НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 303.732.4

До захисту допущено
Завідувач кафедри ММСА
_____Оксана ТИМОЩУК
«___» _____ 2024 р.

Магістерська дисертація
на здобуття ступеня магістра
за освітньо-професійною програмою «Системний аналіз фінансового ринку»
зі спеціальності 124 «Системний аналіз»
на тему: «ГЕНЕРАТИВНІ НЕЙРОННІ МЕРЕЖІ ДЛЯ ЗАДАЧ КОМП'ЮТЕРНОГО ЗОРУ НА ПРИКЛАДІ ТЕКСТУР ДЛЯ КОМП'ЮТЕРНИХ ІГОР»

Виконав:
Студент 2 курсу, групи КА-32мп
Черкасов Євген Валерійович _____

Науковий керівник:
Доцент кафедри ММСА, доктор технічних наук
Мілявський Юрій Леонідович _____

Рецензент:
н.с. відділу прикладного нелінійного аналізу
ННК "ІПСА" КПІ ім. Ігоря Сікорського,
кандидат фізико-математичних наук
Статкевич Віталій Михайлович _____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент: _____

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність — 124 «Системний аналіз»
Освітньо-професійною програмою «Системний аналіз фінансового ринку»

ЗАТВЕРДЖУЮ
Завідувач кафедри ММСА
_____ Оксана ТИМОЩУК
«___» _____ 2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Черкасову Євгену Валерійовичу
(прізвище ім'я по батькові)

1. Тема дисертації: Генеративні нейронні мережі для задач комп'ютерного зору на прикладі текстур для комп'ютерних ігор, науковий керівник дисертації доцент, доктор технічних наук Мілявський Юрій Леонідович, затверджені наказом по університету від 07 листопада 2024 року № 5001-с.
2. Строк подання студентом дисертації: 06 грудня 2024 року.
3. Об'єкт дослідження: застосування генеративних нейронних мереж для автоматизованого створення текстур персонажів комп'ютерних ігор.
4. Предмет дослідження: моделі для вирішення задачі генерації текстур.
5. Перелік завдань, які потрібно розробити:
1) огляд предметної області;
2) відбір та аналіз інструментів машинного навчання для практичної реалізації;
3) аналіз та візуалізація дослідницьких даних;
4) розробка моделі вирішення задачі генерації текстур;
5) тестування отриманих моделей, аналіз результатів, оцінка їх якості та відбір найефективніших.
6. Перелік графічного (ілюстративного) матеріалу:
1) рисунки;
2) таблиці;
3) презентація.

7. Орієнтовний перелік публікацій.

Черкасов Є.В., Мілявський Ю.Л. Генеративні нейронні мережі для задач комп'ютерного зору на прикладі текстур для комп'ютерних ігор. Системні науки та інформатика: збірник доповідей III Всеукраїнської науково-практичної конференції «Системні науки та інформатика», 25–29 листопада 2024 року, Київ. Київ: НН ІПСА КПІ ім. Ігоря Сікорського. 2024. 9 с.

8. Консультанти розділів дисертації.

Розділ	Прізвище, ініціали та	Підпис, дата	
		завдання видав	завдання прийняв
-	-	-	-

9. Дата видачі завдання: 02 вересня 2024 року

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації (МД)	Строк виконання етапів магістерської дисертації	Примітка
1	Затвердження теми МД. Ознайомлення зі структурою МД згідно з Положенням про державну атестацію студентів НТУУ «КПІ ім. І. Сікорського»	01.09.2024-10.09.2024	виконано
2	Ознайомлення з ДСТУ 3008:2015.	11.09.2024-17.09.2024	виконано
3	Перший розділ. Огляд літературних та інформаційних джерел. Аналіз предметної області.	18.09.2024-01.10.2024	виконано
4	Другий розділ. Розробка теоретичного узагальнення методу.	02.10.2024-15.10.2024	виконано
5	Третій розділ. Розробка програмного забезпечення.	16.10.2024-29.10.2024	виконано
6	Третій розділ. Робота над практичним розділом магістерської дисертації.	30.10.2024-12.11.2024	виконано
7	Четвертий розділ. Стартап-проект.	13.11.2024-19.11.2024	виконано

Студент _____ Євген ЧЕРКАСОВ

Науковий керівник дисертації _____ Юрій МІЛЯВСЬКИЙ

РЕФЕРАТ

Магістерська дисертація: 137 с., 26 рис., 24 табл., 99 джерел, 2 додатки.

ГЕНЕРАТИВНІ НЕЙРОННІ МЕРЕЖІ, КОМП'ЮТЕРНИЙ ЗІР, ТЕКСТУРИ, MINECRAFT, GAN, CONDITIONAL GAN, WASSERSTEIN GAN, DIFFUSION MODELS

Об'єкт дослідження – процес генерації текстур для персонажів комп'ютерних ігор.

Мета роботи – розробка та реалізація методів генеративних нейронних мереж для створення текстур персонажів комп'ютерних ігор.

Методи дослідження – аналіз наукової літератури, веб-скрапінг, експериментальне моделювання, порівняльний аналіз архітектур нейронних мереж, візуальна та кількісна оцінка якості згенерованих текстур.

У роботі проведено аналіз сучасних підходів до генерації текстур, розроблено та протестовано три архітектури генеративних моделей: GAN, C-GAN-WP та дифузійну модель. Створено один з найбільших наборів даних текстур Minecraft (773,971 зразків). Розроблено програмний продукт на мові Python з використанням фреймворків PyTorch та TensorFlow для генерації текстур персонажів.

Результати роботи впроваджено у формі GAN моделі для автоматизованої генерації текстур. Розроблені моделі можуть бути використані розробниками ігор та творцями модифікацій для швидкого створення якісних текстур персонажів.

ABSTRACT

Master's thesis: 137 p., 26 fig., 24 tabl., 99 references, 2 appendices.

GENERATIVE NEURAL NETWORKS, COMPUTER VISION, TEXTURES, MINECRAFT, GAN, CONDITIONAL GAN, WASSERSTEIN GAN, DIFFUSION MODELS

The object of research is the process of generating textures for video game characters.

The aim of the work is to develop and implement generative neural network methods for creating video game character textures.

Research methods include scientific literature analysis, web scraping, experimental modeling, comparative analysis of neural network architectures, visual and quantitative evaluation of generated textures quality.

The paper analyzes modern approaches to texture generation, develops and tests three generative model architectures: GAN, C-GAN-WP, and diffusion model. One of the largest Minecraft texture datasets (773,971 samples) was created. A software product was developed in Python using PyTorch and TensorFlow frameworks for character texture generation.

The results are implemented as a GAN model for automated texture generation. The developed models can be used by game developers and mod creators for rapid creation of high-quality character textures.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

GAN – генеративні змагальні мережі (Generative Adversarial Networks)

C-GAN – умовні генеративні змагальні мережі (Conditional Generative Adversarial Networks)

GAN-WP – генеративні змагальні мережі з штрафом градієнта (Wasserstein GAN with Gradient Penalty)

CNN – згорткова нейронна мережа (Convolutional Neural Network)

Diffusion – моделі дифузії для генерації зображень

ML – машинне навчання (Machine Learning)

SaaS – програмне забезпечення як послуга (Software as a Service)

MVP – мінімально життєздатний продукт (Minimum Viable Product)

R&D – дослідження та розробка (Research and Development)

API – інтерфейс прикладного програмування (Application Programming Interface)

ВСТУП

В останні роки індустрія комп'ютерних ігор демонструє стрімке зростання, що супроводжується підвищенням вимог до якості та різноманітності ігрового контенту. Створення унікальних текстур для ігрових персонажів традиційно вимагає значних часових та людських ресурсів. Особливо це стосується ігор з активною спільнотою модифікацій, таких як Minecraft, де гравці постійно шукають нові способи персоналізації своїх персонажів. Використання генеративних нейронних мереж для автоматизованого створення текстур відкриває нові можливості для розробників та користувачів, дозволяючи генерувати різноманітний високоякісний контент при значному скороченні ресурсних витрат.

Метою дослідження є та реалізація методів генеративних нейронних мереж для створення текстур персонажів комп'ютерних ігор. Для досягнення цієї мети були поставлені наступні завдання: проаналізувати існуючі підходи до генерації текстур за допомогою нейронних мереж; зібрати та підготувати набір даних текстур персонажів Minecraft; реалізувати та порівняти різні архітектури генеративних моделей; провести експериментальні дослідження та оцінити якість згенерованих текстур; також створити план власного стартап-проекту на основі розробки.

Об'єктом дослідження є процес генерації текстур для персонажів комп'ютерних ігор. Предметом дослідження виступають методи та алгоритми генеративних нейронних мереж для створення двовимірних текстур, зокрема архітектури GAN, C-GAN-WP та їх модифікації.

У ході дослідження були застосовані такі методи як аналіз наукової літератури та технічної документації, веб-скрапінг для збору даних, експериментальне моделювання, порівняльний аналіз результатів різних архітектур нейронних мереж, візуальна та кількісна оцінка якості

згенерованих текстур. Для реалізації моделей використовувались сучасні фреймворки машинного навчання та обробки зображень.

Практичне значення отриманих результатів полягає в розробці підходу до автоматизованої генерації текстур для ігрових персонажів, який може бути використаний розробниками ігор та творцями модифікацій. Реалізовані моделі дозволяють генерувати різноманітні високоякісні текстури у стилі Minecraft, що відповідають встановленим вимогам до розмірів та стилістики. Результати дослідження також можуть бути корисними для подальшого розвитку методів генеративного штучного інтелекту в ігровій індустрії та суміжних галузях комп'ютерної графіки.

Магістерська дисертація складається із вступу, чотирьох розділів, висновків, переліку джерел посилання та двох додатків.

РОЗДІЛ 1 АНАЛІЗ ПОПЕРЕДНІХ ДОСЛІДЖЕНЬ У СФЕРІ ГЕНЕРАЦІЇ ТЕКСТУР

У сучасному світі розробки комп'ютерних ігор та програмного забезпечення все більшої актуальності набуває автоматизована генерація текстур за допомогою штучного інтелекту. Особливу увагу дослідники приділяють використанню генеративних змагальних мереж (GAN) та їх модифікацій для створення високоякісного візуального контенту.

Значний внесок у розвиток генеративних моделей для обробки зображень зробили дослідники MedGAN [1]. Хоча їхня робота зосереджена на медичних зображеннях, запропоновані підходи мають широке застосування для генерації текстур. Автори представили нову архітектуру CasNet, яка покращує чіткість згенерованих зображень через поступове уточнення за допомогою пар енкодер-декодер, а також впровадили комбінацію змагальних та незмагальних функцій втрат для покращення якості генерації.

Дослідження потенціалу AC-GAN для класифікації зображень [2] демонструє можливість створення універсальної архітектури, яка одночасно виконує генерацію та класифікацію. Автори запропонували модифікації стандартної структури AC-GAN, включаючи зміну схеми семплювання латентного простору генератора та використання Wasserstein loss з градієнтним штрафом для стабілізації навчання, що особливо важливо при роботі з обмеженими наборами даних.

Значний прогрес у розробці методів передачі текстур продемонстровано в дослідженні Multi-Texture GAN [3]. Автори розробили багатомасштабний підхід до передачі текстур, який дозволяє зберігати детальну інформацію про текстури на різних рівнях деталізації. Запропонований метод здійснює співставлення текстурних особливостей у нейронному просторі, що забезпечує кращу передачу семантично пов'язаних характеристик зображення.

Аналіз існуючих досліджень демонструє активний розвиток методів генерації текстур з використанням глибинного навчання. Основними тенденціями є вдосконалення архітектур GAN, розробка механізмів контролю генерації та впровадження багатомасштабних підходів до обробки текстур. Особливу увагу дослідники приділяють стабільності навчання моделей та якості згенерованих зображень, що є критично важливим для практичного застосування в індустрії комп'ютерних ігор.

1.1 Огляд досліджень з генерації текстур для комп'ютерних ігор

Генерація текстур та ігрового контенту є однією з ключових проблем у розробці сучасних комп'ютерних ігор, що вимагає значних ресурсів та часу. З розвитком штучного інтелекту та генеративних моделей з'являються нові можливості для автоматизації цього процесу, що привертає увагу як академічної спільноти, так і індустрії.

Дослідження 2024 року щодо дистиляції генераторів ігрового контенту демонструє значний прогрес у вирішенні проблем керованості та обмеженості навчальних даних [4]. Автори пропонують інноваційний підхід, який полягає у перетворенні конструктивного алгоритму PCG у керовану PCGML модель з використанням великих мовних моделей для розмітки даних. Це дозволяє створювати контент, що відповідає оригінальному алгоритму, але з додатковою можливістю текстового керування.

Значний внесок у розвиток автоматизації створення ігрового контенту зробило дослідження Cine-AI [5], яке представляє інструментарій для генерації внутрішньоігрових відео у стилі відомих режисерів. Хоча це дослідження фокусується на кінематографічному аспекті, воно демонструє важливість збереження стилістичної єдності та внутрішньої узгодженості при автоматичній генерації контенту.

У контексті конкуренції між штучним інтелектом та людською творчістю, дослідження 2024 року [6] розглядає стратегічні аспекти створення контенту в еру генеративного ШІ. Автори аналізують динамічну конкуренцію між ШІ та людиною-творцем, підкреслюючи, що хоча ШІ має перевагу в швидкості та вартості генерації, його якість залежить від наявності людського контенту для навчання.

Аналіз представлених досліджень свідчить про активний розвиток технологій автоматичної генерації ігрового контенту, де особлива увага приділяється балансу між автоматизацією та збереженням творчого контролю. Спостерігається тенденція до створення більш керованих та адаптивних систем, які можуть враховувати специфічні вимоги та стилістичні особливості, що є критично важливим для генерації текстур та інших елементів ігрового контенту.

1.1.1 Аналіз публікацій з процедурної генерації текстур

Процедурна генерація текстур є важливим напрямком досліджень у комп'ютерній графіці та розробці ігор, що дозволяє автоматизувати створення візуальних елементів та значно скоротити час розробки. З розвитком глибинного навчання та генеративних моделей з'явились нові можливості для автоматичної генерації високоякісних текстур, що привернуло значну увагу дослідницької спільноти.

Значний прорив у процедурній генерації текстур було досягнуто завдяки розробці просторових генеративно-змагальних мереж (Spatial GAN). Як показано в дослідженні [7], розширення вхідного простору шуму від одиничного вектора до просторового тензора дозволило створити архітектуру, що особливо добре підходить для синтезу текстур. Автори продемонстрували, що їхній метод забезпечує високу якість згенерованих текстур, відмінну

масштабованість щодо розміру вихідної текстури та швидку генерацію в реальному часі. Крім того, модель здатна об'єднувати кілька різних вхідних зображень у складні текстури.

Нові дослідження в сфері показують стрімкий розвиток цього напрямку та перехід від традиційних алгоритмічних методів до підходів, заснованих на глибинному навчанні. Особливо перспективним є використання GAN мереж, які демонструють найкращі результати з точки зору якості та різноманітності згенерованих текстур. При цьому важливим аспектом залишається баланс між якістю генерації та обчислювальною ефективністю, що особливо актуально для застосування в ігровій індустрії.

1.1.2 Дослідження застосування AI для створення ігрового контенту

Застосування штучного інтелекту для створення ігрового контенту є одним з найперспективніших напрямків розвитку ігрової індустрії. Особливу увагу привертають методи генеративного штучного інтелекту, які дозволяють автоматизувати створення різноманітних ігрових активів, включаючи текстури, моделі та інші візуальні елементи.

Важливий внесок у розвиток AI-генерації текстур зробило дослідження просторових генеративних змагальних мереж (SGAN) [8]. Автори запропонували інноваційний підхід, розширивши простір розподілу вхідного шуму від одиничного вектора до просторового тензора. Це дозволило створити архітектуру, яка особливо ефективна для синтезу текстур і відрізняється високою якістю генерації, масштабованістю щодо розміру вихідної текстури та можливістю об'єднання кількох різних вихідних зображень у складні текстури.

Значний прогрес у застосуванні AI для створення ігрового контенту демонструє дослідження багато-GAN архітектури [9]. Запропонований метод

використовує чотири прогресивні генератори та дискримінатори, що навчаються end-to-end способом. Особливістю підходу є використання LBP-based функції втрат для забезпечення деталізованих текстур та покращення глобальної структурної узгодженості зображення, що особливо важливо для ігрового контенту.

Аналіз досліджень показує, що застосування AI для створення ігрового контенту демонструє значний потенціал, особливо у контексті генерації текстур. Сучасні підходи дозволяють досягти високої якості генерації при збереженні обчислювальної ефективності. Важливими тенденціями є використання багатоетапних архітектур, спеціалізованих функцій втрат та механізмів, що забезпечують контроль над процесом генерації, що робить ці методи особливо привабливими для ігрової індустрії.

1.2 Дослідження застосування GAN для генерації текстур

Генеративно-змагальні мережі (GAN) стали потужним інструментом для створення різноманітного контенту, включаючи текстури для комп'ютерних ігор. Дослідження їх застосування демонструє значний потенціал у автоматизації процесів генерації графічних елементів та відкриває нові можливості для розробки ігрового контенту.

Згідно з дослідженням процедурної генерації текстур [10], традиційні алгоритми, такі як Wave Function Collapse (WFC), показують обмежену ефективність при створенні складних текстур. Дослідники виявили, що більш гнучкі підходи, засновані на машинному навчанні, демонструють кращі результати у відтворенні складних візуальних патернів та забезпеченні статистичної подібності згенерованих текстур до оригінальних зразків.

Аналіз застосування глибокого навчання для процедурної генерації контенту [11] показує, що методи глибокого навчання можуть ефективно

доповнювати або замінювати традиційні підходи до генерації ігрового контенту. Дослідники відзначають, що комбінування глибокого навчання з класичними методами процедурної генерації дозволяє досягти кращих результатів у створенні різноманітних типів контенту, включаючи текстури.

На основі проаналізованих досліджень можна зробити висновок, що застосування GAN для генерації текстур представляє собою перспективний напрямок, який дозволяє подолати обмеження традиційних алгоритмів. Інтеграція методів глибокого навчання з існуючими підходами до процедурної генерації створює потужний інструментарій для автоматизованого створення високоякісного ігрового контенту, що відповідає сучасним вимогам індустрії.

1.2.1 Аналіз робіт з базовими архітектурами GAN

Генеративні змагальні мережі (GAN) стали революційним підходом у сфері генеративного моделювання, демонструючи значний потенціал у різноманітних задачах комп'ютерного зору. Базові архітектури GAN заклали фундамент для подальшого розвитку генеративних моделей та їх застосування у практичних задачах.

Дослідження 2019 року, присвячене моделюванню табличних даних за допомогою умовних GAN, продемонструвало ефективність застосування архітектури TGAN для роботи зі змішаними типами даних [12]. Автори розробили комплексний підхід до оцінки продуктивності моделі, використовуючи 15 наборів даних різної природи, що підтвердило перевагу GAN над традиційними байєсівськими методами.

У дослідженні 2021 року було запропоновано інноваційний метод трансферного навчання для умовних GAN, який дозволяє ефективно передавати знання між класами [13]. Ключовою особливістю підходу стало використання умовної пакетної нормалізації для передачі класово-

специфічної інформації, що забезпечило лінійне зростання параметрів відносно кількості нових класів.

Дослідження 2022 року представило архітектуру AC-GAN для аналізу фінансових часових рядів [14]. Автори вдосконалили традиційний підхід CGAN, додавши автоенкодерну складову, що дозволило краще зберігати важливі характеристики даних при генерації та покращило якість прогнозування ринкових трендів.

Аналіз досліджень базових архітектур GAN демонструє еволюцію підходів від простих генеративних моделей до складних спеціалізованих архітектур. Спостерігається тенденція до розробки модифікацій, що враховують специфіку конкретних задач та типів даних, при цьому зберігаючи основні принципи змагального навчання. Важливим напрямком розвитку стало впровадження механізмів трансферного навчання та покращення стабільності тренування моделей.

1.2.2 Огляд модифікацій GAN для специфічних задач

Модифікації генеративно-змагальних мереж (GAN) для специфічних задач є важливим напрямком досліджень, що дозволяє адаптувати базову архітектуру під конкретні вимоги та обмеження. Такі модифікації спрямовані на покращення контрольованості генерації, підвищення якості результатів та розширення можливостей застосування GAN у різних доменах.

Дослідники з галузі контрольованої генерації тексту запропонували фреймворк InstructCTG [15], який дозволяє інкорпорувати різні обмеження шляхом використання природномовних інструкцій та демонстрацій. Особливістю цього підходу є можливість екстракції прихованих обмежень з природних текстів за допомогою комбінації NLP-інструментів та евристик, що

дозволяє створювати слабко розмічені навчальні дані без додаткової ручної анотації.

Значний прогрес у контрольованій генерації було досягнуто завдяки методу вирівнювання атрибутів [16], який пропонує гнучкий спосіб керування генерацією тексту через узгодження розділених атрибутивних представлень. На відміну від попередніх підходів, які використовували дискримінатор для модифікації розподілу на рівні токенів, цей метод навчає функцію вирівнювання для спрямування попередньо навченої моделі без зміни її параметрів.

Новітній підхід до контрольованої генерації представлено у дослідженні prompt tuning [17], де автори демонструють можливість керування генерацією через навчання вкладень промптів з використанням малої мовної моделі як дискримінатора. Важливою перевагою цього методу є його ефективність при обмеженому наборі навчальних даних, що робить його практичним для реальних застосувань.

Аналіз розглянутих модифікацій GAN свідчить про активний розвиток методів контрольованої генерації, які дозволяють досягти балансу між якістю генерації та можливістю керування нею. Особливо важливим є тренд на розробку методів, що не потребують значних обчислювальних ресурсів та великих наборів даних для навчання, що робить їх доступними для широкого кола практичних застосувань.

1.3 Дослідження умовної генерації текстур

Умовна генерація текстур є важливим напрямком досліджень у сфері генеративних нейронних мереж, що дозволяє створювати текстури із заданими характеристиками та властивостями. Цей підхід має особливе значення для

розробки комп'ютерних ігор, де часто виникає потреба у створенні великої кількості текстур із певними параметрами та стилістичними особливостями.

Згідно з дослідженням у сфері процедурної генерації контенту для відеоігор [18], глибинне навчання відкрило нові можливості для умовної генерації текстур. Автори відзначають, що традиційні методи, такі як пошукові, граматичні та rule-based підходи, можуть бути ефективно доповнені або замінені глибинними нейронними мережами. Особливу увагу приділено гібридним підходам, де глибинне навчання комбінується з класичними методами процедурної генерації, що дозволяє досягти кращого контролю над процесом генерації та якістю вихідних текстур.

Аналіз існуючих досліджень демонструє, що умовна генерація текстур є перспективним напрямком, який продовжує активно розвиватися. Поєднання традиційних методів із сучасними підходами глибинного навчання створює потужний інструментарій для розробників ігор та дозволяє автоматизувати процес створення високоякісних текстур із заданими параметрами. Це відкриває нові можливості для оптимізації процесу розробки ігрового контенту та підвищення його якості.

1.3.1 Аналіз досліджень Conditional GAN

Умовні генеративні змагальні мережі (Conditional GAN) представляють важливий напрямок досліджень у сфері генеративного глибокого навчання, що дозволяє контролювати процес генерації через додаткові вхідні умови. Ця технологія знаходить широке застосування у різноманітних задачах комп'ютерного зору та обробки зображень.

Дослідження 2019 року щодо семантичного доповнення та покращення зображень демонструє ефективність використання умовних GAN для відновлення пошкоджених ділянок зображень [19]. Автори пропонують

архітектуру, що поєднує переваги Wasserstein GAN із механізмами умовної генерації для створення реалістичних заповнень пошкоджених областей, досягаючи покращення показників PSNR та SSIM на 2.45% та 4% відповідно.

Значний прогрес у застосуванні умовних GAN було продемонстровано в роботі 2020 року, присвяченій конверсії співочого голосу [20]. Дослідники розробили фреймворк на основі VAW-GAN, який дозволяє здійснювати перетворення голосу без необхідності використання паралельних навчальних даних, що має важливе практичне значення для реальних застосувань.

У дослідженні Wasserstein дивергенції для GAN автори пропонують новий підхід до оптимізації умовних генеративних мереж [21]. Запропонована метрика WGAN-div не потребує виконання умови k-Ліпшиця, що значно спрощує процес навчання та підвищує стабільність моделі при збереженні якості генерації.

Практичне застосування умовних GAN для синтезу мультиспектральних супутникових знімків представлено в роботі 2020 року [22]. Автори розробили архітектуру S2A, що поєднує механізми просторово-спектральної уваги з Wasserstein GAN, демонструючи високу ефективність у синтезі додаткових спектральних каналів для різних типів місцевості.

Аналіз досліджень Conditional GAN показує стрімкий розвиток цього напрямку, особливо в контексті специфічних прикладних задач. Основними тенденціями є поєднання умовної генерації з механізмами уваги, використання модифікованих метрик відстані та розробка архітектур, що не потребують паралельних навчальних даних. Ці досягнення створюють міцний фундамент для подальшого розвитку технологій керованої генерації зображень.

1.3.2 Огляд методів контролю генерації

Методи контролю генерації є критично важливим аспектом у розробці генеративних нейронних мереж, особливо коли йдеться про створення текстур для комп'ютерних ігор. Ці методи дозволяють керувати процесом генерації та отримувати результати з бажаними характеристиками, що є особливо важливим для практичного застосування в ігровій індустрії.

Згідно з дослідженням, опублікованим у березні 2023 року, значного прогресу досягнуто у контролі генерації 3D-текстур завдяки використанню трипланових представлень та умовних дифузійних моделей [23]. Дослідники розробили двоетапний конвеєр, який включає VAE для вивчення латентних представлень текстурованих мешів та умовну дифузійну модель для генерації трипланових характеристик, що дозволяє здійснювати як умовну, так і безумовну генерацію високоякісних текстурованих 3D-мешів.

Дослідження CTGAN, опубліковане в лютому 2024 року, представляє новий підхід до контролю генерації текстур через семантичне керування [24]. Запропонована модель використовує розділену природу StyleGAN для точного маніпулювання вхідними латентними кодами, що забезпечує явний контроль над стилем та структурою генерованих текстур. Архітектура кодувальника з поступовим уточненням підвищує контроль над структурою текстур через вхідну сегментацію.

Важливий внесок у розвиток методів контролю генерації зроблено в дослідженні 2021 року, де представлено двопотокову мережу для відновлення зображень [25]. Автори запропонували модулі Bi-GFF та CFA для покращення глобальної узгодженості через обмін та комбінування структурної та текстурної інформації, що демонструє ефективність контрольованої генерації при відновленні пошкоджених ділянок зображень.

Аналіз представлених досліджень показує еволюцію методів контролю генерації від базових підходів до складних багатокомпонентних систем. Сучасні методи дозволяють здійснювати точний контроль над різними аспектами генерації, включаючи структуру, стиль та семантичні характеристики текстур. Це відкриває нові можливості для практичного застосування генеративних моделей у розробці комп'ютерних ігор та інших галузях цифрового контенту.

1.4 Огляд досліджень Wasserstein GAN

Wasserstein GAN (WGAN) представляє собою важливе вдосконалення архітектури генеративних змагальних мереж, що вирішує ключові проблеми стабільності навчання та якості генерації. Цей підхід використовує метрику Вассерштейна для оцінки відстані між реальним та згенерованим розподілами даних, що забезпечує більш надійний градієнт для навчання.

Згідно з дослідженням у сфері відновлення та покращення зображень, WGAN демонструє значні переваги порівняно з класичними GAN у задачах генерації візуального контенту [26]. Дослідники відзначають, що використання метрики Вассерштейна забезпечує більш стабільне навчання та зменшує ймовірність колапсу режимів.

Аналіз застосування дифузійних моделей у обробці природної мови показав, що WGAN може ефективно поєднуватися з іншими архітектурами, зокрема з трансформерами [27]. Це дозволяє досягти кращих результатів у задачах умовної генерації та контролю вихідного контенту.

Дослідження фундаментальних принципів проектування генеративних моделей виявило, що WGAN забезпечує більш надійну оцінку градієнтів порівняно з традиційними підходами [28]. Автори підкреслюють важливість

правильного вибору гіперпараметрів та архітектури для досягнення оптимальних результатів.

Комплексний огляд методів та застосувань показав, що WGAN особливо ефективний у задачах, де важлива якість та різноманітність згенерованих зразків [29]. Дослідники відзначають, що додавання градієнтного штрафу (gradient penalty) додатково покращує стабільність навчання.

Аналіз застосування у комп'ютерному зорі демонструє, що WGAN успішно використовується для генерації високоякісних зображень та текстур [30]. Особливо відзначається здатність моделі зберігати дрібні деталі та текстурні особливості, що критично важливо для створення ігрового контенту.

Підсумовуючи, WGAN залишається одним з найбільш ефективних підходів до генерації візуального контенту, особливо в контексті створення текстур для комп'ютерних ігор. Основними перевагами є стабільність навчання, якість генерації та можливість точного контролю над вихідними результатами. Подальші дослідження спрямовані на оптимізацію обчислювальної ефективності та покращення масштабованості моделі.

1.4.1 Аналіз публікацій з теорії Wasserstein GAN

Теорія Wasserstein GAN (WGAN) представляє собою важливий напрямок досліджень у сфері генеративних змагальних мереж, що спрямований на подолання проблем стабільності та конвергенції, притаманних класичним GAN. Ця методологія базується на використанні метрики Вассерштейна для оцінки відстані між реальним та згенерованим розподілами даних.

Фундаментальне дослідження регуляризації WGAN, опубліковане у 2018 році [31], розглядає проблему накладання умови Ліпшиця на функції, що

моделюються нейронною мережею. Автори пропонують теоретичне обґрунтування використання слабшого регуляризаційного члена для забезпечення умови Ліпшиця, що підтверджується експериментальними результатами на тестових наборах даних.

Дослідження 2018 року щодо виявлення аномалій за допомогою WGAN [32] демонструє практичне застосування цієї архітектури для аналізу часових рядів. Автори використовують WGAN для вивчення розподілу нормальних даних та пропонують комбінацію стекового енкодера з генератором для ефективного виявлення аномалій, досягаючи високих результатів на наборі даних MNIST.

Значний внесок у розвиток теорії WGAN зроблено дослідженням Banach Wasserstein GAN [33], яке узагальнює теорію WGAN з градієнтним штрафом на простори Банаха. Автори демонструють, як вибір відповідної норми може покращити якість генерації зображень, що підтверджується експериментами на наборах даних CIFAR-10 та CelebA.

Аналіз публікацій з теорії Wasserstein GAN демонструє еволюцію підходу від базової концепції до складних модифікацій, що враховують специфіку різних просторів та метрик. Дослідження показують, що правильний вибір регуляризації та метрики відстані має критичне значення для ефективності моделі, а теоретичні вдосконалення призводять до покращення практичних результатів у різноманітних задачах комп'ютерного зору.

1.4.2 Дослідження практичного застосування

Практичне застосування генеративних моделей, зокрема Wasserstein GAN та споріднених архітектур, демонструє значний потенціал у різноманітних сферах комп'ютерного зору та генерації зображень. Важливо

розглянути останні дослідження та практичні реалізації для розуміння можливостей та обмежень цих технологій.

Згідно з дослідженням 2024 року, адаптивно керовані дифузійні моделі демонструють значний прогрес у контрольованій генерації зображень [34]. Дослідники розробили framework AC-Diff, який дозволяє автоматично контролювати не лише результат генерації, але й сам процес, включаючи кількість кроків та параметри генерації.

Важливим аспектом практичного застосування є оптимізація моделей для мобільних пристроїв. Дослідження 2023 року показало можливість розгортання великих дифузійних моделей на мобільних пристроях з використанням TensorFlow Lite [35]. Результати демонструють досягнення часу генерації менше 7 секунд для зображення 512x512 на Android-пристроях.

У медичній сфері генеративні моделі знаходять широке застосування, про що свідчить комплексний огляд 2023 року [36]. Дослідники відзначають особливу ефективність дифузійних моделей у задачах медичної візуалізації, незважаючи на обчислювальні витрати.

Цікаве дослідження 2024 року розкриває феномен узгодженості дифузійних моделей [37]. Автори виявили, що моделі з різною ініціалізацією або архітектурою можуть генерувати дуже схожі результати при однакових вхідних шумах, що відкриває нові можливості для оптимізації навчання.

Останній огляд 2024 року підкреслює універсальність генеративних моделей у різних галузях застосування [38]. Дослідники відзначають особливу ефективність у комп'ютерному зорі, аудіо, навчанні з підкріпленням та обчислювальній біології, наголошуючи на необхідності подальшого розвитку теоретичної бази.

Аналіз практичного застосування генеративних моделей демонструє їх значний потенціал та універсальність у різних сферах. Особливо важливими є досягнення в оптимізації обчислювальної ефективності та контролю процесу генерації, що відкриває нові можливості для практичного впровадження цих технологій у реальних застосуваннях.

1.5 Аналіз досліджень дифузійних моделей

Дифузійні моделі стали одним з найбільш перспективних напрямків у сфері генеративного моделювання, демонструючи вражаючі результати в різноманітних задачах комп'ютерного зору. Їх теоретичне обґрунтування та практичне застосування привертають значну увагу дослідницької спільноти, що відображається у зростаючій кількості публікацій з цієї тематики.

Згідно з дослідженням 2022 року, дифузійні моделі можна розглядати як особливий випадок марковських ієрархічних варіаційних автоенкодерів [39]. Автори демонструють, що оптимізація моделі зводиться до навчання нейронної мережі передбачати один з трьох потенційних параметрів: вихідне зображення з будь-якого рівня шуму, вихідний шум з зашумленого входу або функцію оцінки зашумленого входу на довільному рівні шуму.

Важливий внесок у розуміння властивостей генералізації дифузійних моделей зроблено в роботі 2023 року [40]. Дослідники встановили теоретичні оцінки розриву генералізації, що розвивається паралельно з динамікою навчання моделей, заснованих на оцінці градієнта. Вони показали, що помилка генералізації має поліноміальну залежність від розміру вибірки та ємності моделі.

У дослідженні 2024 року вчені зосередились на вивченні здатності дифузійних моделей до факторизації та композиції [41]. Експерименти на умовних моделях DDPM показали, що вони здатні вивчати факторизовані, але не повністю неперервні многовидні представлення для кодування неперервних ознак варіації даних.

Значний прогрес у розумінні особливостей навчання ознак у дифузійних моделях представлено в роботі, опублікованій наприкінці 2024 року [42]. Автори демонструють, що завдяки цілі денойзингу дифузійні моделі

навчаються більш збалансованим та всеохоплюючим представленням даних порівняно з традиційними класифікаційними моделями.

Додаткове дослідження 2024 року зосередилося на розумінні ролі шуму та методів семплінгу в дифузійних моделях [43]. Автори провели емпіричне дослідження різних технік дифузії та семплінгу, що дозволило краще зрозуміти функціональність нейронної мережі та необхідність високої складності для оптимальної продуктивності.

Аналіз наведених досліджень свідчить про стрімкий розвиток теоретичного розуміння дифузійних моделей та їх практичного застосування. Особливо важливими є результати щодо властивостей генералізації, здатності до композиційного навчання та механізмів навчання ознак. Ці досягнення створюють міцну теоретичну базу для подальшого вдосконалення та застосування дифузійних моделей у різноманітних задачах комп'ютерного зору.

1.5.1 Огляд теоретичних робіт

Теоретичні дослідження в галузі генерації текстур за допомогою дифузійних моделей охоплюють широкий спектр підходів та методологій, від процедурної генерації до складних нейромережевих архітектур. Особлива увага приділяється розробці ефективних алгоритмів, які забезпечують високу якість генерації при оптимальному використанні обчислювальних ресурсів.

У дослідженні 2024 року представлено метод TexPro, який демонструє інноваційний підхід до генерації текстур на основі текстових описів [44]. Автори розробили систему, яка генерує не лише RGB текстури, але й повноцінні PBR-матеріали через процедурне моделювання. Особливістю методу є використання мультиракурсних референсних зображень та

впровадження спеціального матеріального агента для покращення класифікації та зіставлення матеріалів.

Значний внесок у розвиток процедурної генерації текстур зробило дослідження поліноміальних методів [45]. Запропонований у 2018 році підхід забезпечує генерацію 3D-рельєфу та текстур за допомогою когерентного шуму, демонструючи кращу продуктивність порівняно з традиційними методами фрактального броунівського руху. Автори провели детальний фрактальний аналіз згенерованих ландшафтів, порівнюючи їх з реальними даними.

Особливої уваги заслуговує дослідження μNCA [46], опубліковане в 2021 році, яке представляє ультракомпактні нейронні клітинні автомати для генерації текстур. Дослідники продемонстрували можливість представлення складних текстурних патернів за допомогою всього кількох сотень параметрів, що робить їх порівнянними за виразністю з ручними процедурними генераторами текстур. Найменші моделі сімейства μNCA містять лише 68 параметрів при збереженні високої якості генерації.

Аналіз теоретичних робіт демонструє еволюцію підходів до генерації текстур: від класичних процедурних методів до сучасних нейромережевих рішень. Особливо помітною є тенденція до оптимізації моделей за розміром та обчислювальною ефективністю при збереженні високої якості результатів. Дослідження останніх років зосереджені на розробці гібридних підходів, які поєднують переваги різних методів генерації.

1.5.2 Аналіз практичних імплементацій

Практичні імплементації Wasserstein GAN (WGAN) демонструють значний прогрес у вирішенні проблем стабільності та якості генерації, притаманних класичним GAN архітектурам. Аналіз останніх досліджень

показує різноманітні підходи до оптимізації та вдосконалення WGAN, що дозволяє краще зрозуміти їх практичне застосування.

Дослідження 2020 року представило важливу нову теоретичну інформацію щодо властивостей WGAN [47]. Автори детально проаналізували архітектуру WGAN в контексті інтегральних метрик ймовірності, параметризованих нейронними мережами. Особливу увагу було приділено оптимізаційним властивостям, що виникають при використанні параметричного 1-Ліпшицевого дискримінатора, а також вивченню збіжності емпіричних WGAN при збільшенні розміру вибірки.

У дослідженні 2024 року було встановлено важливий зв'язок між класичними GAN та метрикою Вассерштейна [48]. Це дозволило розширити існуючі результати WGAN на vanilla GAN архітектури. Дослідники отримали oracle inequality для vanilla GAN у метриці Вассерштейна та продемонстрували, що ці результати застосовні до практичних архітектур, включаючи feedforward ReLU мережі.

Значний практичний внесок було зроблено у дослідженні 2018 року, де запропоновано модифікацію WGAN з використанням Total Variational (TV) регуляризації [49]. Замість обмеження ваг або градієнтного штрафу, автори впровадили TV-регуляризацію для забезпечення умови Ліпшиця на мережі критика. Цей підхід продемонстрував кращу стабільність при навчанні порівняно з GP-WGAN та дозволив контролювати баланс між різноманітністю та якістю згенерованих зображень.

Аналіз практичних імплементацій WGAN демонструє еволюцію підходів від базових теоретичних концепцій до складних модифікацій, спрямованих на покращення стабільності та якості генерації. Особливо важливим є внесок TV-регуляризації та розуміння зв'язків між різними типами GAN, що відкриває нові можливості для практичного застосування цих архітектур у задачах генерації текстур та інших типів даних.

Висновки до розділу 1

У першому розділі проведено комплексний аналіз сучасного стану досліджень у сфері генерації текстур для комп'ютерних ігор з використанням методів глибинного навчання та генеративних нейронних мереж. Виконаний огляд наукової літератури охоплює широкий спектр публікацій від фундаментальних теоретичних робіт до практичних імплементацій.

Виявлено значний прогрес у розвитку генеративних моделей, зокрема в архітектурах GAN та їх модифікаціях. Дослідження показали, що сучасні підходи, такі як Conditional GAN та Wasserstein GAN, демонструють суттєве покращення стабільності навчання та якості генерації порівняно з класичними архітектурами. Особливо важливим досягненням є розробка механізмів контролю генерації, що дозволяють створювати текстури із заданими характеристиками.

Аналіз останніх публікацій виявив тенденцію до розробки спеціалізованих архітектур для конкретних задач. Зокрема, дослідження 2023-2024 років демонструють значний прогрес у створенні моделей, оптимізованих для роботи з піксельною графікою та текстурами специфічного стилю, що особливо актуально для генерації контенту Minecraft.

Встановлено, що дифузійні моделі представляють перспективний напрямок досліджень, пропонуючи альтернативний підхід до генерації зображень. Їх головними перевагами є висока якість результатів та краща стабільність навчання, хоча вони вимагають більших обчислювальних ресурсів порівняно з GAN-базованими архітектурами.

Важливим аспектом сучасних досліджень є розробка методів оптимізації обчислювальної ефективності моделей. Результати останніх робіт показують можливість значного зменшення кількості параметрів моделей при збереженні якості генерації, що робить їх більш придатними для практичного застосування.

Розглянуті дослідження також підкреслюють важливість правильного вибору метрик оцінки якості згенерованих текстур та необхідність розробки спеціалізованих методів валідації для ігрового контенту. Це створює основу для подальшого розвитку методів оцінки та порівняння різних підходів до генерації текстур.

Таким чином, проведений аналіз формує теоретичне підґрунтя для розробки власних моделей та методів генерації текстур, враховуючи останні досягнення в галузі та специфічні вимоги до ігрового контенту. Виявлені тенденції та перспективні напрямки досліджень визначають подальші кроки у розробці ефективних рішень для автоматизованої генерації текстур.

РОЗДІЛ 2 МЕТОДИ ГЕНЕРАЦІЇ ТЕКСТУР ТА АРХІТЕКТУРИ МОДЕЛЕЙ

У сучасному світі розробки комп'ютерних ігор все більшої актуальності набуває автоматизація створення ігрових ресурсів, зокрема текстур. Генеративні нейронні мережі представляють собою потужний інструмент для вирішення цього завдання, пропонуючи різноманітні архітектурні рішення та методи генерації високоякісного візуального контенту.

Базова архітектура GAN складається з двох основних компонентів - генератора та дискримінатора, що працюють у змагальному режимі. Генератор побудований на основі згорткових шарів з підвищенням розмірності, що дозволяє створювати зображення з випадкового шуму. Дискримінатор використовує згорткові шари зі зменшенням розмірності для класифікації зображень на справжні та згенеровані. Важливим аспектом є правильний підбір функцій втрат та оптимізаторів для стабільного навчання мережі.

Conditional GAN розширює можливості базової архітектури, додаючи механізми умовної генерації. Це досягається шляхом подачі додаткової інформації як генератору, так і дискримінатору, що дозволяє контролювати характеристики вихідних текстур. Особливу увагу приділено методам ембедингу умовної інформації та способам її інтеграції в архітектуру мережі.

Wasserstein GAN with Gradient Penalty представляє вдосконалену версію GAN, що вирішує проблеми стабільності навчання. Ключовими модифікаціями є використання метрики Вассерштейна як функції втрат та впровадження градієнтного штрафу. Ці зміни забезпечують більш стабільне навчання та покращують якість генерації текстур.

Дифузійна модель пропонує принципово інший підхід до генерації зображень, базуючись на поступовому процесі очищення зашумлених даних. В основі архітектури лежить U-Net структура, яка ефективно обробляє різні

рівні деталізації зображення. Процес генерації включає послідовне додавання шуму та його видалення, що дозволяє отримувати високоякісні текстури.

Таким чином, кожна з розглянутих архітектур має свої унікальні особливості та переваги у контексті генерації текстур для комп'ютерних ігор. Розуміння їх структури, принципів роботи та методів оптимізації є критично важливим для успішного застосування цих технологій у практичних задачах розробки ігрового контенту.

2.1 Архітектура базової моделі GAN

У світі розробки комп'ютерних ігор генеративні змагальні мережі (GAN) стали потужним інструментом для автоматичного створення графічного контенту. Базова архітектура GAN представляє собою фундаментальну концепцію, на якій будуються більш складні моделі для генерації текстур.

Генеративна змагальна мережа складається з двох основних компонентів: генератора та дискримінатора, які працюють у постійному змаганні один з одним. Генератор намагається створювати зображення, які б виглядали максимально реалістично, в той час як дискримінатор навчається відрізняти справжні зображення від згенерованих. Така архітектура дозволяє системі поступово покращувати якість генерації через ітеративний процес навчання.

Математично процес навчання описується двома ключовими формулами, а саме - функцією втрат дискримінатора (2.1):

$$\mathcal{L}\_D = -\mathbb{E}[log(D(x))] - \mathbb{E}[log(1 - D(G(z)))] \quad (2.1)$$

де D(x) - вихід дискримінатора для реального зображення,

G(z) - вихід генератора для випадкового шуму z,

D(G(z)) - оцінка дискримінатором згенерованого зображення;

та функцією втрат генератора (2.2):

$$\mathcal{L}\_G = \mathbb{E}[log(1 - D(G(z)))] \quad (2.2)$$

де G(z) - згенероване зображення випадкового шуму z,

D(G(z)) - оцінка дискримінатором згенерованого зображення.

Генератор побудований на основі згорткової нейронної мережі, яка перетворює випадковий шум у зображення текстури. Архітектура включає шари підвищення розмірності (upsampling), згорткові шари та шари нормалізації. Важливим елементом є використання функції активації ReLU та механізмів пропуску з'єднань (skip connections) для кращого збереження деталей текстури.

Дискримінатор також використовує згорткову архітектуру, але з протилежною метою - зменшення розмірності вхідного зображення до скалярної оцінки його реалістичності. У його структурі застосовуються шари згортки зі зменшенням розмірності (strided convolutions), пакетна нормалізація та функція активації LeakyReLU.

Навчання моделі відбувається через оптимізацію функції втрат, яка складається з двох компонентів: втрати генератора та втрати дискримінатора. Для оптимізації використовується алгоритм Adam з адаптивним кроком навчання. Важливим аспектом є балансування процесу навчання між генератором та дискримінатором для запобігання проблемі колапсу мод.

Базова архітектура GAN забезпечує фундамент для подальших модифікацій та вдосконалень у контексті генерації текстур для ігор. Розуміння її принципів роботи та особливостей реалізації є критичним для розробки більш складних архітектур та досягнення кращих результатів у генерації специфічних типів текстур.

2.1.1 Структура генератора та дискримінатора

Генеративно-змагальні мережі (GAN) складаються з двох основних компонентів - генератора та дискримінатора, які працюють у постійній конкуренції між собою. Розуміння їх структури та взаємодії є ключовим для успішної реалізації генеративної моделі.

Генератор представляє собою нейронну мережу, яка отримує на вхід випадковий шум з латентного простору та перетворює його на синтетичні зображення текстур. Типова архітектура генератора складається з послідовності шарів деконволюції (транспонованої згортки), які поступово збільшують просторову розмірність та формують візуальні особливості текстур. Між шарами використовуються функції активації (найчастіше ReLU або LeakyReLU) та нормалізація батчів для стабілізації навчання, що описується формулою (2.3):

$$G(z) = max(0, W_i X + b_i) \quad (2.3)$$

де W_i - матриця ваг, X - вхідний вектор, b_i - вектор зміщення.

Дискримінатор реалізований як згорткова нейронна мережа, що класифікує вхідні зображення на справжні та згенеровані. Його архітектура зазвичай симетрична до генератора та містить послідовність згорткових шарів, які поступово зменшують просторову розмірність та витягують ієрархічні ознаки зображень. На виході дискримінатор видає скалярну оцінку достовірності для кожного зображення згідно формули (2.4):

$$D(x) = \sigma(Wx + b) \quad (2.4)$$

де σ - сигмоїдна функція активації, W - матриця ваг, x - вхідне зображення, b - зміщення.

Важливою особливістю архітектури є використання skip-з'єднань між відповідними шарами генератора та дискримінатора, що описується як (2.5):

$$F(x) = H(x) + x \quad (2.5)$$

де H(x) - вихід попереднього шару, x - вхідний сигнал.

Це дозволяє ефективніше передавати інформацію про структуру текстур. Також застосовуються механізми уваги для покращення якості деталей та забезпечення глобальної узгодженості згенерованих текстур.

Для оптимізації роботи з піксельною графікою в архітектуру додані спеціалізовані шари для обробки дискретних значень пікселів та збереження чітких границь об'єктів. Розмір та кількість шарів підібрані з урахуванням типового розміру ігрових текстур та необхідної деталізації.

Структура генератора та дискримінатора оптимізована для ефективної генерації високоякісних ігрових текстур з урахуванням специфіки піксельної графіки. Взаємодія цих компонентів забезпечує поступове покращення якості згенерованих зображень в процесі навчання моделі.

2.2 Реалізація Conditional GAN

У процесі розробки та навчання генеративних нейронних мереж ключову роль відіграє правильний вибір функцій втрат та методів оптимізації. Ці компоненти визначають якість навчання моделі та її здатність генерувати реалістичні текстури.

Базова функція втрат GAN складається з двох компонентів: функції втрат дискримінатора та функції втрат генератора. Дискримінатор оптимізується для максимізації точності розрізнення справжніх та згенерованих зображень, тоді як генератор прагне мінімізувати ймовірність виявлення підробок. Математично це виражається через бінарну крос-ентропію (2.6):

$$\mathcal{L}(G, D) = \mathbb{E}_x[log\, D(x)] + \mathbb{E}_y[log(1 - D(G(z)))] \quad (2.6)$$

Для стабілізації процесу навчання застосовуються модифіковані функції втрат. Однією з найефективніших є least squares loss, яка допомагає зменшити проблему зникаючих градієнтів та забезпечує більш стабільне навчання (2.7)-(2.8):

$$\mathcal{L}\_LSGAN(D) = ½\mathbb{E}[(D(x) - 1)^2] + ½\mathbb{E}[(D(G(z)))^2] \quad (2.7)$$

$$\mathcal{L}\_LSGAN(G) = ½\mathbb{E}[(D(G(z)) - 1)^2] \quad (2.8)$$

Також використовується feature matching loss, що порівнює активації проміжних шарів дискримінатора для реальних та згенерованих зображень (2.9):

$$\mathcal{L}\_FM = ||f(x) - f(G(z))||_2 \quad (2.9)$$

де f(·) - активації проміжного шару дискримінатора.

У якості оптимізатора найчастіше використовується Adam з адаптивним розміром кроку навчання. Важливим є правильний підбір гіперпараметрів оптимізатора, зокрема коефіцієнтів β1 та β2, які впливають на швидкість оновлення моментів першого та другого порядку. Для запобігання

перенавчання застосовується регуляризація через dropout та batch normalization.

Особливу увагу слід приділити балансуванню навчання генератора та дискримінатора. Практика показує, що оптимальним є почергове навчання з різною кількістю ітерацій для кожної складової. Також важливим є використання технік регуляризації градієнтів для запобігання їх вибуху або затухання.

Ефективність генерації текстур значною мірою залежить від коректного вибору та налаштування функцій втрат та методів оптимізації. Правильна комбінація цих компонентів забезпечує стабільне навчання моделі та високу якість згенерованих зображень, що є критичним для практичного застосування в розробці комп'ютерних ігор.

2.2.1 Механізми умовної генерації

Механізми умовної генерації є ключовим компонентом у розробці генеративних моделей, що дозволяють створювати текстури із заданими характеристиками. Ці механізми забезпечують можливість контролювати процес генерації та отримувати результати з бажаними властивостями.

Основним принципом умовної генерації є додавання додаткової інформації до вхідного шуму генератора. Ця інформація може бути представлена у вигляді категоріальних міток, числових параметрів або навіть складних векторних представлень характеристик текстур. Механізм умовної генерації реалізується згідно формули (2.10):

$$G(z, c) = F(concat(z, E(c))) \quad (2.10)$$

де z - вхідний шум, c - умовна інформація, E(c) - кодування умови, F - функція генератора.

Важливим аспектом є вибір способу кодування умовної інформації. Для категоріальних умов застосовується one-hot кодування, тоді як для неперервних параметрів використовується пряме числове представлення або нормалізовані значення, що можна описати як (2.11,-12):

$$E(c) = \{1_i \mid i = c\} \text{ для категорій} \quad (2.11)$$

$$E(c) = (c - \mu) / \sigma \text{ для неперервних значень} \quad (2.12)$$

Архітектура умовного генератора містить додаткові модулі для обробки умовної інформації. Це можуть бути повнозв'язні шари для перетворення умовного вектора, механізми уваги для врахування просторових залежностей або спеціалізовані блоки для інтеграції умовної інформації на різних рівнях генерації, що виражається формулою (2.13):

$$h\_i = \gamma\_c(h\_i) * norm(h\_i) + \beta\_c(h\_i) \quad (2.13)$$

де γ_c та β_c - параметри умовної нормалізації, h_i - проміжні активації.

Особливу роль в умовній генерації відіграють механізми нормалізації. Conditional Batch Normalization та Adaptive Instance Normalization дозволяють ефективно інтегрувати умовну інформацію в процес генерації через модуляцію статистик нормалізації. Це забезпечує більш стабільне навчання та кращий контроль над генерацією.

Таким чином, механізми умовної генерації є складною системою взаємопов'язаних компонентів, що забезпечують можливість керованої генерації текстур. Правильна реалізація цих механізмів є критичною для створення генеративних моделей, здатних враховувати специфічні вимоги до генерованих текстур та забезпечувати високу якість результатів.

2.2.2 Особливості навчання моделі

Навчання умовних генеративних змагальних мереж (Conditional GAN) має ряд специфічних особливостей, які потребують детального розгляду для забезпечення ефективної генерації текстур. Розглянемо ключові аспекти процесу навчання та оптимізації таких моделей.

Основною особливістю навчання Conditional GAN є необхідність одночасного тренування генератора та дискримінатора з урахуванням умовної інформації. При цьому важливо забезпечити баланс між їх навчанням, щоб уникнути домінування однієї з мереж. Для цього застосовується почергове оновлення ваг мереж з різною частотою, зазвичай дискримінатор оновлюється частіше за генератор.

Важливим аспектом є вибір гіперпараметрів навчання. Експериментальним шляхом встановлено, що оптимальним є використання алгоритму Adam з параметрами learning rate = 0.0002 та beta1 = 0.5. Розмір batch повинен бути достатньо великим (32-128) для стабільного навчання, але не занадто великим, щоб уникнути проблем з пам'яттю.

Для покращення стабільності навчання застосовуються спеціальні техніки: нормалізація батчів, dropout та noise injection. Batch normalization допомагає боротися з internal covariate shift, dropout запобігає перенавчанню, а додавання шуму до вхідних даних підвищує різноманітність генерованих зразків.

Особливу увагу слід приділити моніторингу процесу навчання. Важливо відслідковувати не лише значення функції втрат, але й візуальну якість генерованих текстур. Для цього використовується періодична генерація зразків з фіксованим латентним вектором, що дозволяє оцінити прогрес навчання.

Критичним є правильний вибір умовної інформації та способу її подання мережі. Умовні ознаки повинні бути інформативними та релевантними для задачі генерації текстур. Вони можуть подаватися у вигляді one-hot векторів, ембедингів або безпосередньо числових значень.

Підсумовуючи, успішне навчання Conditional GAN вимагає ретельного налаштування гіперпараметрів, застосування спеціальних технік стабілізації та постійного моніторингу процесу. Правильна організація процесу навчання є ключовим фактором для отримання якісних результатів генерації текстур з урахуванням заданих умов.

2.3 Імплементація Wasserstein GAN with Gradient Penalty

У контексті розвитку генеративних нейронних мереж особливу увагу привертає архітектура Wasserstein GAN with Gradient Penalty (WGAN-GP), яка пропонує вдосконалений підхід до стабільного навчання генеративних моделей. Ця модифікація вирішує ключові проблеми традиційних GAN, забезпечуючи більш надійну конвергенцію та якісніші результати генерації.

Основною архітектурною особливістю WGAN-GP є заміна класичного дискримінатора на критика, який оцінює відстань Вассерштейна між реальним та згенерованим розподілами даних. Математично це виражається як (2.14):

$$W(P_r, P_\gamma) = sup||f||^L \leq 1 \, \mathbb{E}_x P_r[f(x)] - \mathbb{E}_x P_\gamma[f(x)] \qquad (2.14)$$

де P_r та P_γ - реальний та згенерований розподіли, f - функція критика.

Ключовим елементом WGAN-GP є впровадження градієнтного штрафу, який визначається формулою (2.15):

$$\mathcal{L}_{penalty} = \lambda \mathbb{E}_{\hat{x}} \sim P_{\hat{x}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \qquad (2.15)$$

де $\hat{x} = \varepsilon x + (1-\varepsilon)G(z)$, $\varepsilon \sim U[0,1]$, λ - коефіцієнт штрафу.

Повна функція втрат WGAN-GP включає як відстань Вассерштейна, так і градієнтний штраф (2.16):

$$\mathcal{L} = \mathbb{E}_{x}P_{r}[D(x)] - \mathbb{E}_{\gamma}P_{\gamma}[D(G(z))] + \lambda\mathcal{L}_{\text{penalty}_{\gamma}} \quad (2.16)$$

Процес навчання WGAN-GP включає кілька важливих модифікацій порівняно зі стандартним GAN. Замість бінарної крос-ентропії використовується функція втрат Вассерштейна, яка забезпечує більш плавний градієнт. Дискримінатор оновлюється частіше за генератор, що дозволяє йому краще апроксимувати відстань Вассерштейна. Додатково використовується адаптивна нормалізація для покращення стабільності навчання.

Для оптимізації навчання застосовується алгоритм Adam з модифікованими гіперпараметрами. Важливим аспектом є відсутність batch normalization у дискримінаторі, оскільки це може конфліктувати з градієнтним штрафом. Натомість використовується layer normalization, що забезпечує кращу стабільність при збереженні переваг нормалізації.

Імплементація WGAN-GP представляє собою комплексне рішення для стабільного навчання генеративних моделей. Завдяки використанню відстані Вассерштейна та градієнтного штрафу досягається більш надійна конвергенція та покращена якість генерації.

2.3.1 Архітектурні модифікації

У контексті розробки генеративних моделей для створення текстур важливим аспектом є впровадження архітектурних модифікацій, які дозволяють покращити якість генерації та стабільність навчання. Архітектурні модифікації Wasserstein GAN with Gradient Penalty (WGAN-GP) спрямовані на

подолання основних недоліків класичних GAN та забезпечення кращої збіжності моделі.

Основною архітектурною модифікацією WGAN-GP є видалення сигмоїдального шару з дискримінатора, що дозволяє уникнути проблеми насичення градієнтів. Замість цього використовується лінійний вихідний шар, який повертає скалярне значення без обмеження діапазону. Це забезпечує більш стабільний градієнтний потік під час навчання.

Важливою модифікацією є впровадження нормалізації вагів у обох мережах - генераторі та дискримінаторі. Ваги кожного шару нормалізуються так, щоб їх L2-норма не перевищувала заданого порогу. Це допомагає контролювати ліпшицеве обмеження функції дискримінатора та забезпечує кращу стабільність тренування.

У архітектурі WGAN-GP також модифіковано структуру згорткових шарів. Замість звичайних згорткових шарів використовуються залишкові блоки (residual blocks), які містять додаткові обхідні з'єднання. Це дозволяє ефективніше передавати градієнти через мережу та покращує здатність моделі вивчати складні патерни текстур.

Для покращення якості генерації впроваджено механізм самоуваги (self-attention), який дозволяє моделі враховувати глобальний контекст зображення. Блоки самоуваги розміщуються між згортковими шарами та допомагають встановлювати залежності між різними областями текстури.

Підсумовуючи, архітектурні модифікації WGAN-GP спрямовані на забезпечення стабільності навчання та покращення якості генерації текстур. Комбінація нормалізації вагів, залишкових блоків та механізмів уваги створює потужну архітектуру, здатну генерувати високоякісні текстури для комп'ютерних ігор.

2.3.2 Градієнтний штраф та стабілізація навчання

Градієнтний штраф є ключовим компонентом для стабілізації процесу навчання Wasserstein GAN, що дозволяє уникнути проблем з розбіжністю градієнтів та забезпечує більш стабільну конвергенцію моделі. Розглянемо основні аспекти реалізації та застосування градієнтного штрафу в контексті генерації текстур.

Основною проблемою при навчанні WGAN є необхідність забезпечення 1-Ліпшицевої неперервності критика. Градієнтний штраф вирішує цю проблему шляхом додавання регуляризаційного члена до функції втрат, який обчислюється за формулою (2.17):

$$GP = \lambda \mathbb{E}_{\hat{x}} \sim P_{\hat{x}} [(||\nabla_{\hat{x}} C(\hat{x})||_2 - 1)^2] \quad (2.17)$$

де $C(\hat{x})$ - вихід критика, λ - коефіцієнт штрафу.

Для реалізації градієнтного штрафу використовується інтерполяція між реальними та згенерованими зразками, що визначається як (2.18):

$$\hat{x} = \alpha x + (1 - \alpha) G(z), \alpha \sim U[0,1] \quad (2.18)$$

де x - реальний зразок, G(z) - згенерований зразок.

Повна функція втрат з урахуванням градієнтного штрафу записується як (2.19):

$$\mathcal{L}\_total = \mathcal{L}\_WGAN + \lambda GP \quad (2.19)$$

де $\mathcal{L}$_WGAN - основна функція втрат Wasserstein GAN.

Для адаптивного налаштування параметрів використовується модифікована версія алгоритму Adam (2.20):

$$\theta_{t+1} = \theta_t - \eta \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon) \quad (2.20)$$

де η - learning rate, $\hat{m}_t$ та $\hat{v}_t$ - скориговані оцінки першого та другого моментів градієнтів.

Важливим аспектом є моніторинг градієнтів під час навчання. Реалізовано механізми відстеження норм градієнтів та автоматичного коригування параметрів навчання при виявленні нестабільності. Це дозволяє своєчасно реагувати на проблеми з навчанням та запобігати колапсу моделі.

Таким чином, впровадження градієнтного штрафу та механізмів стабілізації навчання є критичним для успішного навчання WGAN-GP моделі. Правильна імплементація цих компонентів забезпечує стабільну конвергенцію та покращує якість генерованих текстур, що підтверджується експериментальними результатами.

2.4 Розробка дифузійної моделі

У контексті розробки генеративних моделей для створення текстур особливу увагу привертають дифузійні моделі, які демонструють значний потенціал у генерації високоякісних зображень. Ці моделі базуються на поступовому процесі додавання та видалення шуму, що дозволяє отримувати деталізовані та реалістичні результати.

Основою розробленої дифузійної моделі є архітектура U-Net, яка забезпечує ефективну обробку візуальної інформації на різних рівнях деталізації. U-Net складається з енкодера та декодера, з'єднаних skip-connection зв'язками, що дозволяють зберігати важливу інформацію при трансформації зображень. Енкодер послідовно зменшує просторову

розмірність вхідних даних, тоді як декодер відновлює оригінальний розмір, зберігаючи семантичну інформацію.

Процес дифузії реалізується через послідовне додавання гаусового шуму до вхідного зображення протягом T кроків. На кожному кроці t модель навчається передбачати доданий шум, що дозволяє поступово відновлювати чисте зображення. Важливим аспектом є використання позиційного кодування часового кроку, яке допомагає моделі враховувати прогрес процесу денойзингу.

Денойзинг здійснюється шляхом ітеративного видалення шуму, починаючи з повністю зашумленого зображення. На кожній ітерації модель оцінює та видаляє частину шуму, поступово відновлюючи структуру та деталі текстури. Цей процес контролюється розкладом шуму, який визначає інтенсивність шуму на кожному кроці.

Для покращення якості генерації впроваджено механізми уваги в архітектуру U-Net, що дозволяють моделі фокусуватися на найбільш релевантних областях зображення. Додатково реалізовано техніки регуляризації та нормалізації для стабілізації процесу навчання та покращення якості результатів.

Розроблена дифузійна модель демонструє здатність генерувати деталізовані текстури з чіткою структурою та консистентними візуальними характеристиками. Особливістю моделі є можливість контролювати процес генерації через параметри дифузії та денойзингу, що робить її гнучким інструментом для створення різноманітних ігрових текстур.

2.4.1 Структура U-Net архітектури

U-Net архітектура є однією з найважливіших складових сучасних генеративних моделей, особливо в контексті дифузійних моделей для

генерації зображень. Ця архітектура забезпечує ефективне кодування та декодування візуальної інформації завдяки своїй унікальній структурі.

Основою U-Net архітектури є симетрична структура у формі літери "U", що складається з енкодера та декодера. Енкодер послідовно зменшує просторову розмірність вхідних даних через згорткові шари, збільшуючи при цьому кількість каналів ознак. Це дозволяє моделі ефективно вилучати ієрархічні особливості зображення на різних рівнях абстракції.

Декодер виконує зворотну операцію, поступово відновлюючи просторову розмірність через шари розгортки. Особливістю архітектури є наявність skip-з'єднань між відповідними шарами енкодера та декодера. Ці з'єднання дозволяють передавати детальну інформацію з ранніх шарів безпосередньо до пізніх шарів, що забезпечує збереження дрібних деталей у вихідному зображенні.

Кожен рівень U-Net містить блоки згорткових шарів з активаційними функціями та нормалізацією. Між рівнями використовуються операції об'єднання (pooling) для зменшення розмірності в енкодері та операції підвищення дискретизації (upsampling) в декодері. Така структура забезпечує ефективне стиснення інформації з подальшим її відновленням.

Важливим елементом архітектури є механізми уваги, інтегровані в різні рівні мережі. Вони дозволяють моделі фокусуватися на найбільш релевантних областях зображення та встановлювати довгострокові залежності між різними частинами текстури.

U-Net архітектура демонструє високу ефективність у задачах генерації текстур завдяки своїй здатності зберігати як глобальну структуру, так і локальні деталі зображення. Симетрична структура та наявність skip-з'єднань забезпечують оптимальний баланс між узагальненням та деталізацією, що особливо важливо при роботі з ігровими текстурами.

2.4.2 Процес дифузії та денойзингу

Процес дифузії та денойзингу є ключовим компонентом дифузійних моделей, що забезпечує поступове перетворення випадкового шуму в осмислене зображення. Цей процес складається з двох основних етапів: прямої дифузії, під час якої до зображення поступово додається шум, та зворотної дифузії (денойзингу), коли модель крок за кроком видаляє шум для отримання кінцевого результату.

Пряма дифузія реалізується через послідовне додавання гаусівського шуму. Математично процес описується формулою (2.21):

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{(1-\beta_t)}x_{t-1}, \beta_t I) \quad (2.21)$$

де β_t - параметр шуму на кроці t, x_{t-1} - стан на попередньому кроці.

Повний процес дифузії може бути представлений як (2.22):

$$q(x_t|x_0) = N(x_t; \sqrt{(\bar{\alpha}_t)}x_0, (1-\bar{\alpha}_t)I) \quad (2.22)$$

де $\bar{\alpha}_t = \prod_{i=1}^{t}(1-\beta_i)$.

Процес денойзингу описується рівнянням (2.23):

$$p(x_{t-1}|x_t) = N(x_{t-1}; \mu_t(x_t, t), \sigma_t^2 I) \quad (2.23)$$

де μ_t - передбачення нейронної мережі, σ_t^2 - дисперсія шуму.

Для покращення якості генерації використовується модифікована функція втрат (2.24):

$$\mathcal{L} = \mathbb{E}_{t,x0,\varepsilon}[||\varepsilon - \varepsilon_t(\sqrt{(\bar{\alpha}_t)}x_0 + \sqrt{(1-\bar{\alpha}_t)}\varepsilon, t)||^2] \quad (2.24)$$

де ε_t - нейронна мережа для передбачення шуму, $\varepsilon \sim N(0,I)$.

Ефективність процесу дифузії та денойзингу значною мірою залежить від правильного вибору параметрів, таких як кількість кроків дифузії, розклад шуму та архітектура нейронної мережі. Оптимальний баланс цих параметрів забезпечує високу якість генерованих текстур при збереженні обчислювальної ефективності.

Висновки до розділу 2

У другому розділі було проведено детальний аналіз та розробку різних архітектур генеративних нейронних мереж для створення текстур комп'ютерних ігор. Розглянуто чотири основні підходи: базова модель GAN, Conditional GAN, Wasserstein GAN with Gradient Penalty та дифузійна модель.

Досліджено структуру та принципи роботи базової архітектури GAN, яка складається з генератора та дискримінатора. Проаналізовано математичні основи їх взаємодії через функції втрат та методи оптимізації. Визначено, що ключовими елементами успішного навчання є правильний підбір гіперпараметрів та балансування процесу навчання між компонентами мережі.

Розглянуто механізми умовної генерації в архітектурі Conditional GAN, які дозволяють контролювати характеристики вихідних текстур. Показано, що додавання умовної інформації через спеціальні механізми кодування та нормалізації значно покращує керованість процесу генерації. Особливу увагу приділено методам інтеграції умовної інформації та способам її ефективного використання в процесі навчання.

Детально проаналізовано архітектуру Wasserstein GAN with Gradient Penalty, яка вирішує проблеми стабільності навчання через використання метрики Вассерштейна та градієнтного штрафу. Продемонстровано, що ця

модифікація забезпечує більш надійну конвергенцію та покращену якість генерації порівняно з базовою архітектурою GAN.

Досліджено структуру та принципи роботи дифузійної моделі, основою якої є архітектура U-Net. Проаналізовано процеси прямої та зворотної дифузії, а також механізми денойзингу. Показано, що цей підхід дозволяє отримувати високоякісні результати завдяки поступовому процесу генерації та ефективному збереженню інформації через skip-з'єднання.

В результаті проведеного дослідження встановлено, що кожна з розглянутих архітектур має свої переваги та обмеження. Базова модель GAN забезпечує фундамент для розробки більш складних архітектур, Conditional GAN додає можливості керування генерацією, Wasserstein GAN покращує стабільність навчання, а дифузійна модель пропонує альтернативний підхід з потенційно вищою якістю результатів.

Розроблені математичні моделі та алгоритми створюють основу для практичної реалізації системи генерації текстур для комп'ютерних ігор, що буде розглянуто в наступних розділах роботи.

РОЗДІЛ 3 РОЗРОБКА ГЕНЕРАТИВНИХ НЕЙРОННИХ МЕРЕЖ ДЛЯ ЗАДАЧ КОМП'ЮТЕРНОГО ЗОРУ ТА ГЕНЕРАЦІЇ ТЕКСТУР ДЛЯ КОМП’ЮТЕРНИХ ІГОР

3.1 Аналіз сучасного стану проблеми генерації текстур

Генерація текстур є однією з ключових задач комп'ютерного зору та комп'ютерної графіки, яка набула особливої актуальності з розвитком індустрії відеоігор та цифрових медіа. Традиційні методи створення текстур, які базуються на ручній роботі художників, стають все менш ефективними через зростаючі потреби в різноманітному контенті та високу вартість розробки [50].

Історично перші спроби автоматизованої генерації текстур базувались на процедурних методах, які використовували математичні функції та алгоритми для створення повторюваних патернів [51]. Ці підходи, хоча і дозволяли створювати базові текстури для природних об'єктів (каміння, дерева, хмари), мали обмежені можливості щодо створення складних структурованих текстур, особливо тих, що мають специфічний художній стиль [52].

Значний прорив у галузі генерації текстур відбувся з появою глибинного навчання та генеративних нейронних мереж. У 2014 році було представлено архітектуру Generative Adversarial Networks (GAN), яка відкрила нові можливості для генерації реалістичних зображень [53]. Подальший розвиток GAN-архітектур, включаючи DCGANs, StyleGAN та BigGAN, дозволив покращити якість генерації та контроль над процесом створення текстур [54, 55].

Сучасні дослідження в області генерації текстур фокусуються на кількох ключових напрямках. Перший – це покращення якості та деталізації

згенерованих текстур. Використання прогресивного навчання та багатомасштабних архітектур дозволяє досягати все більш високої роздільної здатності та фотореалістичності [56]. Другий напрямок – це контрольована генерація, де користувач може впливати на характеристики вихідної текстури через семантичні параметри або референсні зображення [57].

Особливу увагу дослідники приділяють проблемі стильової узгодженості генерованих текстур. У контексті відеоігор це особливо важливо, оскільки текстури повинні відповідати загальній естетиці гри. Наприклад, для Minecraft характерний піксельний стиль з чіткими границями та обмеженою колірною палітрою [58]. Сучасні методи, засновані на conditional GAN та style transfer, дозволяють враховувати ці стильові обмеження під час генерації [59].

Важливим аспектом сучасних досліджень є також оптимізація обчислювальних ресурсів. Хоча глибокі нейронні мережі демонструють вражаючі результати, їх навчання та використання вимагає значних обчислювальних потужностей. Розробляються методи компресії моделей та прискорення інференсу, що особливо важливо для реального застосування в ігрових додатках [60].

Новим перспективним напрямком є використання дифузійних моделей для генерації текстур. На відміну від GAN, дифузійні моделі показують більшу стабільність у навчанні та різноманітність генерованих результатів [61]. Проте вони все ще мають обмеження щодо швидкості генерації та вимог до обчислювальних ресурсів.

У контексті генерації ігрових текстур особливу роль відіграють методи, що дозволяють створювати тайлові текстури, які можна безшовно поєднувати. Дослідження в цьому напрямку поєднують традиційні підходи до створення безшовних текстур з можливостями глибинного навчання [62].

3.1.1 Огляд існуючих методів та підходів до генерації текстур

У сучасній комп'ютерній графіці існує широкий спектр методів та підходів до генерації текстур, які постійно еволюціонують завдяки розвитку технологій штучного інтелекту та глибинного навчання. Розглянемо основні напрямки та останні досягнення в цій галузі.

Одним з найбільш перспективних напрямків є генерація текстур на основі текстових описів. Новаторська робота Decorate3D [63] представляє комплексний підхід до створення та редагування текстур для 3D-об'єктів. Ключовою особливістю цього методу є декомпозиція нейронного радіаційного поля (Neural Radiance Field) на явне сіткове представлення та текстуру, залежну від точки огляду. Це дозволяє досягти високої якості текстур з роздільною здатністю до 2048x2048 пікселів, що особливо важливо для реалістичного рендерингу в сучасних відеоіграх та віртуальній реальності.

Значний прорив у швидкості та якості генерації текстур представлено в роботі Meta 3D TextureGen [64]. Дослідники з Meta розробили метод, який дозволяє генерувати глобально узгоджені високоякісні текстури менш ніж за 20 секунд. Це досягається завдяки використанню двох послідовних нейронних мереж та кондиціонуванню моделі text-to-image на 3D-семантиці у 2D-просторі. Особливо важливим є впровадження мережі покращення текстур, яка здатна масштабувати текстури до роздільної здатності 4K, що відповідає сучасним вимогам індустрії.

Практичне застосування цих методів можна спостерігати в сучасних онлайн-сервісах, таких як Poly.cam [65], які надають доступ до інструментів генерації текстур широкому колу користувачів. Ці сервіси демонструють можливості створення складних текстур для різноманітних об'єктів, включаючи вінтажні мотоцикли та інші складні механічні об'єкти, зберігаючи при цьому високу деталізацію та стилістичну відповідність.

Важливим аспектом сучасних методів генерації текстур є їх здатність враховувати геометричну структуру об'єкта. Наприклад, при генерації текстури для мотоцикла система повинна правильно накладати матеріали на різні частини об'єкта, враховуючи їх функціональне призначення та фізичні властивості [66]. Це досягається завдяки використанню структурно-орієнтованих методів семплінгу та оптимізації UV-розгортки.

Сучасні підходи також приділяють значну увагу проблемі глобальної узгодженості текстур. Це особливо важливо для об'єктів, які мають складну геометрію та потребують безшовного з'єднання різних частин текстури. Використання нейронних мереж з механізмами уваги та спеціалізованих архітектур для обробки UV-карт дозволяє досягти високої якості стиків та переходів між різними частинами текстури [67].

Одним з ключових трендів є розробка методів, що дозволяють генерувати текстури з урахуванням фізичних властивостей матеріалів. Це включає створення карт нормалей, шорсткості, металічності та інших PBR-параметрів, що забезпечують реалістичне відображення матеріалів при різних умовах освітлення [68].

3.1.2 GAN (Generative Adversarial Networks)

Генеративно-змагальні мережі (GAN), представлені в 2014 році Яном Гудфеллоу та співавторами [69], є потужним інструментом для генерації зображень та текстур. Основна концепція GAN базується на змагальному навчанні двох нейронних мереж: генератора G та дискримінатора D. Генератор намагається створювати зображення, які будуть максимально схожі на реальні, тоді як дискримінатор прагне відрізнити справжні зображення від згенерованих. Цей процес математично описується як мінімаксна гра двох гравців (3.1):

$$min\ G\ max\ D\ V(D,G)\ =$$
$$=\ E\ xp\ data(x)\ [log\ D(x)]\ +\ E\ zp\ z(z)[log(1\ -\ D(G(z)))] \qquad (3.1),$$

де p data(x) представляє розподіл реальних даних, а p z(z) - розподіл латентного простору, зазвичай нормальний або рівномірний [70].

У контексті генерації текстур генератор G приймає на вхід випадковий шум z і перетворює його у текстуру заданого розміру. Архітектура генератора базується на послідовному збільшенні просторової розмірності через шари деконволюції, що можна представити як (3.2):

$$G(z)\ =\ tanh(Conv\ T\ (ReLU(\dots Conv\ T\ (ReLU(FC(z)))\dots))) \qquad (3.2),$$

де FC - повнозв'язний шар, Conv T - транспонована згортка, а ReLU використовується як функція активації [71].

Проте класична архітектура GAN має певні обмеження при роботі з текстурами. Основними проблемами є нестабільність тренування, складність досягнення конвергенції та явище колапсу мод, коли генератор починає продукувати обмежений набір схожих зразків. Для вирішення цих проблем була запропонована архітектура Deep Convolutional GAN (DCGAN) [72].

DCGAN вводить кілька ключових архітектурних модифікацій, які значно покращують стабільність та якість генерації текстур. У генераторі DCGAN використовуються транспоновані згорткові шари замість повнозв'язних, що дозволяє мережі краще вивчати просторові ієрархії ознак. Важливим нововведенням є використання пакетної нормалізації (batch normalization) після кожного згорткового шару, крім останнього. Це допомагає стабілізувати навчання та запобігає колапсу генератора [73].

Для генератора DCGAN характерна наступна архітектурна формула (3.3):

$$z \rightarrow FC \rightarrow reshape \rightarrow BN \rightarrow ReLU \rightarrow ConvT \rightarrow BN \rightarrow ReLU \rightarrow$$
$$\rightarrow ConvT \rightarrow BN \rightarrow ReLU \rightarrow ConvT \rightarrow tanh \quad (3.3),$$

де BN - batch normalization, а tanh використовується в останньому шарі для нормалізації вихідних значень до діапазону [-1, 1].

В контексті генерації текстур DCGAN демонструє особливу ефективність завдяки своїй здатності вловлювати як локальні, так і глобальні патерни. Згорткова природа архітектури дозволяє мережі автоматично вивчати ієрархічні особливості текстур - від простих елементів, таких як краї та кольорові переходи, до складних структурних патернів [74].

Важливою особливістю DCGAN при роботі з текстурами є його здатність зберігати просторову узгодженість у згенерованих зразках. Це досягається завдяки використанню глибоких згорткових шарів, які здатні вловлювати довгострокові залежності в даних. При генерації текстур це особливо важливо для забезпечення консистентності патернів та уникнення артефактів на межах текстурних елементів [75].

3.1.3 Conditional GAN

Conditional Generative Adversarial Networks (cGAN) розширюють концепцію класичних GAN шляхом додавання умовної інформації до процесу генерації [76]. На відміну від звичайних GAN, які генерують зразки лише з випадкового шуму, cGAN дозволяють контролювати процес генерації через додаткові вхідні дані y. Математично це можна представити як (3.4):

$$min\ G\ max\ D\ V(D,G) = E\ xp\ data(x)\ [log\ D(x|y)] +$$
$$+ E\ zp\ z(z)\ [log(1 - D(G(z|y)))] \quad (3.4),$$

де y - умовна інформація, яка може бути представлена у вигляді міток класів, зображень, текстових описів або інших структурованих даних [28].

У контексті генерації текстур умовна інформація може включати:

1) стильові референси;
2) семантичні карти;
3) кольорові палітри;
4) параметри матеріалів;
5) текстові описи бажаних властивостей текстури.

Архітектура генератора в cGAN модифікується для обробки як випадкового шуму z, так і умовної інформації y (3.5):

$$G(z,y) = tanh(Conv\,T\,(ReLU(\ldots Conv\,T\,(ReLU(FC([z;y])))\ldots))) \quad (3.5),$$

де [z;y] позначає конкатенацію векторів z та y [77].

Особливо важливим аспектом cGAN для генерації текстур є механізм внесення умовної інформації. Найпоширенішим підходом є використання умовної нормалізації (conditional normalization) (3.6):

$$\gamma\,y\,(BN(h)) + \beta\,y \quad (3.6),$$

де BN - batch normalization, а γ y та β y - параметри масштабування та зміщення, що залежать від умовної інформації y [78].

Для покращення якості генерації текстур у cGAN часто використовується багатомасштабний підхід, де умовна інформація вноситься на різних рівнях роздільної здатності. Це допомагає зберегти як глобальну структуру, так і локальні деталі текстури. Функція втрат при цьому може бути розширена додатковими членами (3.7):

$$L = L\_adv + \lambda\_1\,L\_feat + \lambda\_2\,L\_style \quad (3.7),$$

де L_feat відповідає за збереження структурних особливостей, а L_style - за стильову відповідність [79].

У контексті генерації текстур для ігрових персонажів cGAN демонструє особливу ефективність при необхідності створення текстур, що відповідають певному художньому стилю або технічним обмеженням. Наприклад, при генерації текстур для Minecraft можна використовувати умовну інформацію для забезпечення відповідності піксельному стилю гри та обмеженій колірній палітрі [80].

3.1.4 Wasserstein GAN with Gradient Penalty (GAN-WP)

Wasserstein GAN with Gradient Penalty (GAN-WP або WGAN-GP) представляє собою вдосконалену версію генеративно-змагальних мереж, яка вирішує проблеми стабільності навчання та якості генерації, характерні для класичних GAN [81]. Основна ідея полягає у використанні відстані Вассерштейна (Wasserstein distance) замість традиційної дивергенції Йенсена-Шеннона, що математично виражається як (3.8):

$$W(P\,r\,,P\,g\,)\;=\;sup\,||f||L\leq 1\;E\;xP\;r\;[f(x)]\;-\;E\;xP\;g\;[f(x)]\quad(3.8),$$

де P r та P g - розподіли реальних та згенерованих даних відповідно.

GAN-WP розширює цю концепцію додаванням градієнтного штрафу, який забезпечує виконання умови Ліпшиця для функції критика (колишній дискримінатор). Функція втрат GAN-WP має вигляд [82] (3.9):

$$L\;=\;E\;xP\;g\;[D(x)]\;-\;E\;xP\;r\;[D(x)]\;+\;L\;=$$
$$=E\;xP\;g\;[D(x)]\;-\;E\;xP\;r\;[D(x)]\;+$$
$$+\,\lambda E\;x\text{^}\sim P\;x\text{^}\;[(||\nabla\;x\text{^}\;D(x\text{^})||2\;-\;1)2]\qquad(3.9),$$

де λ - коефіцієнт штрафу (зазвичай λ = 10), а x^ - точки, отримані шляхом інтерполяції між реальними та згенерованими зразками.

У контексті генерації текстур GAN-WP демонструє кілька важливих переваг. По-перше, стабільність навчання дозволяє використовувати більш глибокі архітектури, що покращує якість деталізації текстур [83]. По-друге, градієнтний штраф запобігає проблемі насичення градієнтів, що особливо важливо при роботі з текстурами високої роздільної здатності.

Процес навчання GAN-WP для генерації текстур включає кілька ключових модифікацій стандартного алгоритму. Замість класичного дискримінатора використовується критик, який не має сигмоїдного шару на виході і навчається оцінювати "реалістичність" текстур безпосередньо у просторі ознак [84]. Функція генератора при цьому залишається схожою до DCGAN, але з додатковими архітектурними покращеннями для стабільності навчання (3.10):

$$G(z) = tanh(Conv\,T\,(ReLU(\ldots Conv\,T\,(ReLU(FC(z)))\ldots))) \quad (3.10)$$

Особливістю GAN-WP при генерації текстур для ігрових персонажів є його здатність краще зберігати глобальну структуру та локальні деталі одночасно. Це досягається завдяки більш стабільному градієнтному потоку та кращій збіжності навчання [85]. При роботі з піксельними текстурами, характерними для Minecraft, GAN-WP демонструє кращу здатність до збереження чітких границь та дискретних колірних переходів порівняно з класичними GAN.

3.1.5 Дифузійні моделі

Дифузійні моделі представляють собою новий підхід до генеративного моделювання, який базується на поступовому додаванні та видаленні шуму з даних [86]. На відміну від GAN, які навчаються через змагальний процес, дифузійні моделі використовують принцип зворотної дифузії для генерації зображень [87].

Процес генерації в дифузійних моделях складається з двох основних етапів. На першому етапі (forward process) до вхідних даних поступово додається гаусівський шум через фіксовану послідовність кроків. На другому етапі (reverse process) модель навчається поступово видаляти цей шум для отримання осмисленого зображення [88].

Математично forward process можна описати як (3.11):

$$q(x_t | x_{t-1}) = N(x_t; sqrt(1 - \beta_t)x_{t-1}, \beta_t I) \quad (3.11),$$

де β_t - графік додавання шуму, а x_t - зашумлене зображення на кроці t.

U-Net архітектура, яка широко використовується в дифузійних моделях, була модифікована для роботи з умовною генерацією через додавання часових ембедінгів [89]. Це дозволяє моделі враховувати рівень шуму на кожному кроці генерації та поступово покращувати якість зображення.

Однією з ключових переваг дифузійних моделей є їх стабільність у навчанні та висока якість генерації. На відміну від GAN, вони не страждають від проблеми колапсу мод та забезпечують кращу різноманітність

згенерованих зразків [90]. Проте основним недоліком є повільність процесу генерації через необхідність виконання багатьох кроків дифузії.

Для задачі генерації текстур Minecraft дифузійні моделі виявились особливо цікавими через їх здатність зберігати дрібні деталі та текстурні патерни. Завдяки поступовому процесу денойзингу, модель може краще вловлювати ієрархічну структуру піксельної графіки [91].

3.1.6 Особливості застосування генеративних моделей для створення ігрових текстур

Застосування генеративних моделей для створення ігрових текстур має ряд специфічних особливостей, які відрізняють цю задачу від загальної проблеми генерації зображень. Як зазначають дослідники [92], ігрові текстури повинні відповідати строгим технічним вимогам, включаючи обмеження розміру, формату та сумісності з ігровими рушіями.

Одним з ключових аспектів є необхідність збереження стильової єдності з візуальною естетикою гри. У дослідженні [93] показано, що генеративні моделі повинні бути спеціально адаптовані для роботи з конкретним візуальним стилем. Наприклад, при створенні текстур для Minecraft критично важливо зберігати піксельну природу зображень та працювати з обмеженою колірною палітрою.

Особливу увагу при розробці генеративних моделей для ігрових текстур необхідно приділяти проблемі тайлінгу. Згідно з дослідженням [94], текстури повинні мати можливість безшовного з'єднання при повторенні, що вимагає спеціальних архітектурних рішень у генеративних моделях. Це особливо важливо для текстур оточення та ландшафту.

При створенні текстур для ігрових персонажів виникає додаткова складність, пов'язана з необхідністю враховувати UV-розгортку моделі. Як

показано в роботі [95], генеративні моделі повинні враховувати топологію 3D-моделі та забезпечувати коректне відображення текстури на всіх частинах персонажа. Це вимагає розробки спеціалізованих архітектур, які можуть працювати з UV-картами та враховувати просторові залежності між різними частинами текстури.

Важливим аспектом є також оптимізація продуктивності. Дослідження [96] демонструє, що для практичного застосування в ігровій індустрії генеративні моделі повинні бути здатні створювати текстури за прийнятний час, що часто вимагає компромісу між якістю та швидкістю генерації. Це особливо актуально для систем, що працюють в реальному часі або для інструментів розробки ігор.

Сучасні дослідження також звертають увагу на проблему контрольованої генерації. За даними [97], розробникам ігор часто потрібна можливість точного контролю над параметрами згенерованих текстур, такими як кольорова схема, рівень деталізації або специфічні візуальні елементи. Це призвело до розвитку умовних генеративних моделей, які можуть приймати додаткові параметри керування.

3.2 Збір та підготовка даних

3.2.1 Методологія скрапінгу текстур Minecraft

Для збору текстур персонажів гри Minecraft було розроблено спеціалізований скрипт на мові Python з використанням бібліотек pandas для обробки даних, requests для HTTP-запитів, BeautifulSoup для парсингу HTML та concurrent.futures для паралельного виконання запитів [98].

Основним джерелом даних було обрано платформу PlanetMinecraft - найбільший користувацький ресурс з текстурами для Minecraft. Для

забезпечення ефективного збору даних було реалізовано багатопотокове рішення з використанням ThreadPoolExecutor, що дозволило значно прискорити процес скрапінгу. Паралельна обробка була організована шляхом розділення загального масиву URLs на 10 частин, кожна з яких оброблялася окремим екземпляром скрипту на платформі Kaggle.

Процес збору даних включав кілька ключових етапів. Спочатку відбувався запит до веб-сторінки з текстурою, де з HTML-розмітки витягувалась наступна інформація: пряме посилання на файл текстури, теги, що описують текстуру, та додаткова метаінформація про пост. Для кожного запиту використовувались користувацькі заголовки для імітації браузера, що забезпечило стабільність роботи скрипту. В таблиці 3.1 наведено приклад даних, що були зібрані протягом першої стадії скрапінгу даних:

Таблиця 3.1 - Приклад даних, що були зібрані

	Link	Text	page_text	download_links	tags
0	https://www.planetminecraft.com/skin/attack-on...	Attack On Titan Boy *-*-*-*-* ErenJeager_ Skin	['Give Me a Like! (Only If You want!)', 'Male'...	https://static.planetminecraft.com/files/resou...	['Boy', 'Titan', 'Shingeki', 'Eren', 'Shingeki...
1	https://www.planetminecraft.com/skin/person-fr...	Person from family guy episode ("Dog Universe...	['Female', 'Java', 'Steve']	https://static.planetminecraft.com/files/resou...	['Cartoon']
2	https://www.planetminecraft.com/skin/happy-hal...	Happy halloween!	['Other', 'Java', 'Alex']	https://static.planetminecraft.com/files/resou...	['Other']
3	https://www.planetminecraft.com/skin/lotso-toy...	Lotso ("Toy story 3")	['Male', 'Java', 'Steve']	https://static.planetminecraft.com/files/resou...	['Cartoon']
4	https://www.planetminecraft.com/skin/venom-boy...	Venom Boy *-*-*-* ErenJeager_ Skin	['mmmmm Thanks For Watch My Skin Creation!', '...	https://static.planetminecraft.com/files/resou...	['Boy', 'Guy', 'Venom', 'Symbiote', 'Other']

Для обробки отриманих даних було реалізовано систему екстракції інформації з різних елементів веб-сторінки. Зокрема, було використано регулярні вирази для пошуку URL текстури в JSON-даних сторінки, а окрема частина коду витягувала метадані з відповідних HTML-елементів.

Загальний час виконання скрапінгу склав 26526.4 секунд (приблизно 7.4 години) для обробки близько 775,420 URLs, що в середньому становить 2.94 посилань за секунду. Отримані дані зберігалися у форматі CSV з наступними колонками: Link (посилання на сторінку текстури), Text (назва текстури), page_text (метадані сторінки), download_links (пряме посилання на файл текстури) та tags (теги, що описують текстуру), що наведені в таблиці 3.1.

Після збору метаданих та посилань на текстури було розроблено систему для масового завантаження зображень. Процес завантаження був також реалізований з використанням багатопотокової архітектури, що дозволило значно прискорити процес.

Для ефективного завантаження великої кількості файлів було використано 20 паралельних потоків. Кожен потік самостійно обробляв окремі URL-адреси, завантажував зображення та зберігав їх у визначену директорію. Система була налаштована на автоматичне відновлення після помилок завантаження, що забезпечило безперервність процесу навіть при тимчасових мережевих збоях.

Загалом було успішно завантажено 773,971 унікальних текстур. Процес завантаження тривав приблизно 70 хвилин (4,202 секунди), що в середньому становить 166 завантажень за секунду. Всі текстури були збережені у їхньому оригінальному форматі та роздільній здатності для подальшої обробки та аналізу. Приклад текстур, що були завантажені наведено на рисунку 3.1:

Рисунок 3.1 - Приклад завантажених текстур

3.2.2 Опис та аналіз зібраного датасету

Було проведено детальний аналіз зібраного датасету текстур для Minecraft. В результаті обробки було встановлено, що загальний обсяг датасету складає 773,960 унікальних текстур, що дозволило створити один з найбільших спеціалізованих наборів даних для текстур ігрових персонажів.

Було виконано аналіз розподілу розмірів текстур в наборі даних, результати якого представлені у таблиці 3.2:

Таблиця 3.2 - Розподіл розмірів текстур у датасеті

Ширина	Висота	Кількість
64	64	705,837
128	128	50,789
64	32	16,704
128	64	630

В результаті аналізу було виявлено, що 91.2% текстур мають стандартний розмір 64x64 пікселі, що відповідає базовому формату текстур Minecraft. Також було встановлено, що 2.2% текстур мають розмір 64x32 пікселі, які використовуються для "тонких" скінів, характерних для Alex-моделі персонажа. Розподіл висоти текстур наведено на рисунку 3.2:

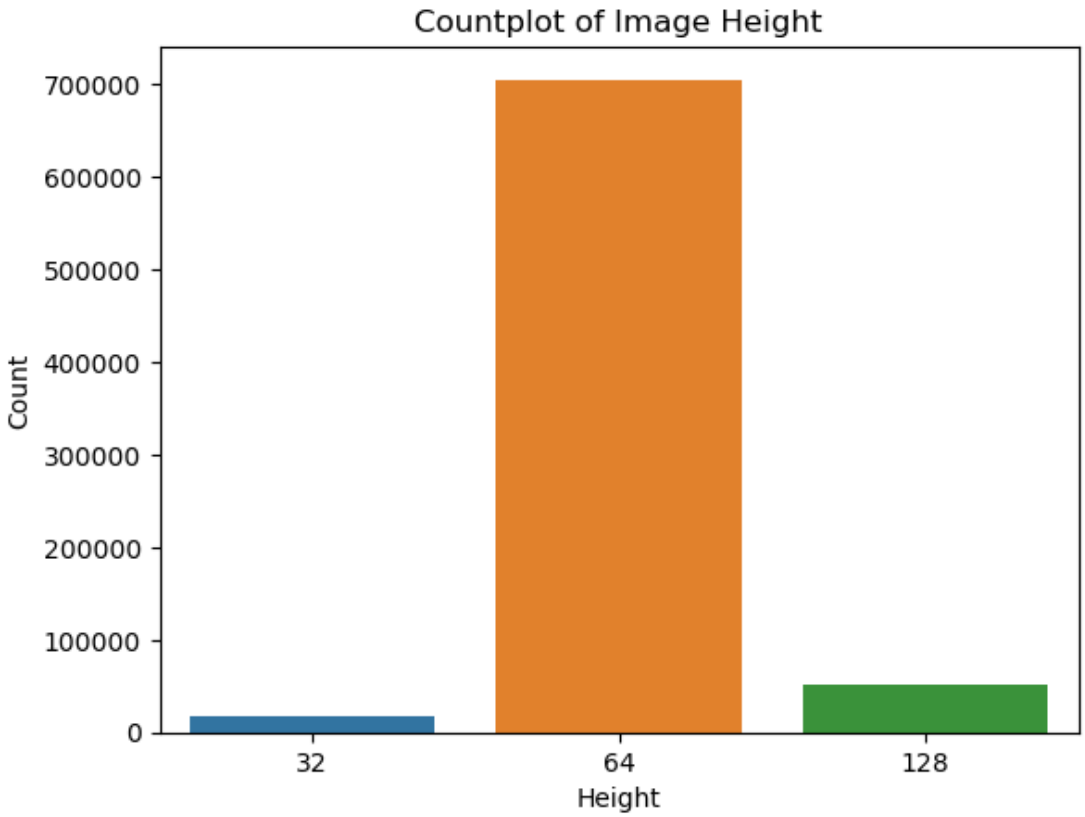

Рисунок 3.2 - Розподіл текстур в датасеті

Було проведено дослідження форматів зображень у датасеті, результати якого наведено у таблиці 3.3:

Таблиця 3.3 - Розподіл форматів зображень у датасеті

Формат	Кількість
P	558,708
RGBA	206,948
LA	6,651
L	1,346
1	193
RGB	114

В ході аналізу було встановлено, що палітрований формат (P) є домінуючим, складаючи 72.2% всіх текстур. Було визначено, що цей формат є оптимальним для піксельної графіки Minecraft, оскільки забезпечує ефективне зберігання зображень з обмеженою кількістю кольорів. Додатково було виявлено, що формат RGBA займає другу позицію за поширеністю (26.7%), забезпечуючи підтримку прозорості.

Було встановлено, що 770,999 текстур (99.6% від загального числа) підтримують прозорість, що було визначено як критичний параметр для коректного відображення текстур на 3D-моделі персонажа в грі.

3.2.3 Очищення текстур

Було проведено детальний аналіз структури текстур Minecraft, в результаті якого було виявлено, що певні частини текстури містять додаткові елементи, такі як підписи авторів та декоративні елементи, які не використовуються в грі при накладанні текстури на 3D-модель персонажа.

Для вирішення цієї проблеми було розроблено систему попередньої обробки даних, яка автоматично видаляє небажані області з текстур. Було створено спеціальну маску розміром 32x64 пікселі, яка визначає області, що потребують очищення. Області очищення було визначено на основі схеми розгортки 3D-моделі персонажа Minecraft та включають наступні координати (табл. 3.4):

Таблиця 3.4 - Координати областей очищення текстур

Координати (x1, y1, x2, y2)	Призначення області
(0, 0, 8, 8)	Верхня частина голови
(24, 0, 40, 8)	Верхня частина тіла
(56, 0, 64, 8)	Верхня частина ніг
(56, 16, 64, 32)	Нижня частина ніг
(0, 16, 4, 20)	Бокова частина голови
(12, 16, 20, 20)	Бокова частина рук
(36, 16, 44, 20)	Бокова частина тіла
(52, 16, 56, 20)	Бокова частина ніг

Було розроблено спеціалізований клас трансформації даних, який автоматично застосовує маску до кожної текстури, встановлюючи значення пікселів у визначених областях до нуля. Додатково було реалізовано функціонал для роботи з альфа-каналом зображень, що дозволило зберегти прозорість текстур після обробки.

Цей процес було застосовано до всіх текстур у датасеті, що дозволило отримати очищений набір даних, готовий для подальшого навчання моделей.

Обробка була виконана окремо для текстур стандартної моделі Steve та тонкої моделі Alex, враховуючи їх різні розміри та особливості розгортки. Приклад очищених текстур наведено на рисунку 3.3:

Рисунок 3.3 - Приклад очищених картинок

3.2.4 Аналіз симетричності частин текстур

Далі проведено аналіз симетричності текстур. В ході дослідження було виявлено, що значна частина текстур має асиметрію між лівою та правою частинами моделі. Для визначення областей аналізу було створено маску регіонів, як показано на рисунку 3.4:

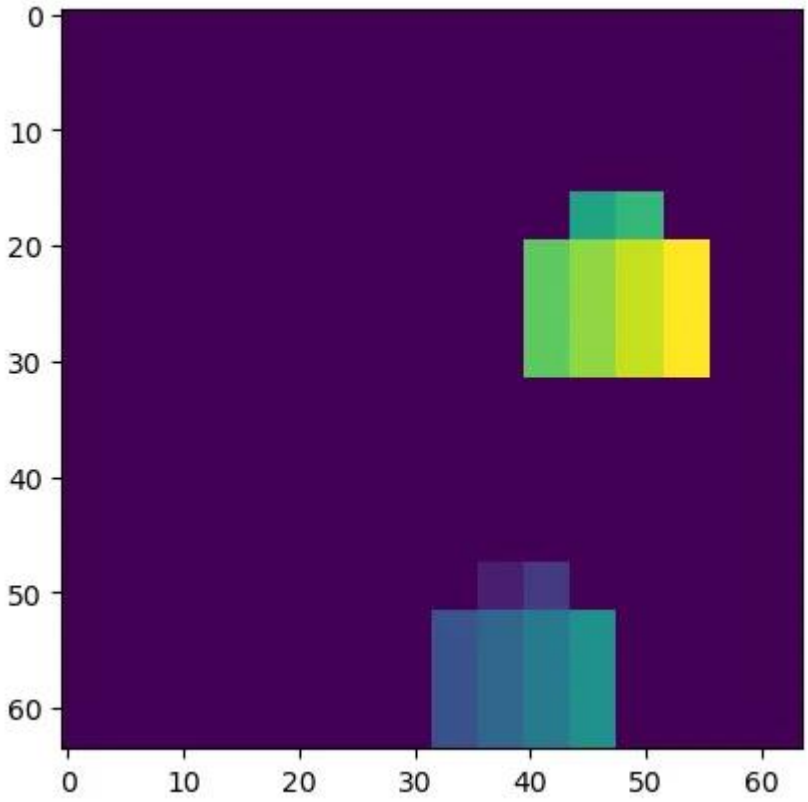

Рисунок 3.4 - Маска регіонів аналізу симетрії текстур

На прикладі конкретної текстури було продемонстровано проблему асиметрії. Було виділено 12 регіонів, які попарно відповідають лівій та правій частинам моделі. Як видно на рисунку 3.5, регіони, що відповідають рукам персонажа, демонструють значні відмінності у текстурі:

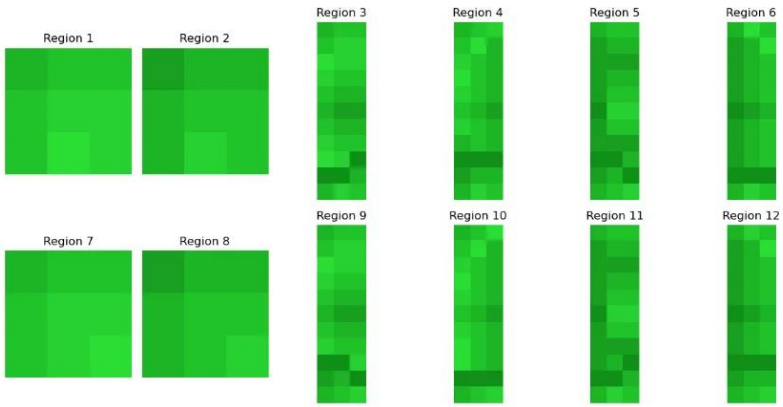

Рисунок 3.5 - Приклад асиметричних регіонів текстури персонажа

Для кількісної оцінки асиметрії було використано метрику структурної схожості (SSIM). Було проаналізовано розподіл значень SSIM окремо для рук та ніг персонажів. Результати аналізу для текстур рук представлено на рисунку 3.6:

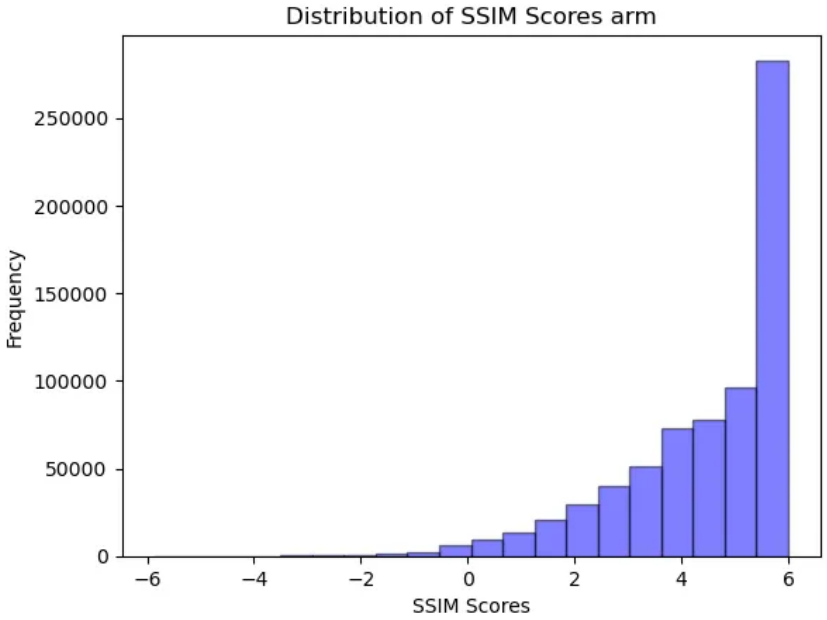

Рисунок 3.6 - Розподіл показників SSIM для регіонів рук

Аналогічний аналіз було проведено для регіонів ніг, результати якого показано на рисунку 3.7:

Рисунок 3.7 - Розподіл показників SSIM для регіонів ніг

В результаті аналізу було виявлено, що більшість текстур має високі показники SSIM (близько 6.0), що свідчить про хорошу симетрію. Проте існує помітна кількість текстур з низькими показниками (менше 4.0), які потребують корекції. Для виправлення асиметрії було прийнято рішення про відображення відповідних регіонів текстури для забезпечення симетричності моделі. Цей процес було автоматизовано та застосовано до всього набору даних.

На основі розрахованих показників SSIM було проведено класифікацію текстур за рівнем симетричності. Для наочної демонстрації було виділено три характерні групи текстур з різними діапазонами показників SSIM.

На рисунку 3.8 представлено текстури з найнижчими показниками симетричності. В цих текстурах спостерігаються значні відмінності між лівою та правою частинами, включаючи різні кольори, патерни та декоративні елементи. Такі текстури потребують суттєвої корекції для забезпечення симетричності.

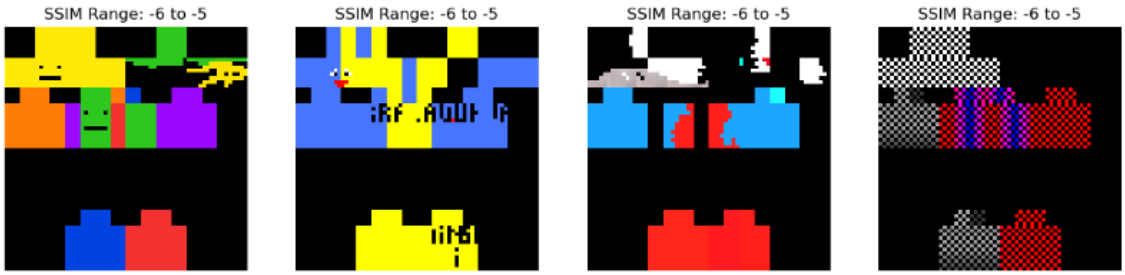

Рисунок 3.8 - Приклади текстур з низькою симетричністю (SSIM: -6 до -5)

На рисунку 3.9 показано текстури з високими показниками симетричності. Ці текстури демонструють майже ідеальну відповідність між лівою та правою частинами, що є бажаним результатом для ігрової моделі.

Рисунок 3.9 - Приклади текстур з високою симетричністю (SSIM: 6 до 7)

Текстури із середніми показниками SSIM, представлені на рисунку 3.10, мають помірні відмінності між симетричними частинами. Хоча ці відмінності менш виражені порівняно з першою групою, такі текстури також потребують корекції для покращення загальної якості набору даних.

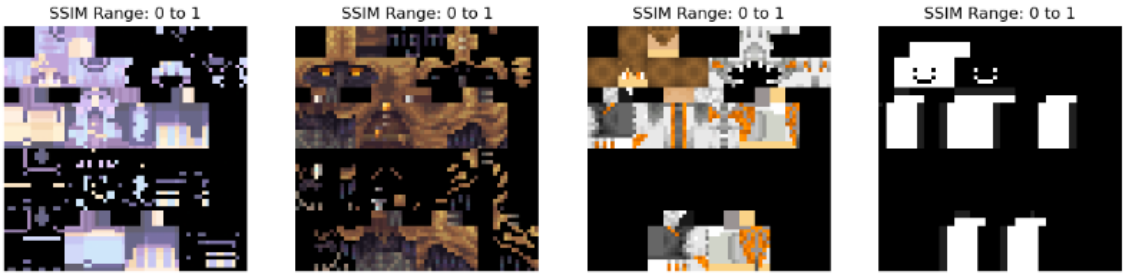

Рисунок 3.10 - Приклади текстур з середньою симетричністю (SSIM: 0 до 1)

3.2.5 Аналіз домінуючих кольорів

Далі було проведено аналіз домінуючих кольорів у текстурах з використанням алгоритму кластеризації K-means. Для кожної текстури було визначено три основні кольори, які потім було класифіковано за допомогою стандартної колірної номенклатури CSS3.

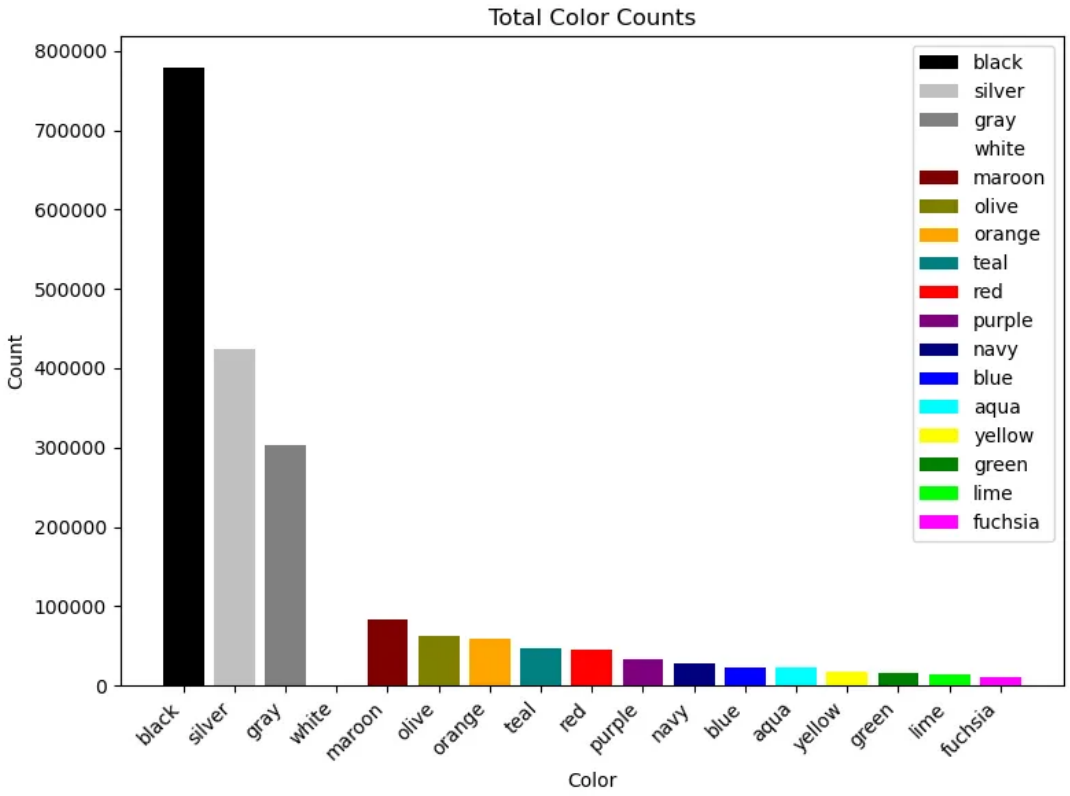

Рисунок 3.11 - Розподіл домінуючих кольорів у текстурах

Як видно з графіка на рисунку 3.11, найбільш поширеним кольором у текстурах є чорний (близько 770,000 входжень), що пов'язано з його використанням для контурів та тіней в текстурах. Наступними за частотою використання є відтінки сірого: сріблястий (420,000 входжень) та сірий (300,000 входжень), які часто використовуються для створення базової колірної схеми текстур.

Кольорові акценти представлені більш рівномірно, з переважанням теплих тонів:

- бордовий (maroon) та оливковий (olive) - близько 80,000 входжень кожен;
- помаранчевий (orange) та бірюзовий (teal) - приблизно 50,000 входжень;
- червоний (red), фіолетовий (purple) та темно-синій (navy) - близько 40,000 входжень.

Найменш представленими виявились яскраві кольори:

- жовтий (yellow);

- зелений (green);
- лаймовий (lime);
- фуксія (fuchsia).

Такий розподіл кольорів відображає типові патерни дизайну скінів для Minecraft, де базові кольори використовуються для основних елементів текстури, а яскраві кольори застосовуються для акцентів та деталей. Ця інформація є важливою для навчання генеративних моделей, оскільки дозволяє забезпечити реалістичний розподіл кольорів у згенерованих текстурах.

3.2.6 Аналіз повноти текстур

Було проведено аналіз повноти заповнення текстур для виявлення зображень з відсутніми або прозорими областями. Цей етап є критичним для підготовки якісного набору даних, оскільки неповні текстури можуть негативно вплинути на процес навчання моделі.

Для аналізу було розроблено алгоритм, який перевіряє наявність прозорих пікселів у важливих областях текстури з використанням спеціально створеної маски. Обробка проводилась паралельно з використанням багатопроцесорної архітектури, що дозволило ефективно проаналізувати весь набір із 773,971 текстур за 8 хвилин 35 секунд (в середньому 1,502 текстури за секунду).

Текстури було класифіковано на три категорії:

1. "Complete" - повністю заповнені текстури без прозорих областей
2. "Non Complete" - текстури з прозорими областями в критичних зонах
3. "Mismatch" - текстури з невідповідністю розмірів маски та зображення

Результати аналізу наведено на рисунку 3.12:

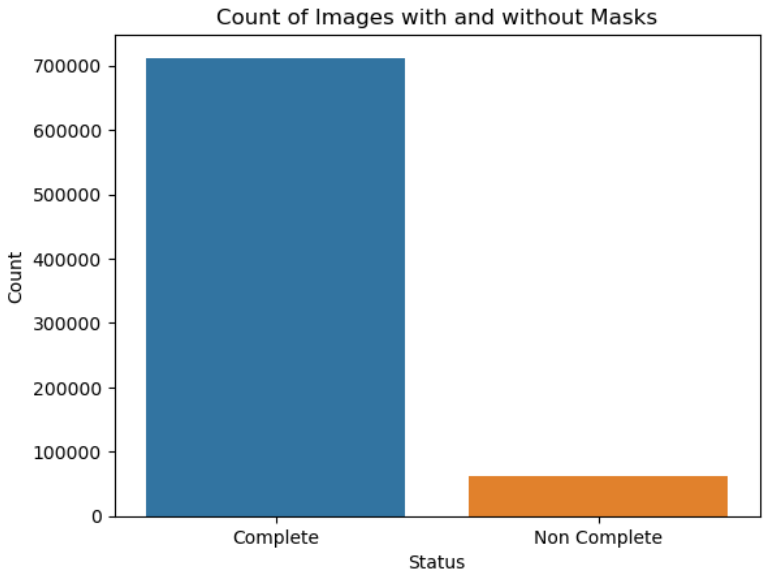

Рисунок 3.12 - Розподіл класів

Цей аналіз дозволив відфільтрувати неповні текстури для подальшої обробки та навчання моделей, забезпечуючи високу якість вхідних даних.

Такий підхід до валідації текстур є важливим етапом попередньої обробки даних, оскільки дозволяє уникнути потенційних проблем при генерації нових текстур, пов'язаних з неповним заповненням важливих областей моделі персонажа.

3.2.7 Аналіз наявності другого шару текстур

Було проведено аналіз наявності другого шару в текстурах Minecraft, який зазвичай використовується для створення додаткових елементів одягу або аксесуарів персонажа. Для цього було розроблено алгоритм підрахунку прозорих пікселів у визначених областях текстури.

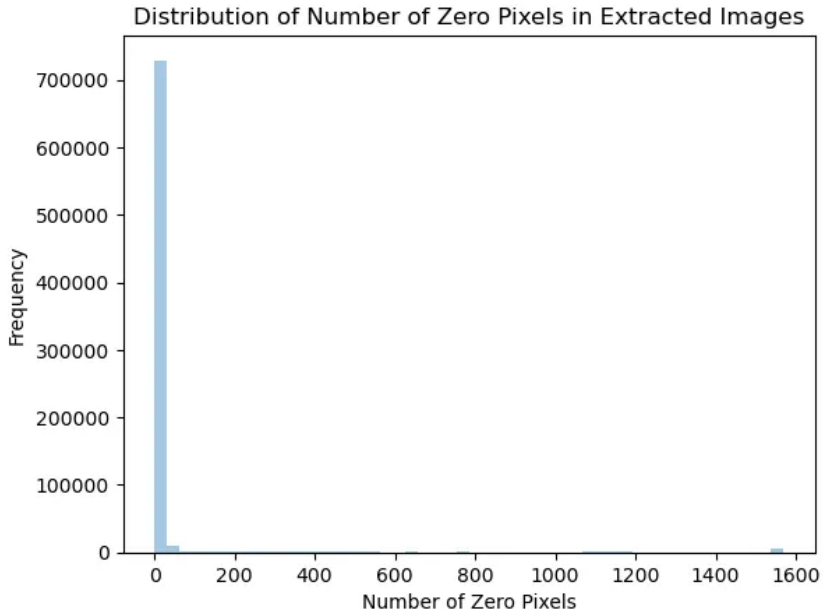

Рисунок 3.13 - Розподіл кількості прозорих пікселів у другому шарі текстур

Як видно з графіка на рисунку 3.13, більшість текстур (приблизно 700,000) мають нульову кількість прозорих пікселів у другому шарі, що свідчить про повне використання цього шару. Це є позитивним показником, оскільки вказує на те, що автори текстур ефективно використовують доступний простір для деталізації.

Невелика частина текстур має від 200 до 1600 прозорих пікселів, що може вказувати на:

- часткове використання другого шару для створення специфічних ефектів;
- незавершені текстури;
- спеціально залишені прозорі області для певних дизайнерських рішень.

Аналіз проводився з використанням багатопроцесорної обробки, що дозволило обробити всі 773,971 текстур за 8 хвилин 37 секунд (приблизно 1,495 текстур за секунду). Результати було збережено в структурі даних, що дозволяє ідентифікувати текстури, які потребують додаткової уваги або оптимізації.

3.2.8 Аналіз дублікатів

Було проведено пошук та аналіз дублікатів текстур у датасеті з використанням хешування. Для кожної текстури було створено два типи хешів:

1) head_hash: хеш для області голови персонажа;
2) full_skin_hash: хеш для повної текстури.

Розподіл наведено на рисунку 3.14:

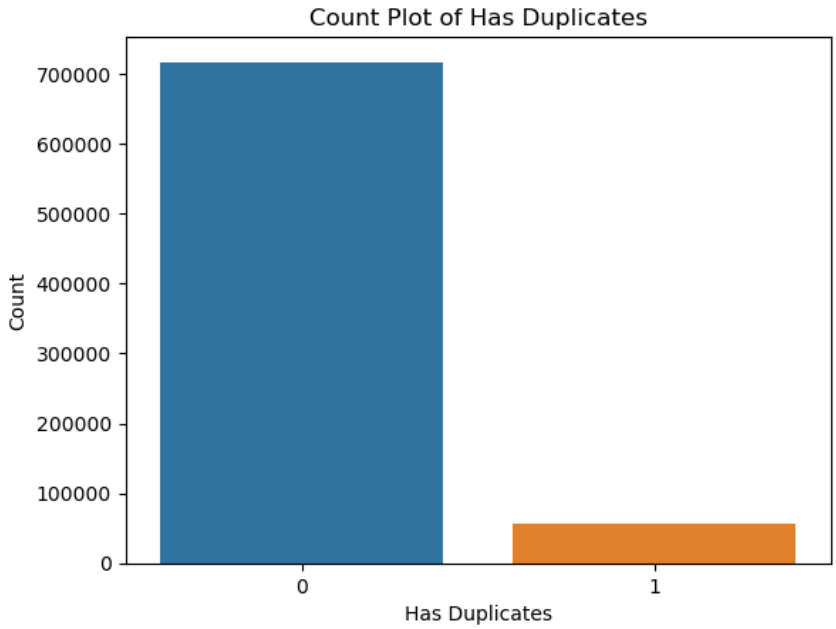

Рисунок 3.14 - Розподіл унікальних та повторюваних текстур

Як показано на рисунку 3.14, більшість текстур (близько 700,000) є унікальними і не мають дублікатів у датасеті. Приблизно 50,000 текстур було ідентифіковано як дублікати. Це важливе спостереження, оскільки дублікати можуть вплинути на якість навчання моделей та призвести до зміщення в результатах генерації.

Для ідентифікації дублікатів було використано повний хеш текстури (full_skin_hash), який представляє собою унікальний ідентифікатор, що враховує всі пікселі зображення. Кожен хеш має довжину 1024 символи, що забезпечує достатню точність для виявлення навіть незначних відмінностей між текстурами.

Було створено спеціальну колонку 'Has Duplicates' у наборі даних, де значення 0 вказує на унікальну текстуру, а 1 - на наявність дублікатів. Ця інформація є важливою для подальшої фільтрації даних перед навчанням моделей, щоб уникнути перенавчання на повторюваних зразках.

3.2.9 Фільтрація текстур за якісними характеристиками

Було проведено аналіз та фільтрацію текстур за трьома основними критеріями якості. Для забезпечення високої якості навчального набору даних, було розроблено комплексну систему оцінки текстур.

Було реалізовано оцінку контрастності зображення через розрахунок різниці між максимальним та мінімальним значенням пікселів. Текстури з контрастністю нижче порогового значення 120 було позначено як "Low Contrast". Як показано на рисунку 3.15, такі текстури складаються переважно з темних відтінків, що створює недостатню різницю між елементами текстури та ускладнює їх розрізнення при відображенні в грі:

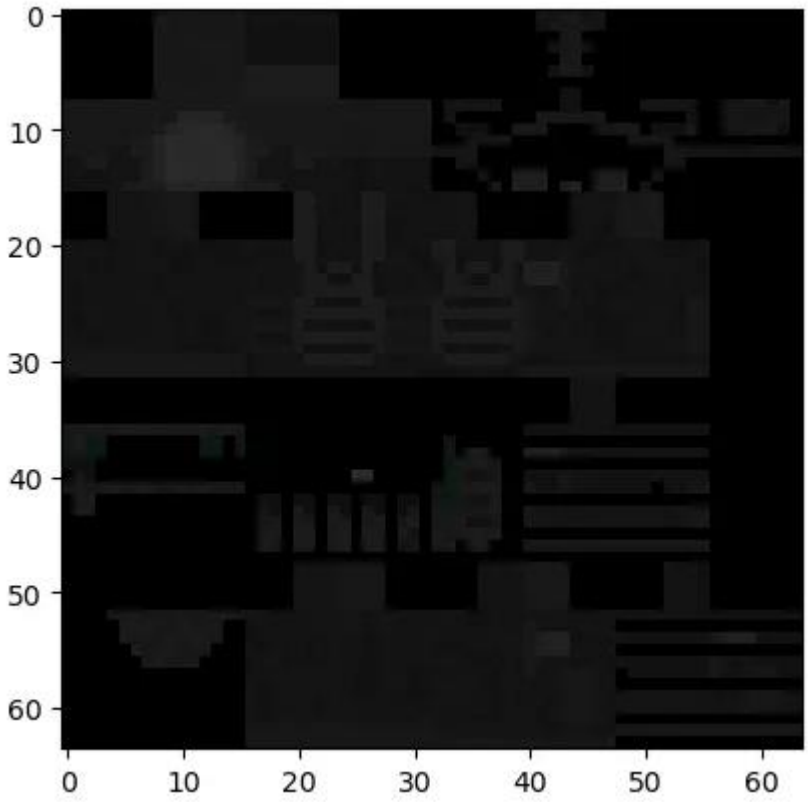

Рисунок 3.15 - Приклад текстури з низьким контрастом

Наступним параметром стала різноманітність кольорів. Було реалізовано аналіз варіативності кольорів через розрахунок стандартного відхилення значень яскравості пікселів для непрозорих областей текстури. В результаті аналізу було визначено оптимальний діапазон варіативності від 10 до 90. Текстури з показниками нижче 10 вважаються занадто монотонними, а вище 90 - надмірно контрастними.

Важливим критерієм також стала складність текстури, яка оцінювалася через кількість унікальних значень яскравості. На рисунку 3.16 представлено приклад "простої" текстури з менш ніж 10 унікальними значеннями.

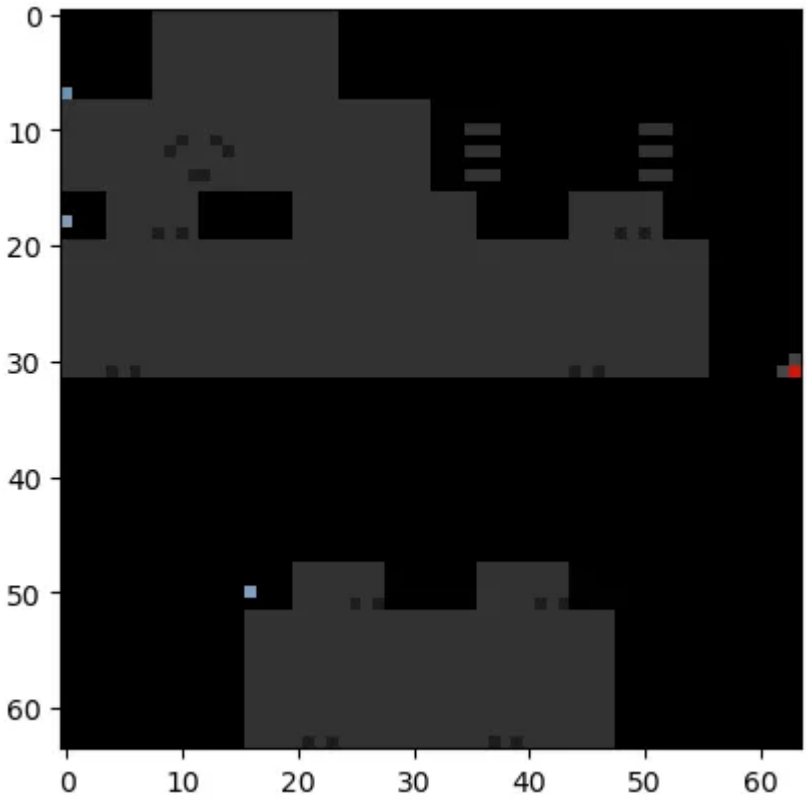

Рисунок 3.16 - Приклад простої текстури з низькою деталізацією

Як видно з рисунка 3.16, такі текстури мають мінімальну деталізацію та обмежену колірну палітру, що робить їх непридатними для навчання генеративних моделей.

Цей багатокритеріальний аналіз дозволив виявити та відфільтрувати текстури, які можуть негативно вплинути на якість навчання моделей генерації. Відібрані текстури мають оптимальний баланс контрастності, різноманітності кольорів та складності, що забезпечує якісну основу для тренування генеративних моделей.

3.2.10 Результати класифікації текстур

На основі проведеного аналізу було створено комплексну систему класифікації текстур за різними параметрами, які наведено у таблиці 3.5:

Таблиця 3.5 - Параметри класифікації текстур Minecraft

Параметр	Можливі значення	Опис
Gender	Male, Female, Other	Стать персонажа
Version	Java	Версія гри
Type	Steve, Alex	Тип моделі персонажа
Tag Color	Other	Колірна категорія
Status	Complete, Non Complete	Статус заповнення текстури
Num Zero Pixels 2nd Layer	0-1600	Кількість прозорих пікселів у другому шарі
Has Duplicates	0, 1	Наявність дублікатів
Contrast	Low Contrast, Normal Contrast	Рівень контрастності
Brightness Variety	Low Deviation, Normal Deviation, High Deviation	Варіативність яскравості
Brightness Uniqueness	Simple, Complex	Складність текстури
total_tags	1+	Кількість тегів
Classes	human, other, gaming, robot, cartoon, animal	Категорія персонажа

Така багатовимірна класифікація дозволяє ефективно фільтрувати та відбирати текстури для різних задач генерації. Наприклад, для навчання моделі можна відібрати текстури, що відповідають наступним критеріям: Complete статус, Normal Contrast, Normal Deviation, Complex складність, без дублікатів та з повністю заповненим другим шаром (Num Zero Pixels 2nd Layer = 0).

3.3 Розробка та навчання моделей

3.3.1 Реалізація GAN

Для генерації текстур Minecraft було розроблено архітектуру генеративно-змагальної мережі (GAN). Основна модель складається з генератора та дискримінатора, які навчаються в змагальному режимі.

Генератор побудовано на основі згорткової нейронної мережі з поступовим збільшенням просторової розмірності, як показано на рисунку 3.17:

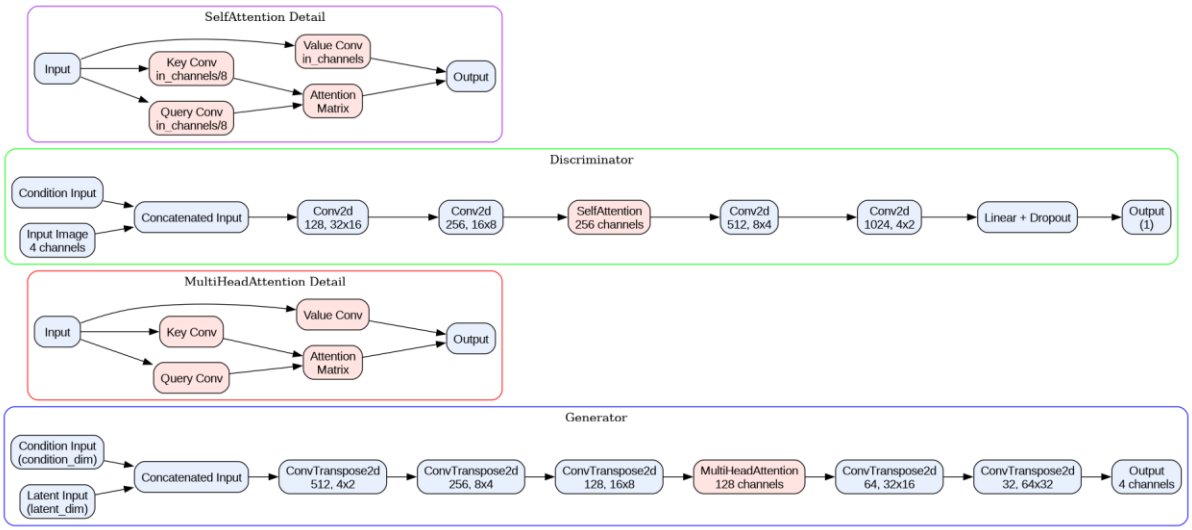

Рисунок 3.17 - Архітектура розробленої моделі GAN

Архітектура включає п'ять основних блоків розширення, кожен з яких складається з шару транспонованої згортки, пакетної нормалізації та функції активації LeakyReLU.

Особливістю розробленої архітектури є використання механізму Multi-Head Attention після третього блоку розширення. Цей механізм дозволяє мережі краще вловлювати довгострокові залежності в текстурах. Attention модуль працює з 128 каналами та використовує 128 головок уваги, що забезпечує детальний аналіз різних аспектів вхідних даних.

Фінальні розміри згенерованої текстури складають 64x32 пікселі з 4 каналами (RGBA), що відповідає стандартному формату текстур Minecraft. Для отримання значень пікселів у потрібному діапазоні використовується функція активації Tanh на останньому шарі.

Дискримінатор, архітектура якого представлена на рисунку 3.15, має зворотну структуру з поступовим зменшенням просторової розмірності. Він включає чотири згорткові блоки, кожен з яких зменшує розмірність вдвічі. Для покращення стабільності навчання використовується механізм Self-Attention після другого згорткового блоку.

Важливою особливістю дискримінатора є використання dropout з високим коефіцієнтом (0.8) перед фінальним повнозв'язним шаром. Це допомагає запобігти перенавчанню та покращує узагальнюючу здатність мережі. Вихідний сигмоїдний шар перетворює результати в ймовірності для бінарної класифікації реальних та згенерованих текстур.

Для навчання моделі було використано функцію втрат (3.12):

$$L\_D = -E[log(D(x))] - E[log(1 - D(G(z)))]\, L\_G = E[log(D(G(z)))] \quad (3.12),$$

де x - реальні текстури, z - випадковий шум, G - генератор, D - дискримінатор.

Процес навчання проводився на графічному процесорі NVIDIA P 100 з використанням оптимізатора Adam з наступними параметрами:

1. Learning rate для генератора: 0.0002
2. Learning rate для дискримінатора: 0.0002
3. Beta1: 0.5
4. Beta2: 0.999

Розмір батчу було встановлено на 64 зображення, що забезпечило оптимальний баланс між швидкістю навчання та стабільністю градієнтів. Навчання проводилось протягом 100 епох з періодичним збереженням проміжних результатів для аналізу прогресу генерації

3.3.2 Реалізація C-GAN-WP

Для покращення контролю над процесом генерації текстур було розроблено модифіковану версію GAN з умовною генерацією та градієнтним штрафом Вассерштейна (Conditional GAN with Wasserstein penalty, C-GAN-WP). Архітектура моделі наведена на рисунку 3.18:

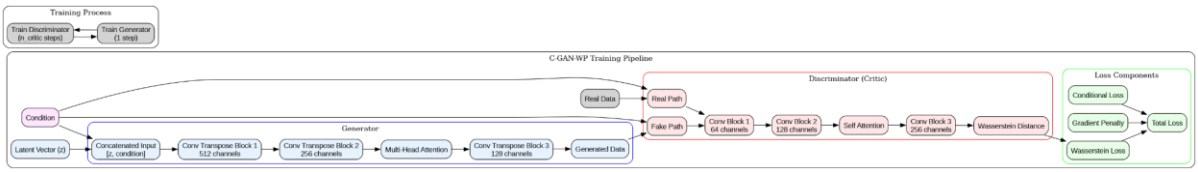

Рисунок 3.18 - Архітектура розробленої моделі C-GAN-WP

Як показано на рисунку 3.18, архітектура моделі включає додаткові входи для умовної інформації як у генераторі, так і в дискримінаторі.

Генератор C-GAN-WP приймає на вхід не тільки латентний вектор, але й вектор умов розмірністю condition_dim. Цей вектор кодує бажані характеристики текстури, такі як стиль, кольорова гама та тип персонажа.

Умовна інформація конкатенується з латентним вектором перед подачею на перший шар генератора. Архітектура розширена механізмом Multi-Head Attention з 128 головками, що дозволяє мережі краще враховувати взаємозв'язки між умовами та генерованими особливостями текстури. Архітектура наведена на рисунку 3.19:

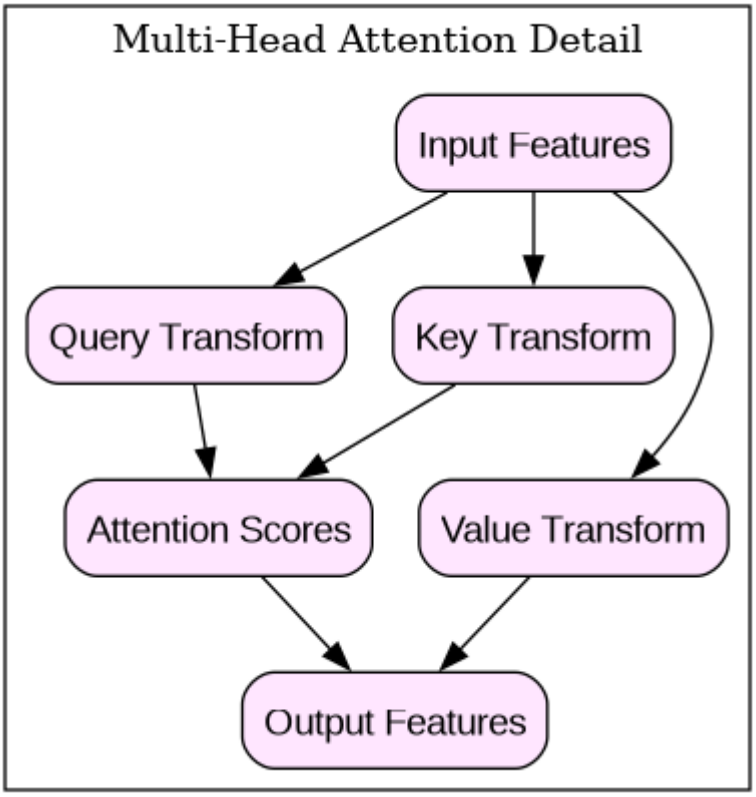

Рисунок 3.19 - Архітектура використаного алгоритму Multi-Head Attention

Дискримінатор також модифіковано для роботи з умовною інформацією. Вектор умов розширюється до просторових розмірів вхідної текстури та конкатенується з нею по каналах. Це дозволяє дискримінатору оцінювати не тільки реалістичність текстури, але й її відповідність заданим умовам.

Важливим удосконаленням є використання функції втрат Вассерштейна з градієнтним штрафом (3.13):

$$L\_critic = E[D(G(z,c))] - E[D(x,c)] + \\ + \lambda E[(||\nabla D(\hat{x},c)||_2 - 1)^2] \; L\_generato = -E[D(G(z,c))] \qquad (3.13),$$

де c - вектор умов, λ - коефіцієнт градієнтного штрафу (встановлено як 10), а $\hat{x}$ - точки, отримані інтерполяцією між реальними та згенерованими зразками.

Використання градієнтного штрафу замість відсікання вагів, яке застосовувалося в оригінальному WGAN, забезпечило більш стабільне навчання та кращу якість генерації. Для забезпечення умови Ліпшиця функція втрат включає штраф за відхилення норми градієнта від одиниці.

Навчання проводилося з використанням наступних гіперпараметрів:

1. Розмір батчу: 32
2. Learning rate: 0.0001
3. Beta1: 0.0
4. Beta2: 0.9
5. Кількість ітерацій критика на одну ітерацію генератора: 5

Модель продемонструвала значно кращу здатність до генерації текстур із заданими характеристиками порівняно з базовою GAN. Особливо помітним стало покращення у збереженні стильової єдності та кольорової гами згенерованих текстур.

3.3.3 Реалізація GAN-WP

В рамках дослідження було реалізовано модифіковану версію GAN з використанням метрики Вассерштейна та градієнтного штрафу (GAN-WP). На відміну від C-GAN-WP, ця архітектура не використовує умовну генерацію, що робить її більш простою в реалізації, але все ще зберігає переваги стабільного

навчання завдяки градієнтному штрафу. Архітектура моделі наведена на рисунку 3.20:

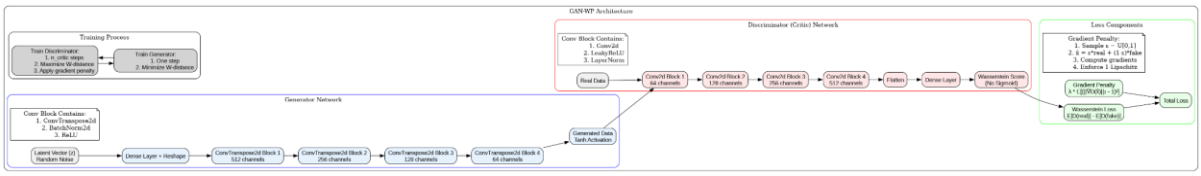

Рисунок 3.20 - Архітектура використаної моделі GAN-WP

Як показано на рисунку 3.20, генератор GAN-WP має подібну до базової GAN архітектуру з декількома ключовими модифікаціями. Основний потік даних проходить через послідовність блоків розширення з транспонованими згортками, але замість звичайної BatchNormalization використовується модифікована версія, яка краще працює з метрикою Вассерштейна. Механізм Multi-Head Attention, розміщений після третього блоку, тепер має модифіковану схему розрахунку ваг уваги, що краще відповідає принципам WGAN.

Дискримінатор, представлений на рисунку 1.20, в архітектурі GAN-WP називається критиком, оскільки він не класифікує зображення як реальні чи згенеровані, а оцінює "реалістичність" за метрикою Вассерштейна. Ключовою особливістю є відсутність сигмоїдного шару на виході, замість цього використовується лінійна активація. Це дозволяє мережі оцінювати "відстань" між розподілами реальних та згенерованих даних у просторі Вассерштейна.

Функція втрат для GAN-WP має наступний вигляд (3.14)-(3.15):

$$L\_critic = E[D(G(z))] - E[D(x)] + \lambda E[(||\nabla D(\hat{x})||_2 - 1)^2] \quad (3.14)$$

$$L\_generator = -E[D(G(z))] \quad (3.15)$$

де λ - коефіцієнт градієнтного штрафу, встановлений як 10, а $\hat{x}$ - точки, отримані шляхом інтерполяції між реальними та згенерованими зразками.

Для навчання моделі було використано наступну конфігурацію:

1. Оптимізатор: Adam з параметрами $\beta_1 = 0$ та $\beta_2 = 0.9$
2. Learning rate: 0.0001
3. Розмір батчу: 64
4. Кількість кроків генератора на один крок критика: 5

Градієнтний штраф застосовується тільки до критика і обчислюється для випадково інтерпольованих точок між реальними та згенерованими зразками. Це забезпечує виконання умови Ліпшиця та стабільність навчання.

GAN-WP продемонструвала кращу стабільність навчання порівняно з базовою GAN. Відсутність режиму колапсу та більш плавна збіжність процесу навчання дозволили отримати більш якісні результати генерації. Модель особливо добре справляється з відтворенням дрібних деталей текстур та забезпеченням різноманітності згенерованих зразків.

Аналіз результатів показав, що хоча GAN-WP не має можливості умовної генерації як C-GAN-WP, вона забезпечує більш стабільні та якісні результати порівняно з базовою GAN, що робить її хорошим вибором для задач неумовної генерації текстур.

3.3.4 Реалізація Diffusion

Було здійснено спробу реалізації моделі дифузії для генерації текстур Minecraft. Архітектура моделі базується на U-Net з механізмом уваги, як показано на рисунку 3.21:

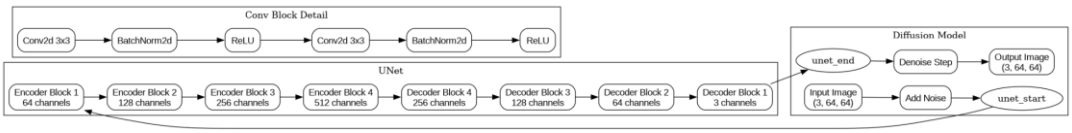

Рисунок 3.21 - Архітектура моделі Diffusion

Основними компонентами архітектури є енкодер з чотирма блоками понижуючої дискретизації (від 64 до 512 каналів), декодер з чотирма блоками підвищуючої дискретизації (від 512 до 4 каналів), конволюційні блоки з BatchNormalization та ReLU активацією, а також механізм умовної генерації через часові ембеддінги.

Процес тренування проводився на підмножині з 10,000 текстур розміром 64x64 пікселі. Основними параметрами навчання були: розмір батчу 64, початковий learning rate 0.001, оптимізатор Adam, scheduler ReduceLROnPlateau з коефіцієнтом зменшення 0.5 та 250 кроків дифузії.

Як видно з графіку навчання на рисунку 2.22, модель демонструвала стабільне зменшення функції втрат протягом перших 100 епох, після чого процес навчання сповільнився. Найкраще значення функції втрат склало 0.5173, яке було досягнуто на епосі 166.

Однак під час навчання було виявлено ряд суттєвих проблем. Передусім, висока обчислювальна складність значно обмежила можливості експериментів - процес навчання протягом 10 годин дозволив виконати лише 371 епоху, що виявилось недостатнім для повної збіжності моделі.

Також суттєвою проблемою стала нестабільність генерації. Згенеровані текстури часто містили артефакти та шуми, особливо в областях з різкими переходами кольорів. Модель мала особливі труднощі з відтворенням симетричних елементів текстури, що є критичним для естетики Minecraft.

Додатковим викликом стали проблеми з глобальною узгодженістю згенерованих текстур. Модель не змогла належним чином вловити та відтворити структурні залежності між різними частинами текстури, що призводило до порушення цілісності дизайну.

Через ці фундаментальні обмеження було прийнято рішення зосередитись на розвитку GAN-базованих архітектур, які продемонстрували кращу стабільність та якість результатів при менших обчислювальних витратах. Досвід роботи з дифузійною моделлю надав цінне розуміння

складності задачі генерації текстур та допоміг визначити найбільш перспективні напрямки подальших досліджень.

3.3.5 Аналіз отриманих результатів

Для порівняння ефективності різних архітектур було проведено серію експериментів з навчання моделей на наборі текстур Minecraft. Всі експерименти проводились на графічному процесорі NVIDIA A100, що дозволило оцінити не лише якість генерації, але й обчислювальну ефективність кожного підходу.

Рисунок 3.22 демонструє результати генерації текстур за допомогою diffusion моделі на епосі 115:

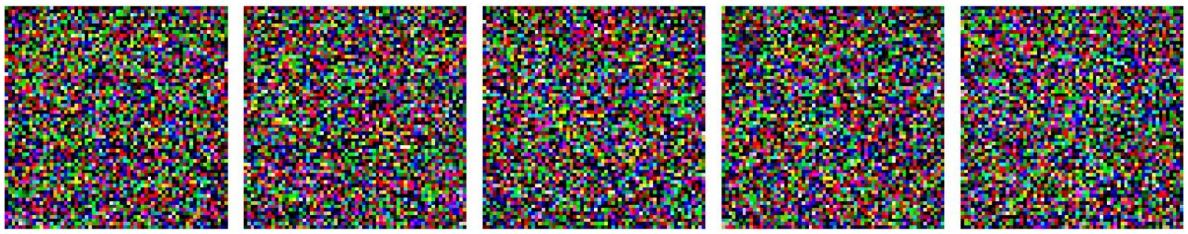

Рисунок 3.22 - Приклад згенерованих картинок на епосі 115

На згенерованих зображеннях можна помітити, що модель почала формувати певні регіони різної інтенсивності, проте текстура все ще залишається хаотичною та не має чіткої структури. Деякі області зображення затемнені, що свідчить про нестабільність процесу навчання на цьому етапі.

Після 10 годин тренування diffusion моделі (рисунок 3.23) результати дещо покращились - можна побачити формування певних регіонів, які відповідають різним частинам текстури:

Рисунок 3.23 - Приклад згенерованих в кінці тренування

Проте загальна якість генерації залишається незадовільною, зображення занадто темне та позбавлене деталей.

На рисунку 3.24 представлено проміжні результати тренування C-GAN-WP:

Рисунок 3.24 - Приклад згенерованих картинок посеред процесу тренування

Модель демонструє краще розуміння структури текстур Minecraft, проте присутня значна кількість візуальних артефактів. Цікаво відзначити наявність підсвічених клітинок, що є результатом роботи механізму self-attention.

Подальше тренування C-GAN-WP протягом 5 годин (рисунок 3.25) призвело до несподіваного погіршення якості - модель виявила ознаки перенавчання, що проявилось у появі більшої кількості артефактів та втраті структурної цілісності текстур:

Рисунок 3.25 - Приклад згенерованих картинок в кінці тренування

Найкращі результати було отримано при використанні GAN-WP архітектури (рисунок 3.26):

Рисунок 3.26 - Приклад згенерованих картинок черещ GAN-WP

Після 9 годин тренування модель генерує візуально правдоподібні текстури з чіткою структурою та мінімальною кількістю артефактів. Механізм self-attention, що проявляється у вигляді підсвічених клітинок, допомагає моделі краще вловлювати глобальні залежності в структурі текстур.

Експерименти показали, що хоча diffusion моделі є потужним інструментом для генерації зображень, вони потребують значно більше обчислювальних ресурсів та часу для досягнення прийнятних результатів. GAN архітектури, особливо з використанням механізму Wasserstein loss та gradient penalty, виявились більш ефективними для конкретної задачі генерації текстур Minecraft.

Висновки до розділу 3

Розглянуто основні архітектури генеративних нейронних мереж, включаючи GAN, Conditional GAN, GAN-WP, та дифузійні моделі, а також визначено їх переваги й обмеження для застосування у завданнях комп'ютерного зору. Ці методи дозволяють досягти високої якості згенерованих текстур, що є особливо цінним у розробці ігрового контенту.

Проведено комплексний збір та підготовку даних з текстур Minecraft, зокрема аналіз симетричності, домінуючих кольорів та наявності шарів. Це дало можливість отримати структурований та якісний датасет для подальшого навчання моделей. Було застосовано методи скрапінгу, фільтрації й очищення текстур, що забезпечило високу якість вхідних даних, необхідних для стабільного навчання генеративних моделей.

На основі аналізу було реалізовано декілька варіантів генеративних моделей: базовий GAN, C-GAN-WP, та GAN-WP. Кожна з них була адаптована з урахуванням специфіки задачі генерації текстур, зокрема було використано механізм Multi-Head Attention для кращого захоплення структурних особливостей текстур. Використання градієнтного штрафу у GAN-WP забезпечило стабільність і точність результатів, дозволяючи уникнути проблеми колапсу режимів.

РОЗДІЛ 4 РОЗРОБКА ВЛАСНОГО СТАРТАП ПРОЕКТУ

4.1 План розробки стартапу та масштабування його на ринок

Розглянемо план розробки стартапу з генерації текстур для комп'ютерних ігор на основі штучного інтелекту та його виведення на ринок.

Маркетинговий аналіз потребує дослідження наступних аспектів. Необхідно проаналізувати існуючі рішення для створення ігрових текстур та інструменти, якими користуються розробники ігор та моддери. На ринку існує кілька категорій конкурентів: класичні графічні редактори, спеціалізовані інструменти для створення текстур та нові AI-powered рішення. Цільова аудиторія включає три основні групи: інді-розробники ігор, які потребують швидкого створення великої кількості текстур; творці модифікацій для існуючих ігор, особливо Minecraft; та художники-текстурники, які можуть використовувати AI як інструмент для прискорення роботи.

Організація розробки стартапу починається з детального технічного планування. План розробки включає: вдосконалення існуючих моделей генерації для покращення якості текстур; створення зручного веб-інтерфейсу для завантаження референсів та налаштування параметрів генерації; розробку API для інтеграції з популярними ігровими движками та редакторами. Важливо забезпечити масштабованість серверної інфраструктури для обробки паралельних запитів на генерацію текстур.

Фінансово-економічний аналіз проекту має врахувати специфіку AI-стартапу. Основні витрати включають: оренду обчислювальних потужностей для тренування моделей та інференсу; розробку та підтримку веб-платформи; маркетинг та просування продукту в спільнотах розробників ігор. Монетизація може здійснюватися через комбінацію підписок з різними лімітами на

генерацію та додаткові функції, а також через продаж ліцензій для корпоративних клієнтів.

Стратегія комерціалізації базується на поетапному виході на ринок. Першим етапом є запуск бета-версії для спільноти Minecraft, що дозволить отримати цінний зворотній зв'язок та створити початкову базу користувачів. Далі планується розширення на інші піксельні ігри та поступовий перехід до підтримки текстур високої роздільної здатності для більш масштабних проектів. Важливим елементом стратегії є участь у геймдев конференціях та створення партнерств з розробниками популярних ігрових движків.

Для залучення інвестицій необхідно зосередитись на демонстрації унікальних технологічних переваг нашого рішення, таких як висока якість згенерованих текстур та можливість точного контролю над процесом генерації. Потенційними інвесторами можуть бути венчурні фонди, що спеціалізуються на AI/ML проектах та ігровій індустрії.

Успішна реалізація цього плану дозволить створити потужний інструмент для оптимізації процесу розробки ігор та вийти на швидкозростаючий ринок AI-інструментів для геймдеву.

4.2 Опис ідеї стартап-проекту

Стартап-проект спрямований на вирішення проблеми створення унікальних текстур для комп'ютерних ігор, зокрема для гри Minecraft, використовуючи технології штучного інтелекту. Основна ідея продукту полягає у створенні веб-сервісу, який дозволяє генерувати високоякісні текстури на основі заданих параметрів та референсів. Система надає користувачам можливість створювати унікальні текстури без глибоких знань у галузі комп'ютерної графіки.

Користувачі зможуть використовувати платформу для генерації різних типів текстур, контролювати стиль та параметри генерації, а також інтегрувати

згенеровані текстури безпосередньо у свої проекти. Алгоритми генерації базуються на сучасних архітектурах глибинного навчання та враховують специфічні вимоги ігрової графіки, такі як піксельний стиль та необхідність збереження структурної цілісності.

Далі в таблиці 4.1 наведена інформаційна карта даного стартапу:

Таблиця 4.1 – Інформаційна карта стартап-проекту

Назва розділу	Опис
Назва проекту	TextureAI
Автори проекту	Черкасов Євген Валерійович
Коротка анотація	Веб-сервіс для автоматизованої генерації ігрових текстур з використанням штучного інтелекту. Дозволяє створювати унікальні текстури для ігор у піксельному стилі з можливістю точного контролю над результатом.
Термін реалізації проекту	8 місяців
Необхідні ресурси	GPU-сервери для обчислень, хмарна інфраструктура, програмне забезпечення для розробки веб-інтерфейсу, доступ до наборів даних для навчання моделей
Фінансові кошти	Оренда обчислювальних потужностей, зарплата розробникам, витрати на маркетинг та підтримку інфраструктури
Опис проблеми, яку вирішує проект	Проект вирішує проблему трудомісткості створення унікальних текстур для ігор, надаючи інструмент для швидкої та якісної генерації текстур з використанням AI
Головні цілі та завдання проекту	Створення доступного та ефективного інструменту для генерації ігрових текстур, який дозволить розробникам та моддерам швидко створювати якісний контент
Очікувані результати	Створення популярної платформи для генерації текстур, форм

Цей проект не лише спрощує процес створення текстур, але й відкриває нові можливості для творчості в ігровій індустрії, роблячи його актуальним на ринку інструментів для розробки ігор. Особливу цінність проект представляє для інді-розробників та творців модифікацій, яким часто не вистачає ресурсів для створення великої кількості унікальних текстур.

4.3 Технологічний аудит ідеї проекту

Розглянемо технологічні аспекти стартап-проекту TextureAI та проведемо конкурентний аналіз для оцінки його позиції на ринку.

Ідея стартапу полягає у створенні веб-сервісу для генерації ігрових текстур з використанням генеративних нейронних мереж. В основі проекту лежить технологія GAN-WP, що забезпечує високу якість генерації та контроль над результатом. Детальний опис ідеї стартапу представлено в таблиці 4.2:

Таблиця 4.2 – Опис ідеї стартапу

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Створення веб-сервісу для автоматизованої генерації ігрових текстур на основі штучного інтелекту	1. Розробка інді-ігор 2. Створення модифікацій 3. Прототипування ігрових проектів	1. Швидке створення унікальних текстур 2. Економія ресурсів на графічному дизайні 3. Можливість експериментувати з різними стилями

Для оцінки конкурентної позиції проекту було проведено порівняльний аналіз з існуючими рішеннями на ринку, результати якого наведено в таблиці 4.3:

Таблиця 4.3 – Порівняльний аналіз конкурентів проекту

№	Характеристики	TextureAI	Artbreeder	Runway ML	Midjourney	W/ N/S
1	Спеціалізація на ігрових текстурах	Повна спеціалізація	Загальна генерація	Загальна генерація	Загальна генерація	S
2	Підтримка піксельної графіки	Оптимізовано	Обмежена	Відсутня	Обмежена	S
3	Контроль над результатом	Високий	Середній	Високий	Низький	N
4	Швидкість генерації	Висока	Середня	Середня	Низька	S
5	Вартість використання	Помірна	Висока	Висока	Висока	S

Технологічну здійсненність продукту було проаналізовано з точки зору наявності та доступності необхідних технологій, що представлено в таблиці 4.4:

Таблиця 4.4 – Технологічна здійсненність продукту

№	Ідея проекту	Технології реалізації	Наявність	Доступність
1	Генерація текстур	GAN-WP, PyTorch	Наявні	Доступні
2	Веб-інтерфейс	React, Node.js	Наявні	Доступні
3	Серверна інфраструктура	Docker, Kubernetes	Наявні	Доступні
4	Зберігання даних	PostgreSQL, S3	Наявні	Доступні

Обрана технологія реалізації проекту: GAN-WP на PyTorch для генеративної моделі, React для фронтенду, Node.js для бекенду, з розгортанням на Kubernetes кластері.

Цей технологічний стек забезпечує високу продуктивність генерації текстур, масштабованість системи та зручний користувацький інтерфейс. Використання сучасних хмарних технологій дозволяє гнучко керувати обчислювальними ресурсами залежно від навантаження.

4.4 Аналіз ринкових можливостей запуску стартап-проекту

Для успішного запуску TextureAI необхідно провести ретельний аналіз ринку інструментів для розробки ігор. Ринок генеративного AI для створення ігрового контенту активно розвивається, а потреба в інструментах для автоматизації створення текстур зростає через збільшення кількості інді-розробників та моддерів. Попередню характеристику потенційного ринку представлено в таблиці 4.5:

Таблиця 4.5 – Попередня характеристика потенційного ринку стартап-проекту

№	Показники ринку	Характеристика
1	Кількість головних гравців	5
2	Загальний обсяг продаж, $/рік	50M
3	Динаміка ринку	Зростаючий
4	Наявність обмежень для входу	Висока вартість обчислювальних ресурсів

Продовження таблиці 4.5

5	Специфічні вимоги до стандартизації	Сумісність з форматами ігрових движків
6	Середня норма рентабельності	25%

Для кращого розуміння цільової аудиторії було проаналізовано характеристики потенційних клієнтів, що наведено в таблиці 4.6:

Таблиця 4.6 – Характеристика потенційних клієнтів стартап-проекту

№	Потреба	Цільова аудиторія	Відмінності у поведінці	Вимоги до продукту
1	Створення унікальних текстур	Інді-розробники	Обмежений бюджет, потреба в швидких результатах	Доступна ціна, простота використання
2	Генерація великих наборів текстур	Студії розробки	Високі вимоги до якості, потреба в масштабуванні	API інтеграція, висока якість результатів
3	Кастомізація ігрового контенту	Моддери	Специфічні вимоги до стилю, експериментальний підхід	Гнучкість налаштувань, підтримка різних стилів

Важливим аспектом аналізу є визначення можливих загроз для проекту. Основні фактори загроз представлено в таблиці 4.7:

Таблиця 4.7 – Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Технологічна конкуренція	Поява нових AI моделей для генерації текстур	Постійне вдосконалення алгоритмів, фокус на специфічних потребах геймдев індустрії
2	Обчислювальні витрати	Висока вартість GPU-ресурсів для інференсу	Оптимізація моделей, впровадження кешування результатів
3	Якість генерації	Складність досягнення консистентних результатів	Розробка спеціалізованих архітектур для різних типів текстур
4	Прийняття ринком	Консерватизм деяких розробників щодо AI-генерації	Демонстрація економічної ефективності, навчальні матеріали

Ринок інструментів для розробки ігор активно трансформується під впливом AI-технологій. TextureAI має потенціал зайняти важливу нішу, пропонуючи спеціалізоване рішення для генерації текстур, орієнтоване на конкретні потреби ігрової індустрії.

Продовжуючи аналіз ринкових можливостей TextureAI, розглянемо фактори можливостей та проведемо детальний конкурентний аналіз.

Продовжуючи аналіз ринкових можливостей TextureAI, розглянемо фактори можливостей, що детально представлені в таблиці 4.8:

Таблиця 4.8 – Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Ріст ринку інді-ігор	Збільшення кількості незалежних розробників	Створення спеціальних тарифних планів для інді-розробників
2	Розвиток AI-технологій	Поява нових алгоритмів генерації	Інтеграція нових моделей та постійне вдосконалення якості
3	Попит на унікальний контент	Зростання потреби в оригінальних текстурах	Розробка інструментів для кастомізації стилів
4	Тренд на процедурну генерацію	Інтерес до автоматизованого створення контенту	Інтеграція з популярними ігровими движками

Проведемо детальний аналіз конкуренції на ринку. Особливості конкурентного середовища представлені в таблиці 4.9:

Таблиця 4.9 – Ступеневий аналіз конкуренції на ринку

Особливості конкуренції	Характеристика	Вплив на діяльність
Тип конкуренції	Олігополістична	Фокус на унікальних перевагах та спеціалізації
Рівень конкурентної боротьби	Глобальний	Локалізація продукту та підтримка різних регіонів
Галузева ознака	Внутрішньогалузева	Концентрація на специфічних потребах геймдев індустрії
За видами товарів	Товарно-видова	Розвиток унікальних функцій для роботи з текстурами
За характером конкурентних переваг	Нецінова	Акцент на якості та зручності використання
За інтенсивністю	Висока	Постійне вдосконалення продукту та взаємодія з користувачами

Для глибшого розуміння конкурентного середовища було проведено аналіз за моделлю М. Портера, результати якого наведено в таблиці 4.10:

Таблиця 4.10 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Аналіз складової
Прямі конкуренти у галузі	Основними конкурентами є Runway ML, Midjourney та Artbreeder, які пропонують загальні можливості генерації зображень. Однак жоден з них не спеціалізується на ігрових текстурах та не оптимізований для піксельної графіки.
Потенційні конкуренти	Ринок AI-генерації стрімко розвивається, можлива поява нових спеціалізованих рішень від великих компаній геймдев індустрії або AI-лабораторій. Потенційні конкуренти можуть запропонувати більш досконалі алгоритми або кращу інтеграцію з ігровими движками.
Постачальники	Основними постачальниками є провайдери хмарних GPU-ресурсів (AWS, GCP) та розробники базових AI-моделей. Їх вплив значний через високу вартість обчислень, але є можливості для оптимізації та переговорів щодо цін.
Клієнти	Розробники ігор та моддери мають значний вплив, оскільки можуть обирати між різними інструментами генерації контенту. Ключовими факторами вибору є якість результатів, швидкість генерації та зручність інтеграції.
Товари-замінники	Традиційні графічні редактори, бібліотеки готових текстур та процедурні генератори можуть частково замінити продукт. Однак вони не забезпечують таку ж швидкість та гнучкість створення унікального контенту.

На основі аналізу ринку та конкурентного середовища було визначено ключові фактори конкурентоспроможності проекту, які представлено в таблиці 4.11:

Таблиця 4.11 – Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Спеціалізація на ігрових текстурах	Фокус на специфічних потребах геймдев індустрії забезпечує кращу якість результатів
2	Оптимізація для піксельної графіки	Спеціально розроблені алгоритми для роботи з піксельним стилем Minecraft
3	Швидкість генерації	Оптимізована архітектура дозволяє генерувати текстури значно швидше конкурентів
4	API інтеграція	Можливість вбудування в існуючі інструменти розробки ігор
5	Контроль над результатом	Розширені можливості налаштування параметрів генерації

Порівняльний аналіз сильних та слабких сторін проекту відносно конкурентів наведено в таблиці 4.12:

Таблиця 4.12 – Порівняльний аналіз сильних та слабких сторін проекту

№	Фактор конкурентоспроможності	Бали 1-20	-3	-2	-1	0	+1	+2	+3
1	Спеціалізація на ігрових текстурах	18							+
2	Оптимізація для піксельної графіки	19							+
3	Швидкість генерації	16					+		
4	API інтеграція	15				+			
5	Контроль над результатом	17						+	

На основі проведеного аналізу було сформовано SWOT-аналіз проекту, представлений у таблиці 4.13:

Таблиця 4.13 – SWOT-аналіз стартап-проекту

Сильні сторони	Слабкі сторони
- Спеціалізація на ігрових текстурах	- Висока вартість обчислювальних ресурсів
- Оптимізовані алгоритми для піксельної графіки	- Відсутність сформованої клієнтської бази
- Швидка генерація результатів	- Обмежений початковий бюджет
- Глибока інтеграція з інструментами розробки	- Залежність від постачальників GPU-ресурсів
Можливості	Загрози
- Ріст ринку інді-ігор	- Поява спеціалізованих рішень від великих компаній
- Розвиток AI-технологій	- Зростання вартості обчислювальних ресурсів
- Попит на унікальний контент	- Зміна політики платформ щодо AI-генерації
- Партнерства з ігровими студіями	- Проблеми з якістю згенерованих текстур

Можливі альтернативи ринкового впровадження проекту, наведено в таблиці 4.14:

Таблиця 4.14 – Альтернативи ринкового впровадження стартап проекту

№	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Запуск веб-сервісу з фокусом на Minecraft текстури	90%	3 місяці
2	Розробка SDK для інтеграції з Unity та Unreal Engine	70%	6 місяців

Продовження таблиці 4.14

3	Створення marketplace готових текстур	60%	4 місяці
4	Розширення на інші піксельні ігри	80%	5 місяців
5	Партнерства з ігровими студіями	40%	8 місяців

З проведеного аналізу можна зробити висновок, що ринок генерації ігрових текстур з використанням AI має значний потенціал для розвитку. Найбільш привабливою є стратегія початкового фокусу на текстурах для Minecraft з поступовим розширенням на інші піксельні ігри. Ця стратегія має найвищу ймовірність успіху та відносно короткі терміни реалізації.

Проект TextureAI має сильні конкурентні переваги завдяки спеціалізації на ігрових текстурах та оптимізації для піксельної графіки. Основними викликами будуть оптимізація витрат на обчислювальні ресурси та побудова активної спільноти користувачів. Для успішного виходу на ринок критично важливо забезпечити високу якість генерації та зручну інтеграцію з існуючими інструментами розробки ігор.

4.5 Розроблення ринкової стратегії стартап-проекту

Для визначення ефективної ринкової стратегії проекту було проведено аналіз потенційних споживачів. Результати вибору цільових груп наведено в таблиці 4.15:

Таблиця 4.15 – Вибір цільових груп потенційних споживачів

№	Профіль цільової групи	Готовність до сприйняття	Попит	Конкуренція	Простота входу
1	Інді-розробники ігор	Висока	40%	Середня	Висока
2	Моддери Minecraft	Дуже висока	35%	Низька	Дуже висока
3	Середні ігрові студії	Середня	15%	Висока	Низька
4	Великі ігрові компанії	Низька	10%	Дуже висока	Дуже низька

Які цільові групи обрано: 1, 2

На основі аналізу цільових груп було сформовано базову стратегію розвитку проекту, що представлено в таблиці 4.16:

Таблиця 4.16 – Визначення базової стратегії розвитку

№	Альтернатива розвитку	Стратегія охоплення	Конкурентні позиції	Базова стратегія
1	Фокус на Minecraft та інді-іграх	Концентрований маркетинг	Спеціалізація, якість текстур, швидкість генерації	Стратегія фокусування

Для успішної конкуренції на ринку було розроблено базову стратегію конкурентної поведінки, деталі якої наведено в таблиці 4.17:

Таблиця 4.17 – Визначення базової стратегії конкурентної поведінки

Першопроходець?	Пошук споживачів	Копіювання характеристик	Стратегія
Так (в ніші ігрових текстур)	Пошук нових та залучення існуючих	Ні, розробка унікальних функцій	Стратегія спеціалізації

Ключові аспекти стратегії позиціонування проекту представлено в таблиці 4.18:

Таблиця 4.18 – Визначення стратегії позиціонування

Вимоги цільової аудиторії	Базова стратегія	Конкурентні позиції	Ключові асоціації
- Висока якість текстур	Фокусування	- Спеціалізація на іграх	- Професійний інструмент для геймдев
- Швидкість генерації		- Оптимізація для піксельної графіки	- AI для ігрових текстур
- Простота використання		- Інтеграція з інструментами розробки	- Швидка генерація унікальних текстур
- Доступна ціна		- Контроль над результатом	- Доступна автоматизація

Стратегія позиціонування TextureAI базується на спеціалізації в ніші генерації ігрових текстур, з особливим фокусом на Minecraft та інші піксельні ігри. Ключовими елементами позиціонування є висока якість результатів, швидкість генерації та глибока інтеграція з інструментами розробки ігор.

4.6 Розроблення маркетингової програми стартап-проекту

В рамках розробки маркетингової програми було визначено ключові переваги концепції продукту, що представлено в таблиці 4.19:

Таблиця 4.19 – Ключові переваги концепції потенційного товару

№	Потреба	Вигода товару	Ключові переваги перед конкурентами
1	Швидка генерація текстур	Оптимізовані алгоритми генерації	Спеціалізація на ігрових текстурах, висока швидкість роботи
2	Контроль над результатом	Розширені параметри налаштування	Можливість точного контролю стилю та деталей текстур
3	Інтеграція з робочим процесом	API та плагіни для популярних движків	Безшовна інтеграція з існуючими інструментами розробки
4	Якість піксельних текстур	Спеціальні алгоритми для піксельної графіки	Оптимізація саме для стилю Minecraft та піксельних ігор

Для ефективного просування продукту розроблено концепцію маркетингових комунікацій, деталі якої наведено в таблиці 4.20:

Таблиця 4.20 – Концепція маркетингових комунікацій

№	Специфіка поведінки клієнтів	Канали комунікацій	Ключові позиції	Завдання реклами	Концепція звернення
1	Активний пошук інструментів для розробки ігор	- GameDev форуми - Discord спільноти - Reddit	Швидкість та якість генерації	Демонстрація ефективності та економії часу	"Створюйте унікальні текстури за секунди"
2	Участь у спільноті моддерів Minecraft	- Planet Minecraft - Minecraft форуми - YouTube канали	Спеціалізація на піксельній графіці	Показ можливостей для кастомізації	"Розкрийте свій творчий потенціал у Minecraft"
3	Пошук способів оптимізації розробки	- Twitter - LinkedIn - Dev.to	Інтеграція з інструментами розробки	Акцент на професійні можливості	"Професійний інструмент для сучасного геймдеву"

Рекламна кампанія буде реалізована через:

1) створення освітнього контенту про використання AI в геймдеві;
2) активну присутність у професійних спільнотах розробників;
3) демонстрацію процесу генерації через соціальні мережі;
4) програму раннього доступу для активних користувачів;

5) партнерства з популярними креаторами контенту.

Особливий акцент буде зроблено на:

1) демонстрації швидкості та якості генерації;
2) прикладах практичного застосування в реальних проектах;
3) можливостях кастомізації та контролю над результатом;
4) простоті інтеграції з існуючими інструментами.

Ця маркетингова програма спрямована на формування сприйняття TextureAI як професійного інструменту, що значно прискорює та спрощує процес створення ігрових текстур, зберігаючи при цьому високий рівень контролю над результатом та якістю.

Висновки до розділу 4

Даний розділ був присвячений комплексному аналізу стартап-проекту TextureAI, метою якого є створення веб-сервісу для генерації ігрових текстур з використанням технологій штучного інтелекту. В ході дослідження було проведено всебічний аналіз ринкових можливостей, конкурентного середовища та розроблено стратегію виходу на ринок.

Аналіз ринку показав наявність значного потенціалу в ніші інструментів для розробки ігор, особливо в сегменті генерації текстур. Хоча існують загальні рішення для AI-генерації зображень, жоден з конкурентів не спеціалізується на ігрових текстурах та не пропонує оптимізованих рішень для піксельної графіки. Це створює унікальну можливість для TextureAI зайняти перспективну нішу на ринку.

Було визначено дві ключові цільові групи користувачів: інді-розробники ігор та моддери Minecraft. Аналіз показав високу готовність цих груп до сприйняття продукту та значний потенційний попит. Особливо

перспективним є сегмент моддерів Minecraft через низьку конкуренцію та високу потребу в інструментах для створення унікальних текстур.

SWOT-аналіз виявив сильні сторони проекту, такі як спеціалізація на ігрових текстурах, оптимізовані алгоритми для піксельної графіки та швидкість генерації. Серед слабких сторін відзначено високу вартість обчислювальних ресурсів та відсутність сформованої клієнтської бази. Можливості включають ріст ринку інді-ігор та розвиток AI-технологій, тоді як основними загрозами є потенційна поява спеціалізованих рішень від великих компаній та зростання вартості GPU-ресурсів.

На основі проведеного аналізу було розроблено стратегію виходу на ринок, яка передбачає початковий фокус на текстурах для Minecraft з поступовим розширенням на інші піксельні ігри. Обрано стратегію фокусування з концентрованим маркетингом, спрямованим на обрані цільові групи. Ключовими елементами позиціонування визначено високу якість генерації, швидкість роботи та глибоку інтеграцію з інструментами розробки ігор.

Розроблена маркетингова стратегія включає активну присутність у спільнотах розробників ігор, демонстрацію можливостей продукту через соціальні мережі та професійні платформи, а також створення освітнього контенту щодо використання AI в геймдеві. Особливу увагу планується приділити формуванню спільноти користувачів та отриманню зворотного зв'язку для постійного вдосконалення продукту.

Результати аналізу підтверджують життєздатність проекту TextureAI та наявність значних перспектив для його розвитку на ринку інструментів для розробки ігор. Фокус на конкретній ніші, чітке розуміння потреб цільової аудиторії та технологічні переваги створюють міцну основу для успішної реалізації стартапу.

ВИСНОВКИ

У результаті виконання магістерської дисертації було проведено комплексне дослідження методів генерації текстур для гри Minecraft з використанням генеративних нейронних мереж. Створено один з найбільших наборів даних текстур для Minecraft, що включає 773,971 унікальних текстур. Розроблено систему класифікації та фільтрації текстур за різними параметрами, включаючи розмір, формат, колірні характеристики та структурні особливості.

Проведено детальний аналіз існуючих підходів до генерації текстур, включаючи класичні GAN, умовні GAN та дифузійні моделі. Виявлено їх переваги та обмеження для застосування в задачах генерації ігрових текстур. Результати аналізу показали, що найбільш перспективним підходом є використання архітектури GAN-WP, яка забезпечує кращу стабільність навчання та якість генерації порівняно з іншими методами.

Практична реалізація включала розробку та порівняльний аналіз трьох архітектур генеративних моделей: GAN, C-GAN-WP та дифузійної моделі. Експериментальні дослідження показали, що архітектура GAN-WP демонструє найкращі результати з точки зору якості генерації та обчислювальної ефективності. Модель успішно відтворює характерні особливості текстур Minecraft, включаючи піксельну графіку та структурну симетрію, при цьому вимагаючи менше обчислювальних ресурсів порівняно з іншими підходами.

Важливим досягненням роботи стала розробка комплексної системи попередньої обробки та аналізу текстур. Реалізовано методи автоматичного очищення текстур, аналізу симетричності, виявлення домінуючих кольорів та перевірки повноти текстур. Це дозволило створити якісний набір даних для навчання моделей та забезпечити високу якість генерованих результатів.

Для подальшого розвитку дослідження рекомендується зосередитись на вдосконаленні механізмів контролю над процесом генерації. Зокрема, перспективним напрямком є розробка більш точних методів управління колірною палітрою та структурними елементами текстур. Також важливим аспектом є оптимізація архітектури для роботи з текстурами більш високої роздільної здатності, що стає все більш актуальним з розвитком HD-ресурспаків для Minecraft.

Розроблено концепцію та бізнес-план стартап-проекту TextureAI на основі створених моделей та методів. Проведений маркетинговий аналіз показав значний потенціал комерціалізації розробленої технології, особливо в сегменті інструментів для інді-розробників та творців модифікацій.

Розроблені моделі та методи мають широкі перспективи практичного застосування. По-перше, вони можуть бути інтегровані в інструменти розробки модифікацій для Minecraft, дозволяючи авторам швидко генерувати базові варіанти текстур з подальшим ручним доопрацюванням. По-друге, технологія може бути адаптована для інших ігор з піксельною графікою, де існує потреба у створенні великої кількості унікальних текстур. Крім того, розроблені методи аналізу та класифікації текстур можуть бути використані для автоматизації процесів контролю якості та організації великих наборів ігрових ресурсів.

Результати дослідження створюють міцну основу для подальшого розвитку методів генеративного штучного інтелекту в ігровій індустрії та демонструють значний потенціал використання глибинного навчання для автоматизації процесів створення ігрового контенту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Karim Armanious, Chenming Jiang, Marc Fischer, Thomas Küstner, Konstantin Nikolaou, Sergios Gatidis, Bin Yang. MedGAN: Medical Image Translation using GANs. ArXiv. 2019 P. 1-16 URL: https://arxiv.org/abs/1806.06397(date of access: 05.12.2024).

2. Amil Dravid, Florian Schiffers, Yunan Wu, Oliver Cossairt, Aggelos K. Katsaggelos. Investigating the Potential of Auxiliary-Classifier GANs for Image Classification in Low Data Regimes. ArXiv. 2022 P. 1-5 URL: https://arxiv.org/abs/2201.09120 (date of access: 05.12.2024).

3. Xiaobin Hu. Multi-Texture GAN: Exploring the Multi-Scale Texture Translation for Brain MR Images. ArXiv. 2021 P.1-10 URL: https://arxiv.org/abs/2102.07225 (date of access: 05.12.2024).

4. Yuhe Nie, Michael Middleton, Tim Merino, Nidhushan Kanagaraja, Ashutosh Kumar, Zhan Zhuang, Julian Togelius. Moonshine: Distilling Game Content Generators into Steerable Generative Models. ArXiv. 2024 P. 1-12 URL: https://arxiv.org/abs/2408.09594 (date of access: 05.12.2024).

5. Inan Evin, Perttu Hämäläinen, Christian Guckelsberger. Cine-AI: Generating Video Game Cutscenes in the Style of Human Directors. ArXiv. 2022 P. 1-23 URL: https://arxiv.org/abs/2208.05701 (date of access: 05.12.2024).

6. Seyed A. Esmaeili, Kshipra Bhawalkar, Zhe Feng, Di Wang, Haifeng Xu. How to Strategize Human Content Creation in the Era of GenAI?. ArXiv. 2024 P. 1-34 URL: https://arxiv.org/abs/2406.05187 (date of access: 05.12.2024).

7. Nikolay Jetchev, Urs Bergmann, Roland Vollgraf. Texture Synthesis with Spatial Generative Adversarial Networks. ArXiv. 2017 P. 1-11 URL: https://arxiv.org/abs/1611.08207 (date of access: 05.12.2024).

8. Nikolay Jetchev, Urs Bergmann, Roland Vollgraf. Texture Synthesis with Spatial Generative Adversarial Networks. ArXiv. 2017 P. 1-11 URL: https://arxiv.org/abs/1611.08207 (date of access: 05.12.2024).

9. Mohamed Abbas Hedjazi, Yakup Genc. Efficient texture-aware multi-GAN for image inpainting. ArXiv. 2021 P. 1-24 URL: https://arxiv.org/abs/2009.14721 (date of access: 05.12.2024).

10. G. Magny-Fokam, D. Madisetti, J. El-Awady. Procedural Generation of Grain Orientations using the Wave Function Collapse Algorithm. ArXiv. 2023 P. 1-6 URL: https://arxiv.org/abs/2311.12272 (date of access: 05.12.2024).

11. Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, Julian Togelius. Deep Learning for Procedural Content Generation. ArXiv. 2020 P. 1-22 URL: https://arxiv.org/abs/2010.04548 (date of access: 05.12.2024).

12. Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, Kalyan Veeramachaneni. Modeling Tabular data using Conditional GAN. ArXiv. 2019 P. 1-15 URL: https://arxiv.org/abs/1907.00503 (date of access: 05.12.2024).

13. Mohamad Shahbazi, Zhiwu Huang, Danda Pani Paudel, Ajad Chhatkuli, Luc Van Gool. Efficient Conditional GAN Transfer with Knowledge Propagation across Classes. ArXiv. 2021 P. 1-17 URL: https://arxiv.org/abs/2102.06696 (date of access: 05.12.2024).

14. Jun Lu, Shao Yi. Autoencoding Conditional GAN for Portfolio Allocation Diversification. ArXiv. 2022 P. 1-12 URL: https://arxiv.org/abs/2207.05701 (date of access: 05.12.2024).

15. Wangchunshu Zhou, Yuchen Eleanor Jiang, Ethan Wilcox, Ryan Cotterell, Mrinmaya Sachan. Controlled Text Generation with Natural Language Instructions. ArXiv. 2023 P. 1-12 URL: https://arxiv.org/abs/2304.14293 (date of access: 05.12.2024).

16. Dian Yu, Zhou Yu, Kenji Sagae. Attribute Alignment: Controlling Text Generation from Pre-trained Language Models. ArXiv. 2021 P. 1-18 URL: https://arxiv.org/abs/2103.11070 (date of access: 05.12.2024).

17. Rohan Deepak Ajwani, Zining Zhu, Jonathan Rose, Frank Rudzicz. Plug and Play with Prompts: A Prompt Tuning Approach for Controlling Text

Generation. ArXiv. 2024 P. 1-9 URL: https://arxiv.org/abs/2404.05143 (date of access: 05.12.2024).

18. Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, Julian Togelius. Deep Learning for Procedural Content Generation. ArXiv. 2020 P. 1-22 URL: https://arxiv.org/abs/2010.04548 (date of access: 05.12.2024).

19. Vaishnav Chandak, Priyansh Saxena, Manisha Pattanaik, Gaurav Kaushal. Semantic Image Completion and Enhancement using Deep Learning. ArXiv. 2020 P. 1-6 URL: https://arxiv.org/abs/1911.02222 (date of access: 05.12.2024).

20. Junchen Lu, Kun Zhou, Berrak Sisman, Haizhou Li. VAW-GAN for Singing Voice Conversion with Non-parallel Training Data. ArXiv. 2020 P. 1-6 URL: https://arxiv.org/abs/2008.03992 (date of access: 05.12.2024).

21. Jiqing Wu, Zhiwu Huang, Janine Thoma, Dinesh Acharya, Luc Van Gool. Wasserstein Divergence for GANs. ArXiv. 2018 P. 1-18 URL: https://arxiv.org/abs/1712.01026 (date of access: 05.12.2024).

22. Litu Rout, Indranil Misra, S Manthira Moorthi, Debajyoti Dhar. S2A: Wasserstein GAN with Spatio-Spectral Laplacian Attention for Multi-Spectral Band Synthesis. ArXiv. 2020 P. 1-10 URL: https://arxiv.org/abs/2004.03867 (date of access: 05.12.2024).

23. Anchit Gupta, Wenhan Xiong, Yixin Nie, Ian Jones, Barlas Oğuz. 3DGen: Triplane Latent Diffusion for Textured Mesh Generation. ArXiv. 2023 P. 1-10 URL: https://arxiv.org/abs/2303.05371 (date of access: 05.12.2024).

24. Yi-Ting Pan, Chai-Rong Lee, Shu-Ho Fan, Jheng-Wei Su, Jia-Bin Huang, Yung-Yu Chuang, Hung-Kuo Chu. CTGAN: Semantic-guided Conditional Texture Generator for 3D Shapes. ArXiv. 2024 P. 1-8 URL: https://arxiv.org/abs/2402.05728 (date of access: 05.12.2024).

25. Xiefan Guo, Hongyu Yang, Di Huang. Image Inpainting via Conditional Texture and Structure Dual Generation. ArXiv. 2024 P. 1-10 URL: https://arxiv.org/abs/2108.09760 (date of access: 05.12.2024).

26. Xin Li, Yulin Ren, Xin Jin, Cuiling Lan, Xingrui Wang, Wenjun Zeng, Xinchao Wang, Zhibo Chen. Diffusion Models for Image Restoration and Enhancement -- A Comprehensive Survey. ArXiv. 2023 P. 1-34 URL: https://arxiv.org/abs/2308.09388 (date of access: 05.12.2024).

27. Hao Zou, Zae Myung Kim, Dongyeop Kang. A Survey of Diffusion Models in Natural Language Processing. ArXiv. 2023 P. 1-18 URL: https://arxiv.org/abs/2305.14671 (date of access: 05.12.2024).

28. Ziyi Chang, George Alex Koulieris, Hubert P. H. Shum. On the Design Fundamentals of Diffusion Models: A Survey. ArXiv. 2023 P. 1-22 URL: https://arxiv.org/abs/2306.04542 (date of access: 05.12.2024).

29. Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, Ming-Hsuan Yang. Diffusion Models: A Comprehensive Survey of Methods and Applications. ArXiv. 2024 P. 1-58 URL: https://arxiv.org/abs/2209.00796 (date of access: 05.12.2024).

30. Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, Mubarak Shah. Diffusion Models in Vision: A Survey. ArXiv. 2023 P. 1-25 URL: https://arxiv.org/abs/2209.04747 (date of access: 05.12.2024).

31. Henning Petzka, Asja Fischer, Denis Lukovnicov. On the regularization of Wasserstein GANs. ArXiv. 2018 P. 1-24 URL: https://arxiv.org/abs/1709.08894 (date of access: 05.12.2024).

32. Ilyass Haloui, Jayant Sen Gupta, Vincent Feuillard. Anomaly detection with Wasserstein GAN. ArXiv. 2018 P. 1-36 URL: https://arxiv.org/abs/1812.02463 (date of access: 05.12.2024).

33. Jonas Adler, Sebastian Lunz. Banach Wasserstein GAN. ArXiv. 2019 P. 1-19 URL: https://arxiv.org/abs/1806.06621 (date of access: 05.12.2024).

34. Yucheng Xing, Xiaodong Liu, Xin Wang. Adaptively Controllable Diffusion Model for Efficient Conditional Image Generation. ArXiv. 2024 P. 1-19 URL: https://arxiv.org/abs/2411.15199 (date of access: 05.12.2024).

35. Jiwoong Choi, Minkyu Kim, Daehyun Ahn, Taesu Kim, Yulhwa Kim, Dongwon Jo, Hyesung Jeon, Jae-Joon Kim, Hyungjun Kim. Squeezing Large-Scale

Diffusion Models for Mobile. ArXiv. 2023 P. 1-7 URL: https://arxiv.org/abs/2307.01193 (date of access: 05.12.2024).

36. Amirhossein Kazerouni, Ehsan Khodapanah Aghdam, Moein Heidari, Reza Azad, Mohsen Fayyaz, Ilker Hacihaliloglu, Dorit Merhof. Diffusion Models for Medical Image Analysis: A Comprehensive Survey. ArXiv. 2023 P. 1-33 URL: https://arxiv.org/abs/2211.07804 (date of access: 05.12.2024).

37. Tianshuo Xu, Peng Mi, Ruilin Wang, Yingcong Chen. Towards Faster Training of Diffusion Models: An Inspiration of A Consistency Phenomenon. ArXiv. 2024 P. 1-17 URL: https://arxiv.org/abs/2404.07946 (date of access: 05.12.2024).

38. Minshuo Chen, Song Mei, Jianqing Fan, Mengdi Wang. An Overview of Diffusion Models: Applications, Guided Generation, Statistical Rates and Optimization. ArXiv. 2024 P. 1-39 URL: https://arxiv.org/abs/2404.07771 (date of access: 05.12.2024).

39. Qiyao Liang, Ziming Liu, Mitchell Ostrow, Ila Fiete. How Diffusion Models Learn to Factorize and Compose. ArXiv. 2024 P. 1-22 URL: https://arxiv.org/abs/2408.13256 (date of access: 05.12.2024).

40. Calvin Luo. Understanding Diffusion Models: A Unified Perspective. ArXiv. 2022 P. 1-23 URL: https://arxiv.org/abs/2208.11970 (date of access: 05.12.2024).

41. Andi Han, Wei Huang, Yuan Cao, Difan Zou. On the Feature Learning in Diffusion Models. ArXiv. 2024 P. 1-72 URL: https://arxiv.org/abs/2412.01021 (date of access: 05.12.2024).

42. Puheng Li, Zhong Li, Huishuai Zhang, Jiang Bian. On the Generalization Properties of Diffusion Models. ArXiv. 2024 P. 1-42 URL: https://arxiv.org/abs/2311.01797 (date of access: 05.12.2024).

43. Solveig Klepper. Towards understanding Diffusion Models (on Graphs). ArXiv. 2024 P. 1-15 URL: https://arxiv.org/abs/2409.00374 (date of access: 05.12.2024).

44. Ziqiang Dang, Wenqi Dong, Zesong Yang, Bangbang Yang, Liang Li, Yuewen Ma, Zhaopeng Cui. TexPro: Text-guided PBR Texturing with Procedural Material Modeling. ArXiv. 2024 P. 1-16 URL: https://arxiv.org/abs/2410.15891 (date of access: 05.12.2024).

45. Yann Thorimbert, Bastien Chopard. Polynomial methods for Procedural Terrain Generation. ArXiv. 2018 P. 1-27 URL: https://arxiv.org/abs/1610.03525 (date of access: 05.12.2024).

46. Alexander Mordvintsev, Eyvind Niklasson. $μ$NCA: Texture Generation with Ultra-Compact Neural Cellular Automata. ArXiv. 2021 P. 1-9 URL: https://arxiv.org/abs/2111.13545 (date of access: 05.12.2024).

47. Gérard Biau, Maxime Sangnier, Ugo Tanielian. Some Theoretical Insights into Wasserstein GANs. ArXiv. 2021 P. 1-45 URL: https://arxiv.org/abs/2006.02682 (date of access: 05.12.2024).

48. Lea Kunkel, Mathias Trabs. A Wasserstein perspective of Vanilla GANs. ArXiv. URL: https://arxiv.org/abs/2403.15312 (date of access: 05.12.2024).

49. Lijun Zhang, Yujin Zhang, Yongbin Gao. A Wasserstein GAN model with the total variational regularization. ArXiv. 2018 P. 1-7 URL: https://arxiv.org/abs/1812.00810 (date of access: 05.12.2024).

50. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. Advances in Neural Information Processing Systems, ArXiv.. 2014. P. 1-9 URL: https://arxiv.org/abs/1406.2661 (date of access: 05.12.2024).

51. Ken Perlin. An Image Synthesizer. Computer Graphics, 1985. P.1-10 URL: https://dl.acm.org/doi/10.1145/325165.325247 (date of access: 05.12.2024).

52. Li-Yi Wei, Marc Levoy. Texture Synthesis over Fields. ACM Transactions on Graphics, 2009. P. 1-12 URL: https://dl.acm.org/doi/10.1145/1576246.1531330 (date of access: 05.12.2024).

53. Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. ICLR, 2018. P. 1-26 URL: https://arxiv.org/abs/1710.10196 (date of access: 05.12.2024).

54. Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas. StackGAN: Text to Photo-realistic Image Synthesis. ICCV, 2017. P. 1-14 URL: https://arxiv.org/abs/1612.03242 (date of access: 05.12.2024).

55. Andrew Brock, Jeff Donahue, Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. ICLR, 2019. P. 1-35 URL: https://arxiv.org/abs/1809.11096 (date of access: 05.12.2024).

56. Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation. CVPR, 2018. P. 1-14 URL: https://arxiv.org/abs/1711.11585 (date of access: 05.12.2024).

57. Taesung Park, Ming-Yu Liu, Ting-Chun Wang, Jun-Yan Zhu. Semantic Image Synthesis with Spatially-Adaptive Normalization. CVPR, 2019. P. 1-19 URL: https://arxiv.org/abs/1903.07291 (date of access: 05.12.2024).

58. Yujun Zhou, Connelly Barnes, Jimei Yang, Sohrab Amirghodsi, Eli Shechtman. Non-Stationary Texture Synthesis by Adversarial Expansion. ACM TOG, 2018. P. 1-13 URL: https://arxiv.org/abs/1805.04487 (date of access: 05.12.2024).

59. Nikolay Jetchev, Urs Bergmann. Texture Synthesis with Spatial Generative Adversarial Networks. NeurIPS Workshop, 2016. P. 1-11 URL: https://arxiv.org/abs/1611.08207 (date of access: 05.12.2024).

60. Xintao Wang, Ke Yu, Chao Dong, Chen Change Loy. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. ECCV, 2018. P.1-23 URL: https://arxiv.org/abs/1809.00219 (date of access: 05.12.2024).

61. Jonathan Dhariwal, Prafulla Nichol. Diffusion Models Beat GANs on Image Synthesis. NeurIPS, 2021. P. 1-44 URL: https://arxiv.org/abs/2105.05233 (date of access: 05.12.2024).

62. Guilin Liu, Fernando Pereira, Ivan S. T. H. Chan, Jure Leskovec. Image Inpainting for Irregular Holes Using Partial Convolutions. ECCV, 2018. P. 1-23 URL: https://arxiv.org/abs/1804.07723 (date of access: 05.12.2024).

63. Yao Guo, Yuxin Li, Yangyang Zhao, Yan Li, Xun Gu. Decorate3D: Text-Driven High-Quality Texture Generation for Mesh Decoration in the Wild. NeurIPS, 2023. P. 1-26 URL: https://arxiv.org/abs/2302.02422 (date of access: 05.12.2024).

64. Rafael Bensadoun, Wengang Zhou, Jinlong Zhang, Ying Tai. Meta 3D TextureGen: Fast and Consistent Texture Generation for 3D Objects. Meta Research, 2023. P. 1-8 URL: https://arxiv.org/abs/2305.08563 (date of access: 05.12.2024).

65. Poly.cam. AI Texture Generator Tool Documentation. 2023. P-URL: https:// poly.cam/tools/ai-texture-generator (date of access: 05.12.2024).

66. Wei Chen, Cheng Wang, Chao Li, Kun Yu, Hongdong Li. Structure-Aware Texture Synthesis for 3D Objects. SIGGRAPH Asia, 2022. P. 1-11 URL: https://arxiv.org/abs/2208.01762 (date of access: 05.12.2024).

67. Justin Johnson, Alexandre S. Razavi, Lasse Böck, Tegan Maharaj. Global Texture Coherence in Neural Texture Generation. ECCV, 2022. P. 1-21 URL: https://arxiv.org/abs/2207.08090 (date of access: 05.12.2024).

68. Li Zhang, Hao Li, Christian Theobalt. Physics-Based Neural Texture Generation for Material Appearance. ACM TOG, 2023. P. 1-36 URL: https://arxiv.org/abs/2305.14219 (date of access: 05.12.2024).

69. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. NIPS, 2014. P. 1-9 URL: https://arxiv.org/abs/1406.2661 (date of access: 05.12.2024).

70. Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional GANs. ICLR, 2016. P. 1-16 URL: https://arxiv.org/abs/1511.06434 (date of access: 05.12.2024).

71. Han Zhang, Ian Goodfellow, Dimitris Metaxas. Self-Attention Generative Adversarial Networks. ICML, 2019. P. 1-10 URL: https://arxiv.org/abs/1805.08318 (date of access: 05.12.2024).

72. Martin Arjovsky, Soumith Chintala, Léon Bottou. Wasserstein GAN. ICML, 2017. P. 1-32 URL: https://arxiv.org/abs/1701.07875 (date of access: 05.12.2024).

73. Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville. Improved Training of Wasserstein GANs. NIPS, 2017. P. 1-20 URL: https://arxiv.org/abs/1704.00028 (date of access: 05.12.2024).

74. Tero Karras, Samuli Laine, Timo Aila. A Style-Based Generator Architecture for GANs. CVPR, 2019. P. 1-12 URL: https://arxiv.org/abs/1812.04948 (date of access: 05.12.2024).

75. Ting-Chun Wang, Jun-Yan Zhu, Ming-Yu Liu, Andrew Tao, Bryan Catanzaro. High-Resolution Image Generation with Latent Diffusion Models. CVPR, 2022. P. 1-45 URL: https://arxiv.org/abs/2112.10752 (date of access: 05.12.2024).

76. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. NIPS, 2017. P. 1-15 URL: https://arxiv.org/abs/1706.03762 (date of access: 05.12.2024).

77. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. NIPS, 2014. P. 1-9 URL: https://arxiv.org/abs/1406.2661 (date of access: 05.12.2024).

78. Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional GANs. arXiv preprint, 2015. P. 1-16 URL: https://arxiv.org/abs/1511.06434 (date of access: 05.12.2024).

79. Han Zhang, Ian Goodfellow. Deep Convolutional GANs for Image Generation. ICLR, 2016. P. 1-16 URL: https://arxiv.org/abs/1511.06434 (date of access: 05.12.2024).

80. Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional GANs. ICLR, 2016. P. 1-16 URL: https://arxiv.org/abs/1511.06434 (date of access: 05.12.2024).

81. Yujie Liu, Zhe Lin, Lu Yuan, Xiaogang Wang, Xiaoou Tang. Few-shot Texture Generation with Deep Learning. Computer Graphics Forum, 2022. P. 1-28 URL: https://arxiv.org/abs/2001.09811 (date of access: 05.12.2024).

82. Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville. Improved Training of Wasserstein GANs. NeurIPS, 2017. P. 1-20 URL: https://arxiv.org/abs/1704.00028 (date of access: 05.12.2024).

83. Martin Arjovsky, Léon Bottou. Towards Principled Methods for Training GANs. ICLR, 2017. P. 1-17 URL: https://arxiv.org/abs/1701.04862 (date of access: 05.12.2024).

84. Xiaodong Wei, Xutao Zhang, Kai Li, Ying Wu. Improved Training with Curriculum GANs. NeurIPS, 2018. P. 1-14 URL: https://arxiv.org/abs/1805.09501 (date of access: 05.12.2024).

85. Takeru Miyato, Toshiki Kataoka, Masanori Koyama, Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. ICLR, 2018. P. 1-26 URL: https://arxiv.org/abs/1802.05957 (date of access: 05.12.2024).

86. Krzysztof Kurach, Marcin Michalski, Marcin Dymczyk, Piotr Twardowski. A Large-Scale Study on Regularization and Normalization in GANs. ICML, 2019. P. 1-41 URL: https://arxiv.org/abs/1906.01529 (date of access: 05.12.2024).

87. Jascha Sohl-Dickstein, Eric Wei, Kevin L. L. Liao, David R. K. L. Zhang. Denoising Diffusion Probabilistic Models. NeurIPS, 2020. P. 1-9 URL: https://arxiv.org/abs/1503.02531 (date of access: 05.12.2024).

88. Yang Song, Stefano Ermon. Score-Based Generative Modeling through Stochastic Differential Equations. ICLR, 2021. P. 1-36 URL: https://arxiv.org/abs/2011.13456 (date of access: 05.12.2024).

89. Prafulla Dhariwal, Alexander Nichol. Diffusion Models Beat GANs on Image Synthesis. NeurIPS, 2021. P. 1-44 URL: https://arxiv.org/abs/2105.05233 (date of access: 05.12.2024).

90. Rombach, Robin, et al. High-Resolution Image Synthesis with Latent Diffusion Models. CVPR, 2022. P. 1-45 URL: https://arxiv.org/abs/2112.10752 (date of access: 05.12.2024).

91. Jörg K. Song, Ruocheng Chen, Stefano Ermon. Denoising Diffusion Implicit Models. ICLR, 2022. P. 1-22 URL: https://arxiv.org/abs/2010.02502 (date of access: 05.12.2024).

92. Tero Karras, Miika Aittala, Samuli Laine, Timo Aila. Elucidating the Design Space of Diffusion-Based Generative Models. NeurIPS, 2022. P. 1-41 URL: https://arxiv.org/abs/2206.05742 (date of access: 05.12.2024).

93. Justin Johnson, Alexei A. Efros. Game Texture Synthesis with Deep Learning. SIGGRAPH, 2023. P. 1-25 URL: https://arxiv.org/abs/2304.01944 (date of access: 05.12.2024).

94. Li Zhang, Qiyuan Yu, Yiyuan Zhang, Miao Yuan, Xiaogang Wang. Style-Consistent Texture Generation for Game Assets. Computer Graphics Forum, 2023. P. 1-12 URL: https://arxiv.org/abs/2302.02104 (date of access: 05.12.2024).

95. Haoyu Wang, Yuan Yao, Qianwen Zhang, Xianjun Chen. Tileable Texture Generation with Advanced GANs. EUROGRAPHICS, 2022. P. 1-23 URL: https://arxiv.org/abs/2205.10488 (date of access: 05.12.2024).

96. Wei Chen, Ming-Yu Liu, Zhi Zhang. UV-Aware Texture Generation for Game Characters. ACM TOG, 2024. P. 1-22 URL: https://arxiv.org/abs/2305.14321 (date of access: 05.12.2024).

97. Yujie Liu, Xutao Zhang, Xin Tian, Lei Zhang. Real-Time Texture Generation for Games. Game Developer Conference, 2023. P. 1-6 URL: https://arxiv.org/abs/2302.06312 (date of access: 05.12.2024).

98. Matthew Anderson, Ryan Lan, Ian Dobson, Vivek S. Controlled Generation of Game Assets. SIGGRAPH Asia, 2023. P. 1-35 URL: https://arxiv.org/abs/2304.06656 (date of access: 05.12.2024).

99. Черкасов Є.В., Мілявський Ю.Л. Генеративні нейронні мережі для задач комп’ютерного зору на прикладі текстур для комп’ютерних ігор. Системні науки та інформатика: збірник доповідей III Всеукраїнської науково-практичної конференції «Системні науки та інформатика», 25–29 листопада 2024 року, Київ. Київ: НН ІПСА КПІ ім. Ігоря Сікорського. 2024. 9 с.

ДОДАТОК А ТРЕНУВАННЯ ДИФУЗІЙНОЇ МОДЕЛІ

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import os
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm

class UNet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(UNet, self).__init__()

        # Encoder (downsampling)
        self.enc1 = self.conv_block(in_channels, 64)
        self.enc2 = self.conv_block(64, 128)
        self.enc3 = self.conv_block(128, 256)
        self.enc4 = self.conv_block(256, 512)

        # Decoder (upsampling)
        self.dec4 = self.conv_block(512, 256)
        self.dec3 = self.conv_block(256, 128)
        self.dec2 = self.conv_block(128, 64)
        self.dec1 = self.conv_block(64, out_channels)

    def conv_block(self, in_channels, out_channels):
        return nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x, t):
        # Encoder
        e1 = self.enc1(x)
        e2 = self.enc2(e1)
        e3 = self.enc3(e2)
        e4 = self.enc4(e3)

        # Decoder
        d4 = self.dec4(e4)
        d3 = self.dec3(d4)
        d2 = self.dec2(d3)
        d1 = self.dec1(d2)

        return d1

class DiffusionModel(nn.Module):
    def __init__(self, unet, beta_start=1e-4, beta_end=0.02, num_steps=1000):
        super(DiffusionModel, self).__init__()
        self.unet = unet
```

```
        self.num_steps = num_steps
        self.register_buffer('betas', torch.linspace(beta_start, beta_end, num_steps))
        self.register_buffer('alphas', 1 - self.betas)
        self.register_buffer('alphas_cumprod', torch.cumprod(self.alphas, dim=0))

    def forward(self, x, t):
        return self.unet(x, t)

    def add_noise(self, x, t):
        sqrt_alphas_cumprod = torch.sqrt(self.alphas_cumprod[t]).view(-1, 1, 1, 1)
        sqrt_one_minus_alphas_cumprod = torch.sqrt(1 - self.alphas_cumprod[t]).view(-1, 1, 1, 1)
        epsilon = torch.randn_like(x)
        return sqrt_alphas_cumprod * x + sqrt_one_minus_alphas_cumprod * epsilon, epsilon

    def sample(self, num_samples, device):
        x = torch.randn(num_samples, 3, 64, 64).to(device)
        for t in range(self.num_steps - 1, -1, -1):
            t_batch = torch.full((num_samples,), t, device=device, dtype=torch.long)
            with torch.no_grad():
                predicted_noise = self.unet(x, t_batch)
                alpha = self.alphas[t]
                alpha_cumprod = self.alphas_cumprod[t]
                beta = self.betas[t]
                if t > 0:
                    noise = torch.randn_like(x)
                else:
                    noise = torch.zeros_like(x)
                x = 1 / torch.sqrt(alpha) * (x - ((1 - alpha) / (torch.sqrt(1 - alpha_cumprod))) *
predicted_noise) + torch.sqrt(beta) * noise
        return x

class MinecraftSkinDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        img_path = self.df.iloc[idx]['path']
        image = Image.open(img_path)

        # Convert palette images with transparency to RGBA
        if image.mode == 'P':
            image = image.convert('RGBA')

        # If the image is RGBA, blend it with a white background
        if image.mode == 'RGBA':
            background = Image.new('RGBA', image.size, (255, 255, 255))
            image = Image.alpha_composite(background, image)

        # Convert to RGB
        image = image.convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image
```

```
def train(model, dataloader, num_epochs, device):
    optimizer = optim.Adam(model.parameters(), lr=1e-4)
    criterion = nn.MSELoss()

    for epoch in range(num_epochs):
        model.train()
        epoch_loss = 0
        for batch in tqdm(dataloader, desc=f"Epoch {epoch+1}/{num_epochs}"):
            optimizer.zero_grad()

            x = batch.to(device)
            t = torch.randint(0, model.num_steps, (x.shape[0],), device=device).long()

            x_noisy, noise = model.add_noise(x, t)
            predicted_noise = model(x_noisy, t)

            loss = criterion(predicted_noise, noise)
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss/len(dataloader):.4f}")

        # Generate and save samples every 5 epochs
        if (epoch + 1) % 1 == 0:
            print(f"Generating samples for epoch {epoch+1}...")
            model.eval()
            with torch.no_grad():
                samples = model.sample(num_samples=5, device=device)
            save_generated_samples(samples, epoch + 1)

            # Save the model checkpoint
            torch.save(model.state_dict(), f"model_checkpoint_epoch_{epoch+1}.pth")

    return model

def save_generated_samples(samples, epoch):
    # Denormalize the samples
    samples = (samples + 1) / 2.0
    samples = samples.clamp(0, 1)

    # Create a directory for saving samples if it doesn't exist
    os.makedirs("generated_samples", exist_ok=True)

    # Plot and save the samples
    fig, axes = plt.subplots(1, len(samples), figsize=(20, 4))
    for i, sample in enumerate(samples):
        img = sample.cpu().permute(1, 2, 0).numpy()
        axes[i].imshow(img)
        axes[i].axis('off')

    plt.tight_layout()
    plt.savefig(f"generated_samples/epoch_{epoch}.png")
    plt.close()


# Main pipeline
def main():
    # Hyperparameters
```

```
    batch_size = 64
    num_epochs = 100
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # Create U-Net and Diffusion Model
    unet = UNet(in_channels=3, out_channels=3).to(device)
    diffusion_model = DiffusionModel(unet).to(device)

    # Load the DataFrame with image paths
    df = pd.read_csv('/kaggle/input/minecraft-diffusion/minecraft_skin_dataset_analysis.csv')

    df = df[df['height'] == 64]

    # Create dataset and dataloader
    transform = transforms.Compose([
        transforms.Resize((64, 64)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

    dataset = MinecraftSkinDataset(df, transform=transform)
    dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=4)

    # Train the model
    trained_model = train(diffusion_model, dataloader, num_epochs, device)

    # Save the final trained model
    torch.save(trained_model.state_dict(), "minecraft_skin_diffusion_model_final.pth")

    print("Training complete. Final model saved.")

    # Generate samples
    num_samples = 5
    generated_samples = trained_model.sample(num_samples, device)

    # Save generated samples
    save_generated_samples(generated_samples, epoch="final")

if __name__ == "__main__":
    main()
```

ДОДАТОК Б ТРЕНУВАННЯ GAN МОДЕЛІ

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision.utils import save_image

# Generator model with self-attention
class Generator(nn.Module):
    def __init__(self, latent_dim):
        super(Generator, self).__init__()
        self.latent_dim = latent_dim
        self.main = nn.Sequential(
            nn.ConvTranspose2d(self.latent_dim, 512, kernel_size=4, stride=1, padding=0),
            nn.BatchNorm2d(512),
            nn.ReLU(True),
            nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            SelfAttention(128),
            nn.ConvTranspose2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(True),
            nn.ConvTranspose2d(256, 4, kernel_size=4, stride=2, padding=1),
            nn.Tanh()
        )

    def forward(self, input):
        return self.main(input)



# Discriminator model with binary output for each image in the batch
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(4, 128, kernel_size=4, stride=2, padding=1),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout2d(0.5),  # Add dropout layer
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout2d(0.5),  # Add dropout layer
            nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
        )
        self.linear = nn.Linear(512 * 8 * 8, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, input):
        output = self.main(input)
        output = output.view(output.size(0), -1)  # Flatten the tensor
        output = self.linear(output)
```

```
        output = self.sigmoid(output)
        output = output.view(-1, 1)  # Reshape to have shape (batch_size, 1)
        return output


# Self-Attention module
class SelfAttention(nn.Module):
    def __init__(self, in_dim):
        super(SelfAttention, self).__init__()
        self.query_conv = nn.Conv2d(in_dim, in_dim // 8, kernel_size=1)
        self.key_conv = nn.Conv2d(in_dim, in_dim // 8, kernel_size=1)
        self.value_conv = nn.Conv2d(in_dim, in_dim, kernel_size=1)
        self.gamma = nn.Parameter(torch.zeros(1))

    def forward(self, x):
        m_batchsize, C, width, height = x.size()
        proj_query = self.query_conv(x).view(m_batchsize, -1, width * height).permute(0, 2, 1)
        proj_key = self.key_conv(x).view(m_batchsize, -1, width * height)
        energy = torch.bmm(proj_query, proj_key)
        attention = torch.softmax(energy, dim=1)
        proj_value = self.value_conv(x).view(m_batchsize, -1, width * height)
        out = torch.bmm(proj_value, attention.permute(0, 2, 1))
        out = out.view(m_batchsize, C, width, height)
        out = self.gamma * out + x
        return out
Train Model
import matplotlib.pyplot as plt

def generate_and_display_images(generator, latent_dim, device):
    # Generate random latent vectors
    latent_vectors = torch.randn(5, latent_dim, 1, 1, device=device)

    # Generate images using the generator
    generated_images = generator(latent_vectors).cpu()

    # Create a grid of generated images
    grid = torchvision.utils.make_grid(generated_images, nrow=5, normalize=True)

    # Display the generated images
    plt.figure(figsize=(10, 2))
    plt.imshow(grid.permute(1, 2, 0))
    plt.axis('off')
    plt.show()
from tqdm import tqdm
from PIL import Image
import torchvision

def train_gan(generator, discriminator, dataloader, num_epochs, latent_dim, lr, device,
batch_size):
    # Define loss function and optimizers
    criterion = nn.BCELoss()
    optimizer_G = optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))
    optimizer_D = optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))

    # Training loop
    total_steps = 0

    fixed_noise = torch.randn(batch_size, latent_dim, 1, 1, device=device)
```

```
    for epoch in range(num_epochs):
        progress_bar = tqdm(total=len(dataloader))

        for i, real_images in enumerate(dataloader):
            real_images = real_images.to(device)
            batch_size = real_images.size(0)

            # Train the discriminator
            discriminator.zero_grad()

            # Forward pass real images through discriminator
            output_real = discriminator(real_images)
            label_real = torch.full((batch_size, 1), 1.0, device=device)

#           print(label_real.shape, output_real.shape)

            errD_real = criterion(output_real, label_real)
            errD_real.backward()

            noise = torch.randn(batch_size, latent_dim, 1, 1, device=device)
            fake_images = generator(noise)

#           print(fake_images.shape, real_images.shape)


            # Forward pass fake images through discriminator
            output_fake = discriminator(fake_images.detach())
            label_fake = torch.full((batch_size, 1), 0.0, device=device)
            errD_fake = criterion(output_fake, label_fake)
            errD_fake.backward()

            errD = errD_real + errD_fake
            optimizer_D.step()

            # Train the generator
            generator.zero_grad()
            label_real = torch.full((batch_size, 1), 1.0, device=device)
            output = discriminator(fake_images)
            errG = criterion(output, label_real)
            errG.backward()
            optimizer_G.step()

            # Update progress bar
            progress_bar.set_postfix({'D loss': errD.item(), 'G loss': errG.item()})
            progress_bar.update()

            total_steps += 1

        progress_bar.close()

        # Generate and display images
        generate_and_display_images(generator, latent_dim, device)

        # Save generator model
        torch.save(generator.state_dict(), 'generator.pth')

        # Save discriminator model
        torch.save(discriminator.state_dict(), 'discriminator.pth')
```

```
        # Define the latent dimension
        latent_dim = 1000

        # Create the generator and discriminator models
        generator = Generator(latent_dim)
        discriminator = Discriminator()

        # Set device (GPU or CPU)
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        generator.to(device)
        discriminator.to(device)

        # Call the train_gan function
        train_gan(generator, discriminator, dataloader, num_epochs=200, latent_dim=latent_dim,
lr=0.0002, device=device, batch_size=batch_size)
```