектра операцій: від базових геометричних трансформацій та фільтрації до складних алгоритмів розпізнавання об'єктів та аналізу руху. Ключовою

перевагою OpenCV є її кросплатформеність та оптимізація під роботу в реальному часі, що досягається завдяки ефективній реалізації на C/C++ та підтримці апаратного прискорення. Особливу цінність для сучасних розробок становить модуль DNN, який дозволяє завантажувати та виконувати моделі глибокого навчання, натреновані у популярних фреймворках, таких як TensorFlow, PyTorch або Caffe, використовуючи універсальний формат ONNX.

У рамках розробки аддону для Blender бібліотека OpenCV виступає в ролі основного механізму інференсу нейронної мережі, замінюючи собою важковагові залежності TensorFlow. Вибір на користь OpenCV продиктований необхідністю забезпечення портативності та легкості встановлення кінцевого програмного продукту, оскільки ця бібліотека може бути легко локалізована всередині аддону без конфліктів із системним середовищем Python. Завдяки вбудованому модулю DNN, OpenCV забезпечує завантаження конвертованої моделі U-Net у форматі ONNX та її виконання на центральному процесорі, що гарантує стабільну роботу на будь-якому обладнанні користувача, незалежно від наявності спеціалізованих графічних прискорювачів.

Крім безпосереднього виконання нейромережі, OpenCV планується використовувати для етапів попередньої та постобробки графічних даних. Функціонал бібліотеки дозволить ефективно реалізувати алгоритми зміни розміру зображень (resizing) за допомогою білінійної інтерполяції, нормалізації піксельних значень та конвертації колірних просторів, що є необхідною умовою для коректної роботи моделі. Також засоби матричних операцій OpenCV будуть залучені для реалізації логіки тайлового інференсу, включаючи нарізку вхідного зображення високої роздільної здатності на фрагменти, додавання відступів та подальше безшовне зшивання результатів генерації у фінальні текстурні карти [22].

5 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

5.1 Вибір набору даних

Підготовка тренувального набору даних – одна з найважливіших задач тренування моделей машинного навчання. Якісні дані – це основа якісної моделі. Слід також врахувата об’єм набору даних адже для задачі генерації зображень кількість настільки ж важлива як і якість самих даних. Також важливим аспектом є легальність використання певних текстур для тренування моделі генератора, адже кожна текстура – це робота людей, які можуть бути проти використання їхньої інтелектуальної власності для тренування ШІ, і цю позицію слід поважати.

Зважаючи на це все було вирішено використати готовий набір даних «MatSynth», набір матеріалів для рендерингу, основаного на фізиці (PBR). Цей набір даних містить більше 4000 текстур високої роздільної здатності та пропонує високу різноманітність, деталізацію та об’єм.

Структура обраного набору даних передбачає представлення кожного матеріалу через уніфікований комплект текстурних карт, що, зокрема, включає: base color, diffuse, normal, height, roughness, metallic, specular, а також карту opacity (за необхідності). Такий комплексний підхід дозволяє детально моделювати як світловідбивні характеристики, так і особливості мезоструктури поверхні об’єкта, що є необхідною умовою для фізично коректного рендерингу.

Окрім графічної складової, кожен елемент вибірки супроводжується розширеним набором метаданих. Анотація включає відомості про походження матеріалу, умови ліцензування, категоризацію, семантичні теги, методику створення, а також, за наявності, фізичні розміри відображуваної ділянки. Наявність такої деталізованої інформаційної бази забезпечує можливість прецизійної фільтрації та відбору матеріалів, що

дозволяє адаптувати навчальну вибірку під специфічні вимоги задачі, що розв’язується.

Набір даних MatSynth сформовано шляхом систематичної агрегації інформації з низки відкритих онлайн-джерел, що функціонують у межах ліцензійних угод CC0 та CC-BY. Обрана стратегія збору даних забезпечила високу репрезентативність вибірки, дозволивши охопити широкий спектр матеріалів: від стандартизованих загальновживаних типів до унікальних вузькоспеціалізованих варіацій. Такий підхід гарантує не лише різноманітність навчальних даних, але й можливість їх легітимного використання для вирішення широкого кола прикладних та дослідницьких завдань.

5.2 Тренування, збереження і перевірка моделей машинного навчання

В ході виконання практичної частини кваліфікаційної роботи було створено декілька записників Google Colab з метою тренування, збереження та перевірки моделей машинного навчання. Перший записник було створено для завантаження набору даних, передоброки цих даних, створення та тренування U-Net моделі штучної нейронної мережі.

На початку записника було підключено необхідні бібліотеки та встановлено базові параметри, такі як розмір зображення, розмір набору даних та розмір батчу (рисунок 5.1).

```
import tensorflow as tf
import numpy as np
from datasets import load_dataset
from tqdm import tqdm

IMG_SIZE = 256
DATASET_LIMIT = 500
BATCH_SIZE = 8
```

Рисунок 5.1 – Початок роботи

Наступний етап – завантаження даних та їх передобробка. Так як обраний датасет підтримує завантаження за допомогою вбудованих функцій Python, даний етап є максимально простим та зрозумілим. Єдиний нюанс – створення токену на зчитування (рисунок 5.2) даних для сервісу «Huggingface» задля оптимізації підвантаження набору даних в Google Colab.

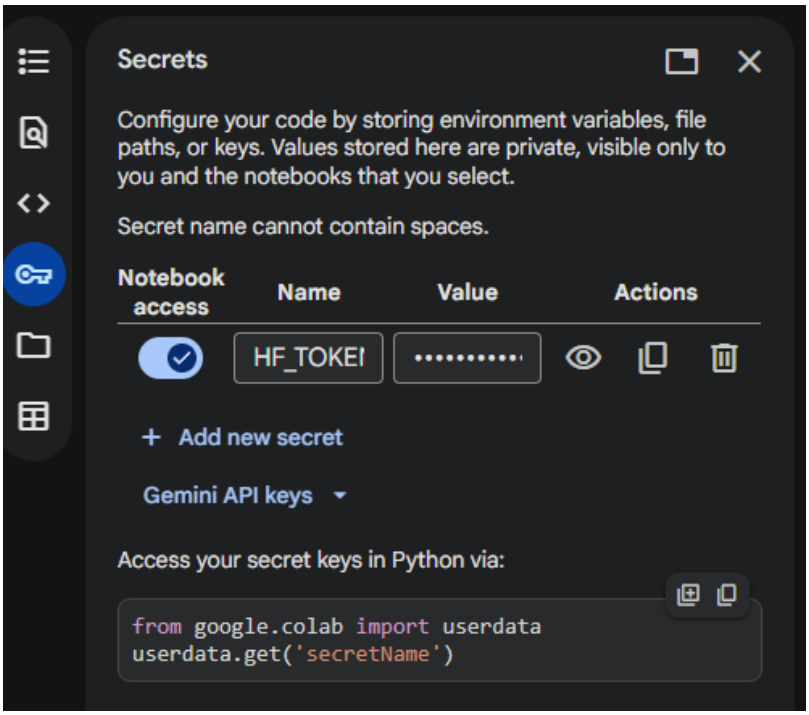


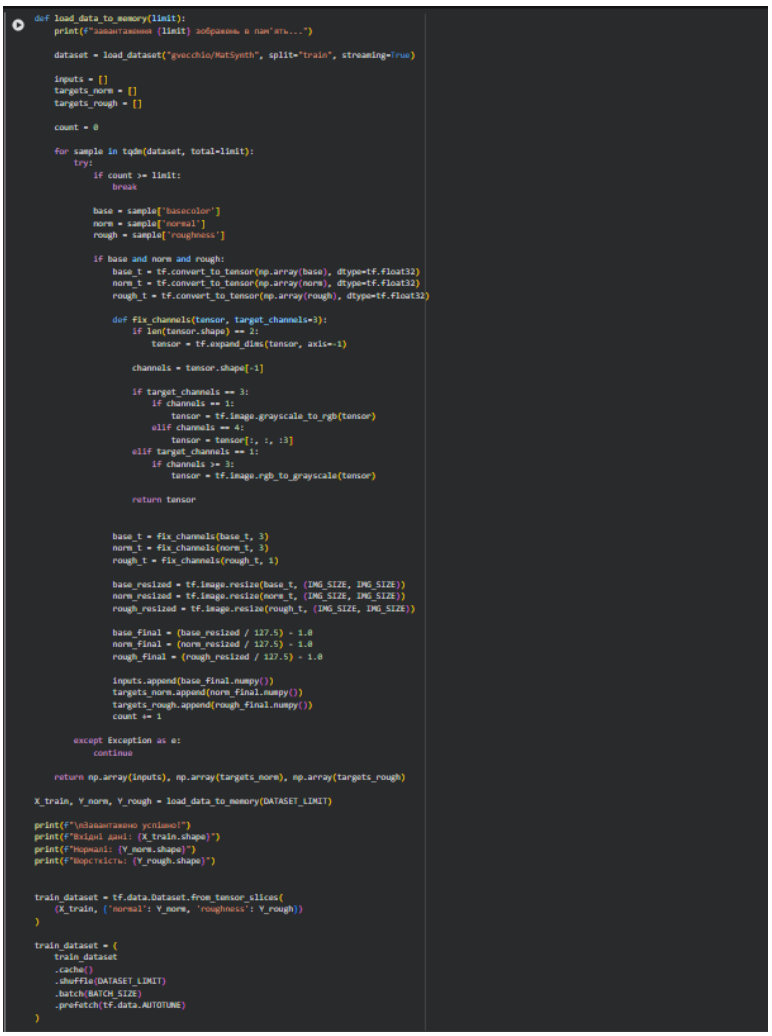
Рисунок 5.2 – Створення токену на зчитування

Завантаження набору даних було вирішено оптимізувати через обмеженість обчислювальних ресурсів в безкоштовній версії Google Colab (12 гігабайт оперативної пам’яті та 15 гігабайт відеопам’яті). Саме тому було вирішено обмежити набір даних до 1000 зображень, зтиснути дані зображення до роздільної здатності 256x256 та завантажити все в оперативну пам’ять для прискорення навчання.

Для реалізації етапу попередньої обробки даних було розроблено спеціалізований програмний модуль, який забезпечує автоматизоване завантаження, трансформацію та структурування навчальної вибірки з

хмарного репозиторію MatSynth. Процес розпочинається з потокового отримання зображень, що дозволяє ефективно працювати з великими обсягами даних без необхідності їх повного збереження на локальному диску, і включає ітеративний відбір заданої кількості зразків (у даному випадку, лімітованої константою DATASET_LIMIT). Кожен елемент вибірки, що складається з тріади зображень – базового кольору, карти нормалей та карти шорсткості, проходить через конвеєр перетворень, ключовим етапом якого є уніфікація розмірності каналів. Спеціально розроблена функція fix_channels аналізує тензорну структуру кожного зображення та приводить її до стандартизованого вигляду: триканального RGB для кольору та нормалей, та одноканального grayscale для карти шорсткості, що гарантує гомогенність вхідних даних для нейронної мережі.

На наступному етапі відбувається геометрична та радіометрична нормалізація даних. Всі зображення масштабуються до фіксованого розміру (256x256 пікселів) за допомогою білінійної інтерполяції, а значення інтенсивності пікселів, що початково знаходяться в діапазоні [0, 255], лінійно трансформуються у нормований простір ознак [0, 1]. Це є критично важливою умовою для забезпечення стабільності градієнтного спуску під час навчання глибоких згорткових мереж. Оброблені тензори агрегуються у масиви NumPy, після чого на їх основі формується оптимізований об’єкт tf.data.Dataset. Для підвищення продуктивності навчання застосовуються методи кешування, перемішування та попередньої вибірки, що дозволяє мінімізувати затримки вводу-виводу та забезпечити безперервне надходження даних до графічного процесора під час тренування моделі. На рисунку 5.3 зображено код функції, яка виконує вище описані дії. Впровадження описаної методики оптимізації дозволило ефективно нівелювати апаратні обмеження хмарного середовища, забезпечивши високу швидкість подачі даних без перевантаження доступної відеопам’яті.



```python
def load_data_to_memory(limit):
    print(f"Завантаження {limit} зображень в пам'ять...")

    dataset = load_dataset("gvecchio/MatSynth", split="train", streaming=True)

    inputs = []
    targets_norm = []
    targets_rough = []

    count = 0

    for sample in tqdm(dataset, total=limit):
        try:
            if count >= limit:
                break

            base = sample['basecolor']
            norm = sample['normal']
            rough = sample['roughness']

            if base and norm and rough:
                base_t = tf.convert_to_tensor(np.array(base), dtype=tf.float32)
                norm_t = tf.convert_to_tensor(np.array(norm), dtype=tf.float32)
                rough_t = tf.convert_to_tensor(np.array(rough), dtype=tf.float32)

                def fix_channels(tensor, target_channels=3):
                    if len(tensor.shape) == 2:
                        tensor = tf.expand_dims(tensor, axis=-1)

                    channels = tensor.shape[-1]

                    if target_channels == 3:
                        if channels == 1:
                            tensor = tf.image.grayscale_to_rgb(tensor)
                        elif channels == 4:
                            tensor = tensor[:, :, :3]
                    elif target_channels == 1:
                        if channels >= 3:
                            tensor = tf.image.rgb_to_grayscale(tensor)

                    return tensor


                base_t = fix_channels(base_t, 3)
                norm_t = fix_channels(norm_t, 3)
                rough_t = fix_channels(rough_t, 1)

                base_resized = tf.image.resize(base_t, (IMG_SIZE, IMG_SIZE))
                norm_resized = tf.image.resize(norm_t, (IMG_SIZE, IMG_SIZE))
                rough_resized = tf.image.resize(rough_t, (IMG_SIZE, IMG_SIZE))

                base_final = (base_resized / 127.5) - 1.0
                norm_final = (norm_resized / 127.5) - 1.0
                rough_final = (rough_resized / 127.5) - 1.0

                inputs.append(base_final.numpy())
                targets_norm.append(norm_final.numpy())
                targets_rough.append(rough_final.numpy())
                count += 1

        except Exception as e:
            continue

    return np.array(inputs), np.array(targets_norm), np.array(targets_rough)

X_train, Y_norm, Y_rough = load_data_to_memory(DATASET_LIMIT)

print(f"\nЗавантажено успішно!")
print(f"Вхідні дані: {X_train.shape}")
print(f"Нормалі: {Y_norm.shape}")
print(f"Шорсткість: {Y_rough.shape}")


train_dataset = tf.data.Dataset.from_tensor_slices(
    (X_train, {'normal': Y_norm, 'roughness': Y_rough})
)

train_dataset = (
    train_dataset
    .cache()
    .shuffle(DATASET_LIMIT)
    .batch(BATCH_SIZE)
    .prefetch(tf.data.AUTOTUNE)
)
```

Рисунок 5.3 – Початкова робота з даними

Для верифікації цілісності попередньо обробленого набору даних та візуальної оцінки якості вхідних тензорів було розроблено процедуру візуалізації на базі бібліотеки Matplotlib (рисунок 5.4). Алгоритм розпочинає роботу зі стохастичного відбору індексу з навчальної вибірки, що забезпечує отримання відповідної тріади даних: вхідного зображення базового кольору та еталонних карт нормалей і шорсткості. Ключовим етапом даного процесу є застосування спеціалізованої функції денормалізації, яка здійснює зворотне лінійне перетворення значень пікселів з діапазону [-1, 1], необхідного для ефективного навчання нейронної мережі, у діапазон [0, 1], що є стандартом для коректного відображення RGB-зображень на дисплеї.

У рамках програмної реалізації також передбачено виведення діагностичних метрик, зокрема мінімальних та максимальних значень інтенсивності «сирих» даних, що дозволяє контролювати коректність нормалізації. Графічне представлення результатів організовано у вигляді фігури з трьома графіками, де послідовно візуалізуються вхідні дані та цільові карти. При цьому для одноканальної карти шорсткості застосовується специфічна обробка розмірності та накладання сірої колірної карти, що дозволяє коректно інтерпретувати розподіл значень шорсткості поверхні, тоді як відключення координатних осей забезпечує фокусування на візуальних характеристиках текстур.

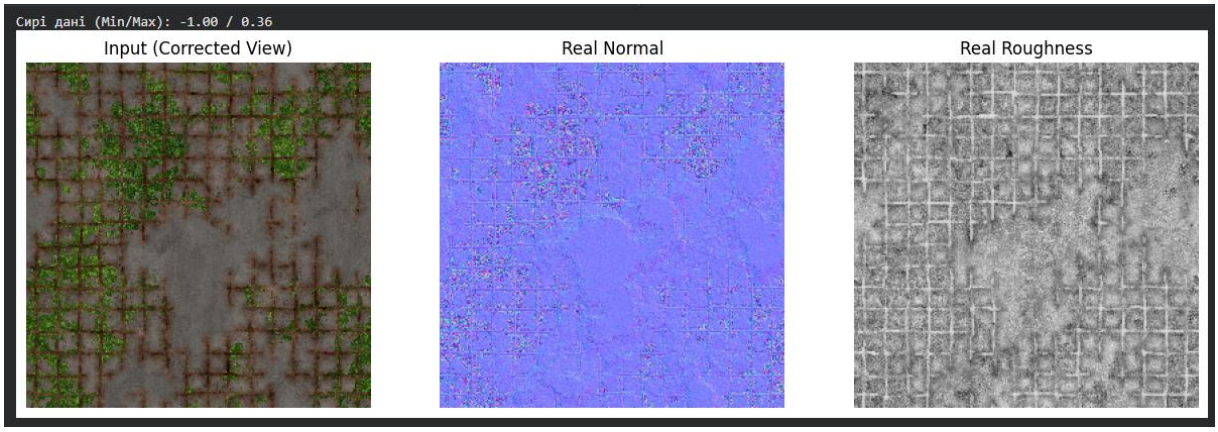


Рисунок 5.4 – Візуалізація опрацьованих даних

Архітектура розробленої згорткової нейронної мережі базується на класичній U-Net структурі, модифікованій для задачі багатоканальної регресії зображень. Мережа складається з двох основних шляхів: енкодера (шлях стиснення) та декодера (шлях розширення). Енкодер послідовно зменшує просторову розмірність вхідного тензора за допомогою операцій згортки (Conv2D) з функцією активації ReLU та підвибірки (MaxPooling2D), що дозволяє моделі екстрагувати ієрархічні ознаки різного рівня абстракції – від простих геометричних примітивів до складних текстурних патернів. Центральна частина мережі – «bottleneck»,

містить найбільшу кількість фільтрів (1024), забезпечуючи глибоке контекстне представлення вхідних даних.

Декодер відповідає за відновлення просторової роздільної здатності зображення до початкового розміру. Для уникнення артефактів типу «шахівниця», характерних для операції транспонованої згортки, у даній реалізації використано комбінацію білінійної інтерполяції (UpSampling2D) та стандартної згортки. Ключовою особливістю архітектури є використання з'єднань пропуску, які передають деталізовані карти ознак безпосередньо з відповідних шарів енкодера до декодера. Це дозволяє відновити втрачену під час пулінгу просторову інформацію, що є критично важливим для точної генерації високочастотних деталей текстур. Фінальна частина мережі розгалужується на дві незалежні голови з активацією tanh, що генерують карту нормалей (3 канали) та карту шорсткості (1 канал) у нормалізованому діапазоні значень [-1, 1].

Після створення моделі лишається лише її скомпілювати та зберегти. На етапі компіляції нейронної мережі було визначено конфігурацію процесу навчання, ключовим елементом якої є використання адаптивного оптимізатора Adam, що забезпечує ефективну збіжність алгоритму. В якості цільової функції втрат для обох вихідних гілок моделі – генерації карти нормалей та карти шорсткості – обрано середню абсолютну похибку, яка дозволяє мінімізувати попіксельне відхилення між синтезованими та еталонними текстурами, забезпечуючи високу стійкість до викидів у даних. Безпосереднє навчання моделі ініціюється методом fit і виконується протягом 10 епох на попередньо сформованому датасеті, при цьому метрика MAE використовується як кількісний індикатор якості регресії для моніторингу динаміки мінімізації помилки в процесі ітеративного оновлення вагових коефіцієнтів.

Після тренування дані було візуалізовано. Для якісної верифікації результатів роботи навченої нейронної мережі розроблено процедуру візуального порівняльного аналізу, яка дозволяє зіставити згенеровані

текстурні карти з еталонними даними. Алгоритм розпочинає роботу зі стохастичного відбору індексу тестового зразка з навчальної вибірки, після чого вхідний тензор зображення подається на вхід моделі для виконання прямого проходу. Отримані в результаті прогнозування тензори карти нормалей та карти шорсткості, разом із відповідними їм істинними значеннями, проходять обов'язковий етап постобробки за допомогою спеціалізованої функції денормалізації. Ця функція здійснює лінійне перетворення значень пікселів з робочого діапазону мережі [-1, 1] у стандартний простір відображення [0, 1], що є необхідною умовою для коректної візуалізації даних бібліотекою Matplotlib.

Графічне представлення результатів організовано у вигляді матриці підграфіків розмірністю 2x3, що забезпечує зручне зіставлення вхідних даних та результатів генерації. Верхній ряд фігури демонструє вхідне зображення базового кольору та еталонні карти, тоді як нижній ряд відображає відповідні результати роботи нейронної мережі. При візуалізації враховано специфіку розмірності даних: триканальні карти нормалей відображаються у просторі RGB для коректної передачі векторної інформації, а для одноканальних карт шорсткості застосовується сіра колірна карта, що дозволяє чітко інтерпретувати розподіл параметрів глянцевості поверхні без спотворення кольоровими градієнтами.

Аналіз отриманих результатів експериментального моделювання демонструє, що розроблена архітектура U-Net успішно оволоділа здатністю до екстракції глобальних семантичних ознак та розуміння морфології вхідного зображення. Зокрема, на карті шорсткості чітко простежується структурна сегментація поверхні: модель коректно ідентифікувала розташування цеглин та швів, відтворивши загальний патерн розподілу матеріалів. Це свідчить про ефективну роботу енкодера та коректну передачу контекстної інформації через з'єднання пропуску. Втім, суттєвим недоліком є критична втрата високочастотних деталей, що проявляється у значному розмитті текстури шорсткості та фактичній відсутності корисної

інформації на карті нормалей, яка колапсувала до усередненого значення (плоского вектора). Така поведінка є характерною для моделей, що навчаються виключно на попіксельних функціях втрат (L1/MAE або L2/MSE), які схильні до згладжування результатів задля мінімізації статистичної похибки, ігноруючи перцептивну чіткість та мікрорельєф. Результат тренування моделі можна побачити на рисунку 5.5.

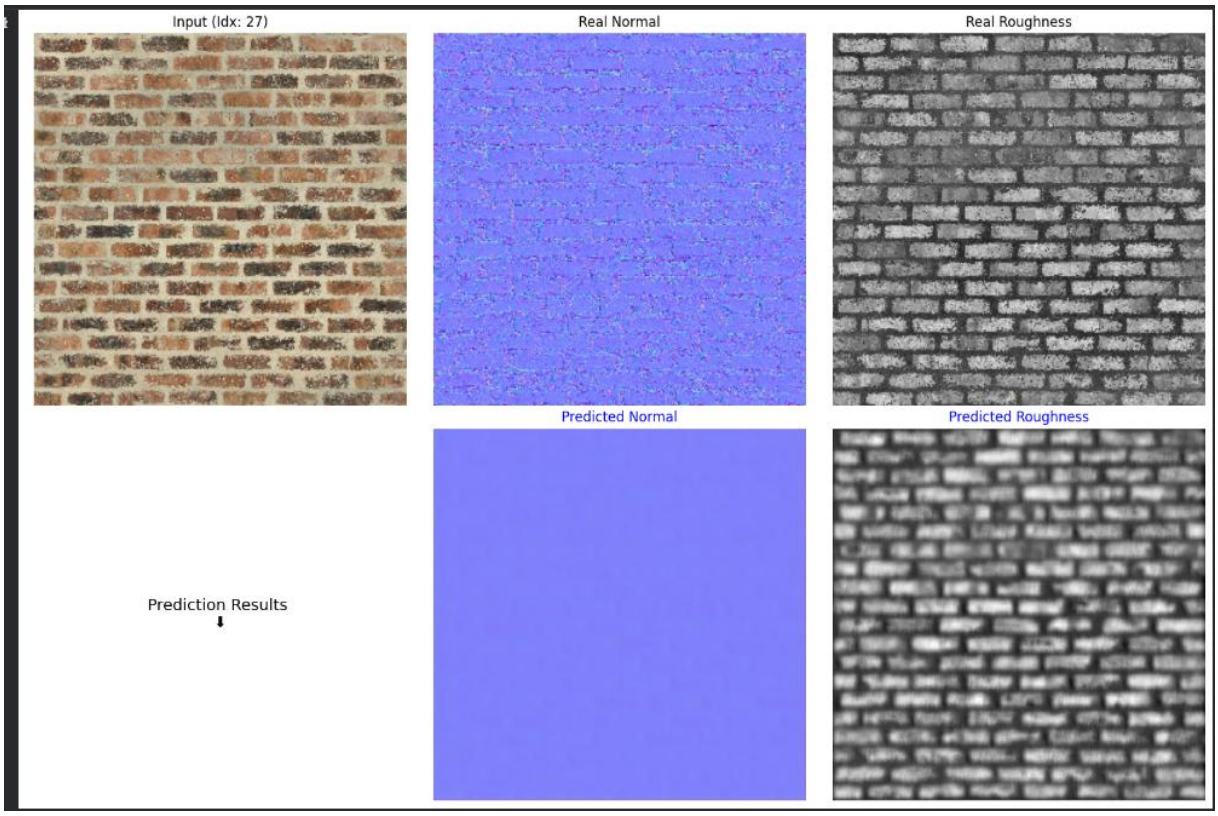


Рисунок 5.5 – Результат тренування штучної нейронної мережі

Для подолання проблеми низької деталізації та «розмиття» текстур у подальших дослідженнях було вирішено змінити парадигму оптимізації, відійшовши від виключного використання попіксельних метрик. Найбільш перспективним шляхом є інтеграція компонентів змагального навчання, трансформуючи архітектуру в умовну генеративно-змагальну мережу (cGAN, наприклад, Pix2Pix). Введення дискримінатора змусить генератор створювати правдоподібні високочастотні деталі, оскільки

«розмиті» зображення будуть легко класифіковані як фальшиві. Альтернативним або додатковим методом є застосування функції перцептивних втрат (perceptual loss), яка обчислює різницю не між кольорами пікселів, а між картами ознак, витягнутими з попередньо навченої мережі (наприклад, VGG-19), що дозволить зберегти структурну цілісність та різкість країв.

В ході виконання роботи найкращим варіантом для усунення проблем, що виникли, було обрано використання функції перцептивних втрат. Програмна реалізація включає створення екстрактора ознак (feature extractor) на основі попередньо навченої згорткової нейронної мережі VGG-19, з якої вилучено повнозв'язні шари класифікації. Вибір проміжних згорткових шарів (block1_conv2, block3_conv4, block5_conv4) дозволяє порівнювати зображення на різних рівнях абстракції: від низькорівневих геометричних примітивів до високорівневих текстурних патернів. Розроблений клас PerceptualLoss обчислює комбіновану помилку, що складається з середньоквадратичного відхилення між картами ознак (content loss) та попіксельної різниці інтенсивностей (pixel loss), причому вхідні дані попередньо денормалізуються та адаптуються за кількістю каналів для відповідності вимогам архітектури VGG.

Процес компіляції та навчання моделі модифіковано з урахуванням специфіки задачі багатоканальної регресії. Для оптимізації вагових коефіцієнтів застосовано алгоритм Adam з уточненими гіперпараметрами швидкості навчання (learning_rate=0.0002) та моменту (β=0.5), що сприяє стабільній збіжності генеративних моделей. Стратегія зважування втрат передбачає присвоєння значно вищого пріоритету (λ=10.0) помилці генерації карти нормалей порівняно з картою шорсткості, що емпірично обґрунтовано необхідністю подолання проблеми «згладжування» рельєфу та стимулювання мережі до відтворення високочастотних деталей поверхні. Тривалість тренування збільшено до 20 епох для забезпечення достатнього часу на мінімізацію складної функції перцептивних втрат.

Результат даних модифікацій виявився позитивний, а згенеровані текстури стали більш схожими на оригінали (рисунок 5.6).

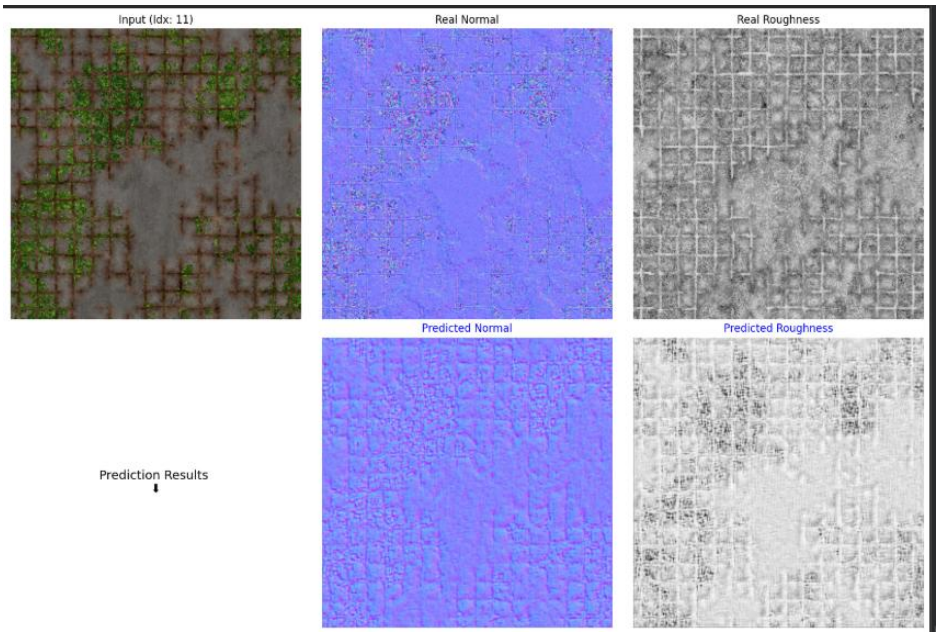


Рисунок 5.6 – Результат перевірки покращеної моделі

Натренована модель штучної нейронної мережі зберігається на Google диск у форматі .h5. Формат .h5, що базується на ієрархічному стандарті даних HDF5, є нативним контейнером для екосистеми Keras та TensorFlow, призначеним для збереження повної топології нейронної мережі, вагових коефіцієнтів та стану оптимізатора. Це робить його ідеальним інструментом на етапі розробки та навчання, оскільки він дозволяє в будь-який момент відновити тренувальний процес без втрати прогресу, проте ця універсальність накладає жорстку залежність від наявності повної бібліотеки TensorFlow для інтерпретації даних.

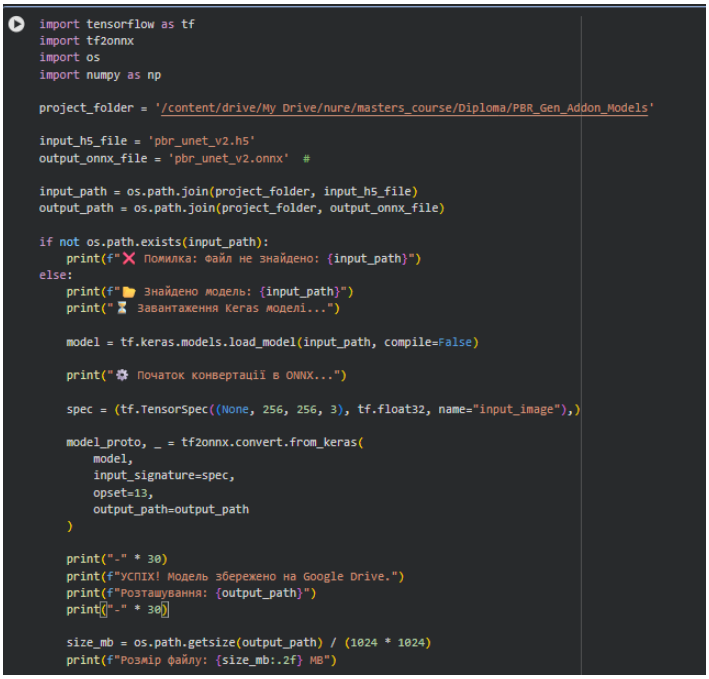
Натомість формат ONNX (Open Neural Network Exchange) був розроблений як відкритий промисловий стандарт для забезпечення інтероперабельності моделей машинного навчання, дозволяючи відокремити етап створення моделі від середовища її виконання. ONNX представляє нейромережу як оптимізований обчислювальний граф,

позбавлений надлишкових метаданих, необхідних лише для тренування, що робить його ключовим інструментом для розгортання систем на різноманітних апаратних платформах з акцентом на швидкість інференсу.

Так як аддон для Blender 3D має бути портативним та містити в собі всі необхідні ресурси для його роботи, використання бібліотеки Tensorflow не є оптимальним через її великий об’єм та складні залежності. Також сама модель збережена в форматі .h5 важить 500 мегабайт, коли така ж модель конвертована в формат .onnx займає 150 мегабайт.

З метою оптимізації програмної архітектури аддону та забезпечення його портативності було прийнято обґрунтоване рішення щодо конвертації навченої нейронної мережі з нативного формату Keras (.h5) у відкритий стандарт обміну моделями ONNX. Такий перехід дозволив відмовитися від громіздких залежностей TensorFlow на користь використання високоефективної бібліотеки комп’ютерного зору OpenCV для виконання інференсу моделі безпосередньо в середовищі Blender. Вибір на користь зв’язки ONNX та OpenCV гарантує, що кінцевий програмний продукт залишатиметься легковаговим та не вимагатиме від користувача складних налаштувань середовища виконання, зберігаючи при цьому високу точність генерації текстур.

Процес конвертації було реалізовано в рамках окремого, спеціалізованого записника Google Colab, який забезпечує автоматизований конвеєр: завантаження вихідної моделі з хмарного сховища, її трансформацію та збереження результуючого .onnx-файлу. Ізоляція цього етапу від основного процесу навчання була необхідною технічною умовою, зумовленою конфліктом версій програмних бібліотек. Оскільки інструментарій конвертації (tf2onnx) залежить від старішої версії математичної бібліотеки NumPy, несумісної з актуальним середовищем навчання TensorFlow, розмежування задач дозволило уникнути критичних помилок сумісності та забезпечити коректність структури експортованого обчислювального графа. Код конвертації моделі показано на рисунку 5.7.



```python
import tensorflow as tf
import tf2onnx
import os
import numpy as np

project_folder = '/content/drive/My Drive/nure/masters_course/Diploma/PBR_Gen_Addon_Models'

input_h5_file = 'pbr_unet_v2.h5'
output_onnx_file = 'pbr_unet_v2.onnx'  #

input_path = os.path.join(project_folder, input_h5_file)
output_path = os.path.join(project_folder, output_onnx_file)

if not os.path.exists(input_path):
    print(f"❌ Помилка: Файл не знайдено: {input_path}")
else:
    print(f"📂 Знайдено модель: {input_path}")
    print("⏳ Завантаження Keras моделі...")

    model = tf.keras.models.load_model(input_path, compile=False)

    print("⚙️ Початок конвертації в ONNX...")

    spec = (tf.TensorSpec((None, 256, 256, 3), tf.float32, name="input_image"),)

    model_proto, _ = tf2onnx.convert.from_keras(
        model,
        input_signature=spec,
        opset=13,
        output_path=output_path
    )

    print("-" * 30)
    print(f"УСПІХ! Модель збережено на Google Drive.")
    print(f"Розташування: {output_path}")
    print("-" * 30)

    size_mb = os.path.getsize(output_path) / (1024 * 1024)
    print(f"Розмір файлу: {size_mb:.2f} MB")
```

Рисунок 5.7 – Код конвертації моделей

Для перевірки коректності конвертації та відпрацювання логіки передпроцесингу вхідних даних було створено третій Colab-записник, в якому відбулося тестування роботи як збереженої моделі у форматі .h5, так і у форматі onnx.

Даний записник реалізує алгоритм валідації та візуалізації роботи нейронної мережі у форматі ONNX, використовуючи бібліотеку OpenCV для інференсу. Логіка скрипту побудована навколо трьох ключових етапів: завантаження та препроцесинг вхідного зображення з нормалізацією значень до діапазону [-1, 1], безпосереднє виконання моделі та інтелектуальний постпроцесинг вихідних тензорів. Особливістю реалізації є адаптивна функція postprocess_smart, яка автоматично аналізує розмірність отриманих даних, визначаючи формат представлення каналів (NCHW або NHWC), і за необхідності виконує транспонування осей для коректного відображення. Крім того, скрипт динамічно ідентифікує типи вихідних карт (нормалей чи шорсткості) на основі кількості каналів,

що забезпечує робастність алгоритму незалежно від порядку виходів у графі моделі.

Перехід від нативного формату Keras (.h5) до універсального стандарту ONNX супроводжувався низкою технічних викликів, пов’язаних переважно з розбіжностями у трактуванні розмірності тензорів різними фреймворками. Основна складність полягала в тому, що TensorFlow за замовчуванням оперує форматом NHWC (канали в кінці), тоді як стандарт ONNX та бібліотека OpenCV оптимізовані під формат NCHW (канали попереду). Це призводило до конфліктів при конвертації моделі інструментом tf2onnx, який автоматично додавав шари транспонування, спричиняючи помилки невідповідності розмірностей на етапі виконання. Вирішення проблеми вимагало ретельного аналізу структури вихідних даних та розробки гнучкого механізму постпроцесингу, здатного програмно нівелювати ці розбіжності без необхідності перенавчання моделі. На рисунку 5.8 можна побачити результат роботи моделі в форматі .onnx.

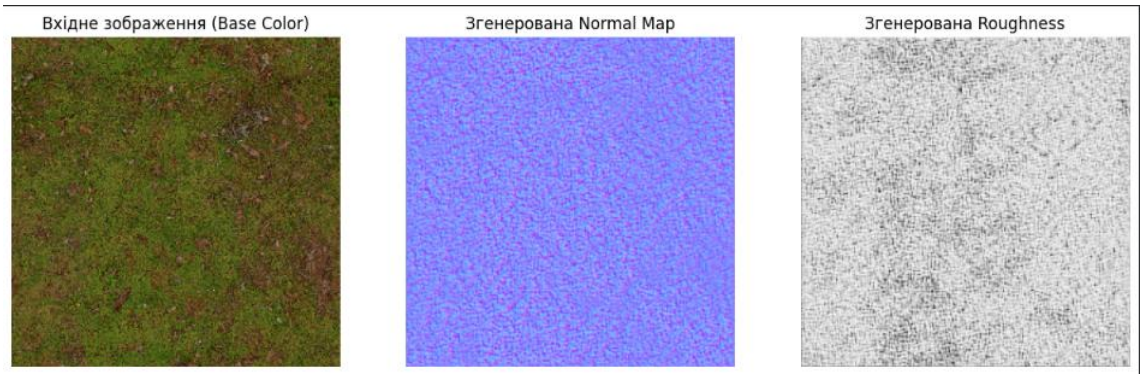


Рисунок 5.8 – Тестування ONNX-моделі

5.3 Створення аддону для Blender

Логічним продовженням та завершальним етапом практичної частини кваліфікаційної роботи є розробка клієнтського програмного забезпечення – спеціалізованого аддону для середовища тривимірного моделювання

Blender 3D. Процес проєктування даного розширення базується на дотриманні низки принципових архітектурних вимог, серед яких ключовими є портативність, інтуїтивна зрозумілість інтерфейсу та безшовна інтеграція в існуючий робочий простір редактора. Реалізація цих вимог передбачає створення автономного рішення, яке не вимагає від кінцевого користувача складного налаштування середовища Python або встановлення важковагових зовнішніх залежностей, та забезпечує нативну взаємодію з нодовою системою матеріалів Blender, ефективно інкапсулюючи складні алгоритми машинного навчання у зручний інструмент для автоматизації задач текстурування.

Особливу увагу при розробці програмного продукту було приділено забезпеченню його портативності, що є критичним фактором для зручного розгортання в різнорідних операційних середовищах без необхідності встановлення додаткового системного програмного забезпечення. Архітектура аддону спроєктована таким чином, що всі необхідні залежності, включаючи легковагові бібліотеки для інференсу нейронних мереж (зокрема, OpenCV), локалізовані безпосередньо у внутрішній структурі розширення.

Процес інсталяції в середовищі Blender 3D реалізовано через стандартний інтерфейс менеджера аддонів (рисунок 5.9). Користувач завантажує єдиний ZIP-архів, який автоматично розпаковується у директорію скриптів редактора, після чого система динамічно додає шлях до локальних модулів у змінну оточення Python sys.path. Такий підхід дозволяє уникнути конфліктів версій бібліотек із системним інтерпретатором Python та гарантує стабільну роботу інструменту одразу після його активації, значно знижуючи поріг входження для кінцевого користувача.

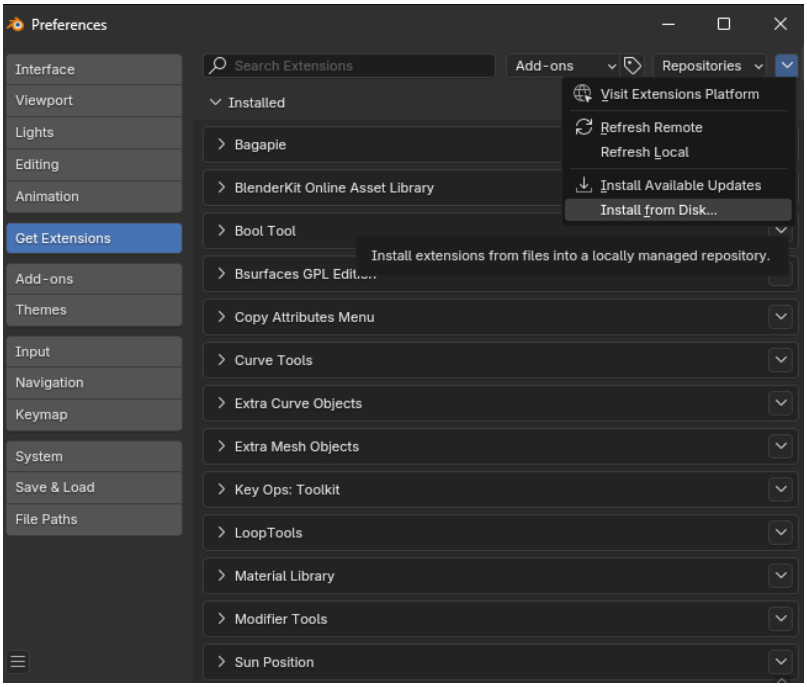


Рисунок 5.9 – Менеджер аддонів Blender 3D

Структурна організація аддона (рисунок 5.10) розроблена за принципом модульності та автономності, що є ключовою вимогою для створення портативного програмного рішення. Коренева директорія містить основний виконавчий скрипт __init__.py, який відповідає за ініціалізацію інтерфейсу користувача в Blender та керування логікою обробки даних, а також файл серіалізованої нейронної моделі pbr_unet_v2.onnx, що є інтелектуальним ядром системи. Критично важливою складовою є підкаталог modules, в якому локалізовано необхідні зовнішні залежності, зокрема бібліотека комп'ютерного зору OpenCV (cv2) та відповідні їй службові файли. Така архітектура дозволяє аддону функціонувати ізольовано від системного середовища Python користувача, усуваючи необхідність у складному налаштуванні оточення та забезпечуючи стабільну роботу інструменту одразу після встановлення.

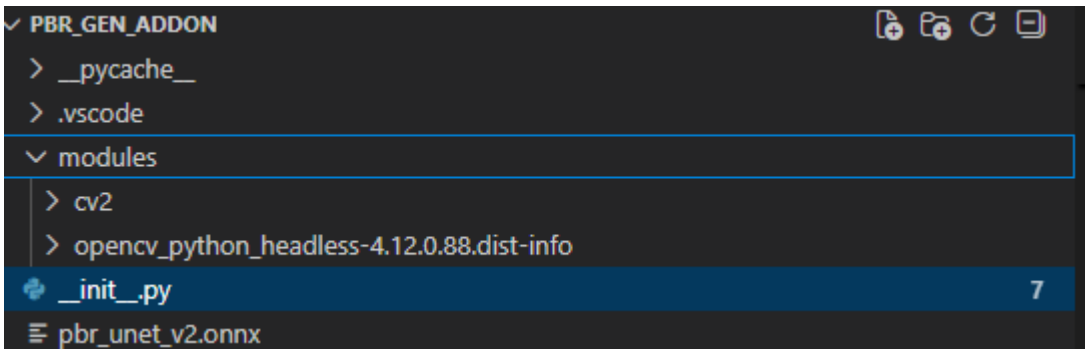


Рисунок 5.10 – Структура аддону

5.3.1 Ініціалізація середовища та імпорт залежностей

Програмний код розпочинається з налаштування середовища виконання, що є критичним для забезпечення портативності аддону. Скрипт динамічно визначає абсолютний шлях до своєї кореневої директорії та реєструє підкаталог modules у системній змінній sys.path. Ця операція дозволяє інтерпретатору Python, вбудованому в Blender, бачити та імпортувати локальні бібліотеки, такі як OpenCV, які постачаються разом з аддоном, оминаючи необхідність їх встановлення в глобальну систему користувача.

Після налаштування шляхів відбувається умовний імпорт необхідних бібліотек. Модуль numpy імпортується безпосередньо з середовища Blender, де він присутній за замовчуванням, тоді як бібліотека cv2 (OpenCV) завантажується з локальної теки. Блок try-except забезпечує відмовостійкість: у разі відсутності бібліотеки встановлюється прапорець стану, який згодом блокує виконання основного функціоналу та повідомляє користувача про помилку через інтерфейс, запобігаючи аварійному завершенню роботи програми. Програмни код, що відповідає за вищеописані дії, зображений на рисунку 5.11.

```python
import bpy
import os
import sys
import numpy as np
import math


current_dir = os.path.dirname(os.path.realpath(__file__))
modules_path = os.path.join(current_dir, "modules")

if modules_path not in sys.path:
    sys.path.append(modules_path)

try:
    import cv2
except ImportError:
    cv2 = None
    print("AI PBR Error: OpenCV not found. Please install into 'modules' folder.")
```

Рисунок 5.11 – Ініціалізація середовища та імпорт залежностей

5.3.2 Логіка обробки даних

Функція postprocess_smart відповідає за нормалізацію та форматування вихідних даних нейронної мережі. Вона автоматично аналізує розмірність вхідного тензора, визначаючи формат представлення каналів (NCHW або NHWC), і за необхідності виконує транспонування осей для приведення даних до стандарту, сумісного з Blender (висота, ширина, канали). Крім того, функція здійснює денормалізацію значень пікселів з робочого діапазону мережі [-1, 1] у стандартний діапазон [0, 1] та виконує відсікання значень (clipping) для усунення можливих артефактів перенасичення.

Функція infer_single_tile інкапсулює логіку інференсу для окремого фрагмента зображення. Вона приймає на вхід RGB-масив тайлу, виконує його попередню обробку (нормалізацію до [-1, 1] та додавання виміру батчу) і передає дані до завантаженої ONNX-моделі, через інтерфейс OpenCV DNN. Після отримання результатів функція ідентифікує карти нормалей та шорсткості на основі кількості каналів у вихідних тензорах і повертає їх у вигляді оброблених NumPy-масивів, готових до подальшої «зшивки».

Функція process_image_seamless_hq реалізує складний алгоритм безшовної генерації текстур високої роздільної здатності (High-Quality Tiling). Вона завантажує вхідне зображення з Blender, конвертує його піксельні дані у формат NumPy та ініціалізує нейронну мережу. Ключовою особливістю є механізм розбиття зображення на перекриваючі фрагменти (тайли) з використанням «ковзного вікна». Для кожного тайлу визначається «корисна» центральна область (VALID_SIZE) та буферна зона (BORDER), що дозволяє ігнорувати крайові артефакти нейромережі при зшиванні фінального зображення, забезпечуючи ідеальну безшовність результату.

5.3.2 Інтеграція з Blender API

Допоміжна функція create_blender_image відповідає за створення нових об’єктів зображень у внутрішній базі даних Blender. Вона перевіряє наявність зображення з заданим іменем, оновлює його розміри або створює нове, після чого заповнює буфер пікселів даними з результуючого NumPy-масиву. Ця функція виступає містком між математичним ядром аддону (NumPy) та графічною підсистемою Blender API.

Клас PBR_OT_GenerateSeamlessHQ є основним оператором, який запускається при взаємодії користувача з інтерфейсом. Метод execute цього класу виконує оркестрацію всього процесу: він перевіряє контекст виконання (наявність активної ноди зображення), викликає функцію обробки process_image_seamless_hq, створює нові текстурні карти та автоматично додає відповідні ноди (ShaderNodeTexImage) у редактор шейдерів, налаштовуючи їх параметри (наприклад, колірний простір «Non-Color») для коректного PBR-рендерингу.

Клас PBR_PT_PanelSeamlessHQ описує графічний інтерфейс користувача (GUI) аддону. Він реєструє нову панель у бічній панелі редактора нодів, яка містить кнопку для виклику оператора генерації.

Завершують код функції register та unregister, які відповідають за коректну інтеграцію та видалення класів аддону з системи Blender при його активації або деактивації користувачем.

5.4 Огляд роботи застосунку

Графічний інтерфейс користувача розробленого аддону інтегровано безпосередньо у робоче середовище Blender 3D, що забезпечує дотримання принципів ергономічності та безшовної взаємодії з існуючим інструментарієм редактора. Візуалізація елементів керування реалізована через спеціалізовану панель у бічній вкладці редактора шейдерів (shader editor) або у вікні властивостей матеріалу (material properties), залежно від контексту виконання задачі. Лаконічний дизайн інтерфейсу мінімізує когнітивне навантаження на оператора: основний функціонал зосереджено навколо єдиної командної кнопки запуску генерації, яка стає активною лише після успішної ініціалізації всіх залежних програмних бібліотек та валідації вхідних даних. Додатково панель оснащена системою статусних індикаторів, що в режимі реального часу інформують користувача про готовність нейронної мережі до роботи або наявність помилок у конфігурації середовища (рисунок 5.12).

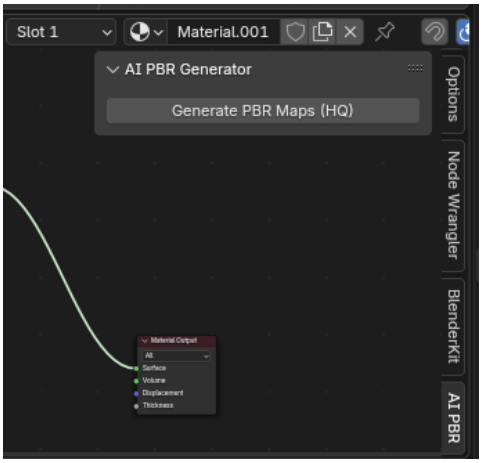


Рисунок 5.12 – Панель аддону в застосунку Blender

Процес експлуатації програмного комплексу розпочинається з підготовчого етапу, який полягає у виборі цільового об’єкта сцени та активації відповідного матеріалу. Критичною умовою коректного функціонування алгоритму є використання нодової системи шейдингу, де в якості вхідного джерела даних виступає вузол текстурного зображення (image texture node), що містить базову карту кольору. Користувач повинен явно вказати на цей вузол, зробивши його активним елементом у дереві шейдера (рисунок 5.13), що слугує сигналом для системи про вибір джерела для подальшого аналізу та обробки. Така логіка взаємодії дозволяє уникнути неоднозначності при роботі зі складними матеріалами, що складаються з множини текстурних карт.

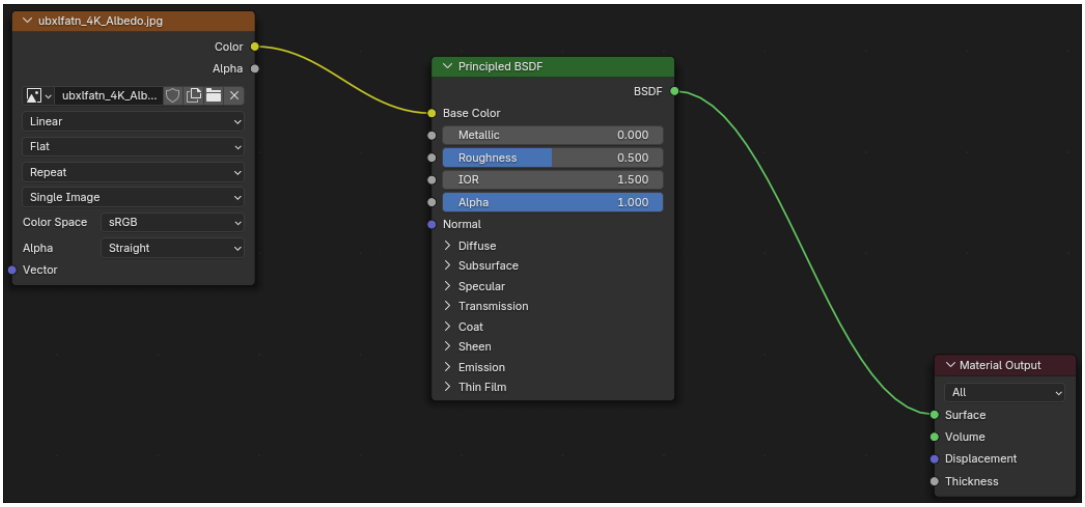


Рисунок 5.13 – Вибір вхідних даних

Після ініціації процесу генерації шляхом натискання відповідної кнопки на панелі керування, система переходить у режим фонового виконання обчислень. На цьому етапі відбувається автоматичне зчитування піксельних даних, їх попередня обробка та передача до інференс-модуля на базі OpenCV. Оскільки операція передбачення нейронною мережею є ресурсомісткою, інтерфейс Blender може короткочасно блокуватися, що є очікуваною поведінкою синхронного виконання скриптів у середовищі

Python API. По завершенню обчислень аддон автоматично створює нові об’єкти зображень для карт нормалей та шорсткості, присвоюючи їм унікальні ідентифікатори та налаштовуючи параметри колірного простору у режим «non-color data», що є обов’язковим для збереження лінійності фізичних параметрів матеріалу.

Фінальний етап роботи алгоритму полягає в автоматизованій інтеграції згенерованих результатів у структуру матеріалу. Програмний модуль самостійно створює необхідні вузли шейдера, завантажує в них отримані текстури та встановлює коректні зв’язки з входами основного шейдера Principled BSDF. Зокрема, карта нормалей підключається через проміжний вузол векторного перетворення (normal map node), що забезпечує правильну інтерпретацію геометричних відхилень поверхні. Такий рівень автоматизації дозволяє користувачеві миттєво отримати візуальний результат PBR-матеріалу без необхідності ручного налаштування параметрів, значно прискорюючи ітеративний процес створення тривимірних сцен (рисунок 5.14).

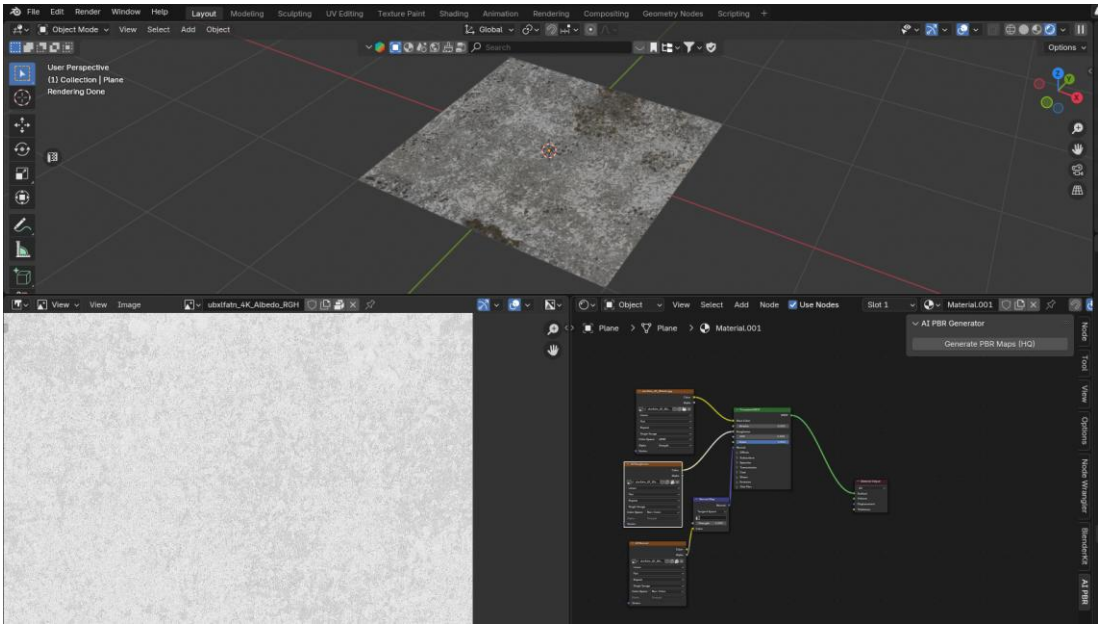


Рисунок 5.14 – Результат роботи аддону

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є розроблений програмний модуль для середовища 3D-моделювання Blender, що дозволяє користувачам автоматизувати процес створення PBR-матеріалів. Система забезпечує генерацію карт нормалей та шорсткості на основі єдиного вхідного зображення базового кольору, використовуючи методи глибокого навчання. Для розробників та дослідників створено гнучкий конвеєр у середовищі Google Colab, що дозволяє автоматизувати завантаження даних, тренування нейронної мережі, валідацію результатів та конвертацію моделей у портативні формати.

Здійснено ґрунтовний аналіз предметної галузі фотограмметрії та існуючих на ринку аналогів з метою визначення недоліків традиційних алгоритмічних рішень. Виявлено їхню обмежену здатність коректно інтерпретувати об’єм на зображеннях з рівномірним освітленням без ручного втручання. На основі цього аналізу було сформульовано задачу розробки інтелектуальної системи та визначено функціональні вимоги до кінцевого продукту, пріоритетом яких стали якість генерації та простота використання.

Для втілення поставленої задачі було спроєктовано трирівневу архітектуру застосунку, що базується на шаблоні MVC. В якості інтелектуального ядра обрано згорткову нейронну мережу типу U-Net, модифіковану для задачі перетворення зображення на зображення із застосуванням функції перцептивних втрат для підвищення деталізації. Для підготовки даних, тренування та експорту моделі розроблено спеціалізовані записники Google Colab, які використовують хмарні ресурси GPU та інтеграцію з Google Drive для зберігання датасетів.

Розроблену модель було відповідно налаштовано за допомогою експериментування з різними типами метрик, параметрів оптимізатора, підбором коректних шарів для моделі, а також роботою з вхідними даними.

На основі обраної архітектури виконана програмна реалізація клієнтської частини аддону мовою програмування Python з використанням Blender API. Для забезпечення портативності та швидкодії було реалізовано механізм інференсу на базі бібліотеки OpenCV та формату моделей ONNX, що дозволило уникнути залежності від громіздких фреймворків на кшталт TensorFlow у кінцевому продукті. Реалізовано алгоритми автоматичного препроцесингу зображень та адаптивного постпроцесингу вихідних тензорів, що вирішує проблеми сумісності форматів даних.

Перевагою розробленого аддону є його повна автономність та портативність: інструмент функціонує за принципом «plug-and-play», не вимагаючи від користувача складного налаштування середовища. Це робить технології машинного навчання доступними для широкого кола 3D-художників, дозволяючи значно прискорити рутинні процеси текстурування.

Недоліком поточної реалізації є відсутність можливості генерувати карту металічності та карти зміщення, що дещо обмежує спектр матеріалів, які можна повноцінно відтворити. Дані обмеження зумовлені обчислювальною складністю навчання на зображеннях високої роздільної здатності.

Даний програмний продукт має значні перспективи розвитку. Багатообіцяючим напрямком є перехід до архітектури умовних генеративно-змагальних мереж, що дозволить генерувати текстури з вищою частотою деталей. Також доцільно розглянути можливість використання GPU-прискорення через ONNX Runtime для пришвидшення генерації на потужних робочих станціях, що дозволить обробляти зображення високої роздільної здатності в реальному часі. Щодо монетизації, проєкт може розвиватися як freemium-рішення, де базові функції доступні безкоштовно, а генерація повного набору карт у високій якості надається за покупкою повної версії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is VFX? | visual effects software | autodesk. *www.autodesk.com*. URL: https://www.autodesk.com/industry/visual-effects (дата звернення: 24.11.2025).

2. What is AI image generation?. *cloudflare*. URL: https://www.cloudflare.com/learning/ai/ai-image-generation/ (дата звернення: 24.11.2025).

3. Mesquita L. Everything About PBR Textures And A Little More - PART 1. *ArtStation*. URL: https://www.artstation.com/blogs/luismesquita/PwEm/everything-about-pbr-textures-and-a-little-more-part-1 (дата звернення: 14.10.2025).

4. Photogrammetry software for 3D capture - Adobe Substance 3D. *Adobe: Creative, marketing and document management solutions*. URL: https://www.adobe.com/products/substance3d/apps/sampler.html (дата звернення: 24.11.2025).

5. Bounding Box Software - Materialize. *Bounding Box Software*. URL: http://www.boundingboxsoftware.com/materialize/ (дата звернення: 14.10.2025).

6. CrazyBump. *CrazyBump*. URL: http://www.crazybump.com/ (дата звернення: 25.11.2025).

7. Tiling 3D Materials, PixPlant. *PixPlant*. URL: https://www.pixplant.com/ (дата звернення: 14.10.2025).

8. GeeksforGeeks. What do you mean by Digital Image? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/computer-vision/what-do-you-mean-by-digital-image/ (дата звернення: 26.11.2025).

9. GeeksforGeeks. Image Processing Algorithms in Computer Vision - GeeksforGeeks. *GeeksforGeeks*.

URL: https://www.geeksforgeeks.org/computer-vision/image-processing-algorithms-in-computer-vision/ (дата звернення: 26.11.2025).

10. GeeksforGeeks. U-Net Architecture Explained - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/machine-learning/u-net-architecture-explained/ (дата звернення: 26.11.2025).

11. Sabbha. Understanding MAE, MSE, and RMSE: Key Metrics in Machine Learning. *DEV Community*. URL: https://dev.to/mondal_sabbha/understanding-mae-mse-and-rmse-key-metrics-in-machine-learning-4la2 (дата звернення: 26.11.2025).

12. Tsang S.-H. Brief Review – Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *Medium*. URL: https://sh-tsang.medium.com/brief-review-perceptual-losses-for-real-time-style-transfer-and-super-resolution-ac4fd2658b8 (дата звернення: 26.11.2025).

13. IBM. What Is Three-Tier Architecture? | IBM. *IBM*. URL: https://www.ibm.com/think/topics/three-tier-architecture (дата звернення: 25.11.2025).

14. MVC - Glossary | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата звернення: 25.11.2025).

15. Contributors to Wikimedia projects. Python (programming language) - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Python_(programming_language) (дата звернення: 26.11.2025).

16. Contributors to Wikimedia projects. TensorFlow - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/TensorFlow (дата звернення: 26.11.2025).

17. Contributors to Wikimedia projects. Google Colab - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Google_Colab (дата звернення: 24.10.2025).

18. Blender Python API. *Blender Documentation - blender.org*. URL: https://docs.blender.org/api/current/index.html (дата звернення: 26.11.2025).

19. Contributors to Wikimedia projects. NumPy - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/NumPy (дата звернення: 26.11.2025).

20. Contributors to Wikimedia projects. Blender (software) - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Blender_(software) (дата звернення: 26.11.2025).

21. Contributors to Wikimedia projects. Google Drive - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Google_Drive (дата звернення: 26.11.2025).

22. Contributors to Wikimedia projects. OpenCV - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/OpenCV (дата звернення: 26.11.2025).