МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Створення анімації за заданим стилем**

Виконав: студент 2 курсу, групи 8.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Комар А. П.
(ініціали та прізвище)
Керівник доцент, к.т.н. В. І. Заяц
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
П. О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри_____В.Г. Вербицький
"_01_" __вересня_____2020_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Комару Андрію Павловичу
(прізвище, ім'я, по батькові)

1.Тема роботи ___Створення анімації за заданим стилем_____
__

керівник роботи __Заяц Валерій Іванович, к.т.н., доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _"25"_травня_2020_року № 600-с_____

2. Строк подання студентом кваліфікаційної роботи___30.11.2020____
3. Вихідні дані магістерської роботи
- комплект нормативних документів;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми створення анімації за заданим стилем;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_
____слайдів презентації.

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2020

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.20	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.20	виконано
3	Аналіз існуючих методів рішення	13.09-17.09.20	виконано
4	Дослідження області анімації за заданим стилем	18.09-24.09.20	виконано
5	Узгодження подальших дій з науковим керівником	25.09-26.09.20	виконано
6	Аналіз теоретичних відомостей	27.09-15.10.20	виконано
7	Проектування функціоналу корекції стилістики анімації	15.10-23.10.20	виконано
8	Узгодження функціоналу з науковим керівником	23.10-24.10.20	виконано
9	Реалізація функціоналу корекції стилістики анімації	25.10-14.11.20	виконано
10	Представлення отриманих результатів науковому керівнику і узгодження плану подальшого дослідження	15.11-16.11.20	виконано
11	Реалізація інтерфейсу додатка	16.11-20.11.20	виконано
12	Проведення аналізу можливостей розроблених програмних за стосунків	21.11-23.11.20	виконано
13	Оформлення звіту	24.11-29.11.20	виконано

Студент _____ Комар А.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Заяц В.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 102

Рисунків: 37

Таблиць: 2

Джерел: 34

Комар А. П. Створення анімації за заданим стилем.

Кваліфікаційна робота для здобуття ступеня вищої освіти магістра за спеціальністю 121 – Інженерія програмного забезпечення, науковий керівник В.І. Заяц. Інженерний навчально-науковий інститут ЗНУ.

Мета кваліфікаційної роботи полягає у дослідженні наявних методів створення анімації за зображенням із заданим сюжетом, зміни стилістики зображення, накладення зображення на відео, допоміжних алгоритмів (створення 3d-моделі за зображенням, заповнення тла), можливостей їхньої комбінації з метою написання алгоритму створення анімації з заздалегідь фіксованим сюжетом, але з можливістю визначення стилістики та зовнішнього вигляду персонажів та декорацій власне користувачем). У нашому випадку буде використано метод зміни стилістики зображення Л. Ґетіса та створення скелетної анімації в середовищі Unity.

Досліджено методи і сучасні конкуруючі технології створення анімації з налаштуванням зовнішнього вигляду персонажів та корекції стилістики зображень. Для розробки використовувалися мова C# та середовище Unity, також почасти MySQL та XAMPP.

Ключові слова: *АНІМАЦІЯ, СТИЛІСТИКА, НЕЙРОННІ МЕРЕЖІ, C#, АЛГОРИТМ ҐЕТІСА, VGG19, UNITY, MYSQL, ДЕСКТОПНИЙ ЗАСТОСУНОК, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС.*

SUMMARY

Pages: 102

Figures: 37

Tables: 2

Sources: 34

Komar A.P. Creating animation for a given style.

Qualification work for obtaining a master's degree in specialty 121 — Software Engineering, supervisor V.I. Zayats. ZNU Engineering Educational and Scientific Institute.

The purpose of the qualification work is to study the existing methods of creating animation on the image with a given plot, changing the style of the image, overlaying the image on video, auxiliary algorithms (creating a 3D model on the image, filling the background), the possibility of combining them to write an animation algorithm. fixed plot, but with the ability to determine the style and appearance of the characters and scenery of the user). In our case, we will use the method of changing the stylistics of the image of L. Gatys and creating skeletal animation in the Unity environment.

Methods and modern competing technologies of animation creation with adjustment of appearance of characters and correction of stylistics of images are investigated. The C# language and the Unity environment were used for development, as well as some MySQL and XAMPP.

Keywords: *ANIMATION, STYLISTICS, NEURAL NETWORKS, C#, GATYS ALGORITHM, VGG19, UNITY, MYSQL, DESKTOP APPLICATION, USER INTERFACE.*

ЗМІСТ

ВСТУП 10
1.1 Алгоритми створення анімації за зображенням 13
1.2 Алгоритми зміни стилістики зображення 16
1.3 Алгоритми модифікації відео 24
1.4 Комбінації алгоритмів 25
1.5 Допоміжні алгоритми 25
1.6 Висновки 30
РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ СТВОРЕННЯ АНІМАЦІЇ З МОЖЛИВІСТЮ НАЛАШТУВАННЯ СТИЛІСТИКИ ТА ЗОВНІШНЬОГО ВИГЛЯДУ ДЕКОРАЦІЙ ТА ПЕРСОНАЖІВ 31
2.1 Створення анімації 31
2.1.1 Скелетна анімація 31
2.2 Надання стилістики 33
2.2.1 Алгоритм, заснований на моделі CCAN 33
2.2.2 Алгоритм, заснований на моделі DNN (алгоритм Ґетіса) 36
2.2.3 Алгоритм, заснований на моделі CCNN 38
2.3. Висновки 41
РОЗДІЛ 3 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ СТВОРЕННЯ АНІМАЦІЇ З МОЖЛИВІСТЮ НАЛАШТУВАННЯ СТИЛІСТИКИ ТА ЗОВНІШНЬОГО ВИГЛЯДУ ДЕКОРАЦІЙ ТА ПЕРСОНАЖІВ 42
3.1. Архітектура системи 42
3.1.1 Архітектура UI на Unity 42
3.1.2 Архітектура 3D-додатків на Unity 44
3.2 Засоби реалізації 50
3.2.1 Середовище розробки Visual Studio 2019 51
3.2.2 Мова програмування C# 52
3.2.3 Середовище розробки ігор Unity 53
3.2.4 Microsoft Cognitive Toolkit 54

3.2.5 Класифікатор зображень VGG19 64
3.2.6 Система управління базами даних MySQL 68
3.2.7 Середовище розробки MySQL Workbench 69
3.2.8 Мова програмування PHP 70
3.2.9 Збірка веб-сервера XAMPP 70
3.3 Модулі і алгоритми 71
3.3.1 Модуль інтерфейсу 71
3.3.2 Модуль отримання списку персонажів та прев'ю 77
3.3.3 Модуль створення відео 78
3.3.4 Модуль покадрової обробки відео 79
3.3.5 Модуль завантаження зображень до бази MySQL 82
3.4 Структури даних 83
3.5 Проект інтерфейсу 84
3.6 Вимоги до апаратного забезпечення 88
3.7 Опис функціональних можливостей 89
РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ СТВОРЕННЯ АНІМАЦІЇ З МОЖЛИВІСТЮ НАЛАШТУВАННЯ СТИЛІСТИКИ ТА ЗОВНІШНЬОГО ВИГЛЯДУ ДЕКОРАЦІЙ І ПЕРСОНАЖІВ 90
4.1 Дослідження результатів корекції стилістики за допомогою наявної реалізації алгоритму Ґетіса 90
4.1.1 Дослідження впливу кількості епох на результат 90
4.1.2 Дослідження оптимального розміру зображення 93
4.1.3 Дослідження можливостей зміни контурів зображення 96
ВИСНОВКИ 99
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 100

ВСТУП

Актуальність теми

У даний момент відбувається справжня революція у галузі нейронних мереж та штучного інтелекту. Важлива частина цих досягнень — рясне застосування цих технологій в галузі стилізації зображень та в галузі створення анімації (в тому числі й високого ступеня реалістичності). Найвідоміший з нещодавніх прикладів — DeepFake, який поставив під загрозу поняття відеодоказу як такого та призвів до необхідності створення нових методів розпізнання слідів роботи нейронних мереж. Також велетенську популярність отримують алгоритми обробки зображень: це й заповнення тла, закритого фігурою, й стилізація зображень, скажімо, під аніме-стилістику, стилістику живопису певного жанру чи напряму або навпаки, під фотореалістичність. Серед менш відомих розробок у цьому напрямі згадаємо про поточні дослідження у галузі створення відео за зображенням.

На наш погляд, цікавою буде спроба поставити ці досягнення на службу мистецтву. Першою на думку спадає анімація, яку відомий сучасний інтернет-діяч С. Штурхальов вважає чи не найголовнішим (або принаймні найперспективнішим) мистецтвом сучасності. Виражальні засоби комп'ютерної графіки дозволяють розповісти таку історію, на яку кіно (хай і з спецефектами) не спроможне. Відтак є потреба в розширенні методів та засобів комп'ютерної графіки, анімації та малюнка.

Серед проблем, які можна було б вирішити за допомогою нових технологій — проблема екранізацій, проблема, яку породжує будь-яка спроба екранізації будь-якого культового твору (чи правильно підібрано акторів? Чи коректно обрано стилістику фільму чи мультфільму?) Тобто має місце проблема суб’єктивного вибору стилістики та зовнішнього вигляду персонажів, чиї дії відомо заздалегідь. Тому проблема створення анімації з заздалегідь визначеним сюжетом та послідовністю рухів героїв, але при

цьому з можливістю користувацького налаштування загальної стилістики, декорацій та зовнішнього вигляду персонажей, видається нам досить актуальною.

Мета і завдання дослідження

Дослідити наявні методи створення анімації за зображенням із заданим сюжетом, зміни стилістики зображення, накладення зображення на відео, допоміжних алгоритмів (створення 3d-моделі за зображенням, заповнення тла), можливостей їхньої комбінації з метою написання алгоритму створення анімації з заздалегідь фіксованим сюжетом, але з можливістю визначення стилістики та зовнішнього вигляду персонажів та декорацій власне користувачем).

Об’єкт дослідження

Алгоритми створення анімації за фото чи художнім зображенням, визначення стилю зображення, трансформації стилю зображення, накладання зображення на анімацію, допоміжні алгоритми, їхня сумісність та комбінації.

Предмет дослідження

Можливість створення алгоритму створення анімації з заздалегідь фіксованим сюжетом, але з можливістю визначення стилістики та зовнішнього вигляду персонажів та декорацій власне користувачем).

Методи дослідження

Для розв’язання представлених завдань та цілей використовуються такі методи дослідження:

Аналіз джерел про маніпулювання 3D об’єктами в контексті комп’ютерної графіки

Навчання нейронних мереж (з учителем та без)

Наукова новизна

Одержані результати, перш за все, мають показати можливість створення анімації за заздалегідь заданим «кістяком» (як можливість створення анімації з фіксованим сюжетом (як-от екранізація якогось твору).

Практичне значення

На базі отриманих результатів можна створювати хоча б найелементарніші мультфільми за якимись творами, які ілюструвалися багато разів.

Апробація результатів

Результати роботи було представлено на науково-технічних конференціях студентів, магістрантів, аспірантів, молодих вчених.

Глосарій

Анімація (з лат. anima — душа і похідного фр. animation — оживлення), мультиплікація (з лат. multiplicatio — розмноження, збільшення, зростання) — вид кіномистецтва, твори якого створюються шляхом знімання послідовних фаз руху намальованих (графічна анімація) або об'ємних (об'ємна анімація) об'єктів.

Штучні нейронні мережі (ШНМ, англ. artificial neural networks, ANN), або конективістські системи (англ. connectionist systems) — це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ СТВОРЕННЯ АНІМАЦІЇ З МОЖЛИВІСТЮ НАЛАШТУВАННЯ СТИЛІСТИКИ ТА ЗОВНІШНЬОГО ВИГЛЯДУ ДЕКОРАЦІЙ ТА ПЕРСОНАЖІВ

1.1 Алгоритми створення анімації за зображенням

Проблема «оживлення» картинки на даний момент досліджена досить глибоко. Почнемо, наприклад, з анімації пейзажів. Серед важливих праць на цю тему згадаємо [6]. Дана робота розглядає проблему анімації води та вітру: під час аналізу пейзаж поділяється на декілька шарів різних типів (серед яких згадаємо такі, як «рослини», «вода», «хмари», «човни», «статичні об’єкти»), після чого на кожен з них накладається відповідна стохастична структура руху. Подібна модель чудово працює як із фотопейзажами, так і с класичними картинами, як-от «Соняшники» В. Ван Гога.

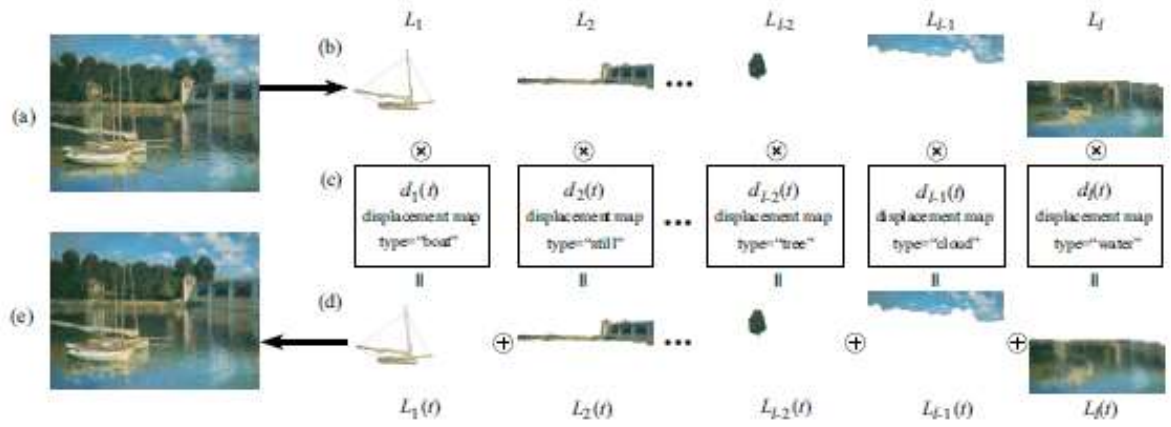

Рисунок *1 Алгоритм анімації пейзажів*

Складнішою є проблема анімації руху тварин (наприклад, слонів або птахів). З праць на цю тему згадаємо [26]. На відміну від руху неживих об’єктів, стаціонарно стохастичного, рух живих об’єктів є регулярним та повторюваним. Відтак алгоритм припускає, що зображена на вхідній картинці зграя птахів може бути розглянута як набір кадрів руху одного й того ж птаха. Відтак від картинки відокремлюються силуети тварин (тло

заповнюється), потім за допомогою метрики подібності кадрів будується граф їхньої подібності. Вже на його базі будується цикл руху тварин, якнайгладша функція руху від кадру до кадру, яка й застосовується.

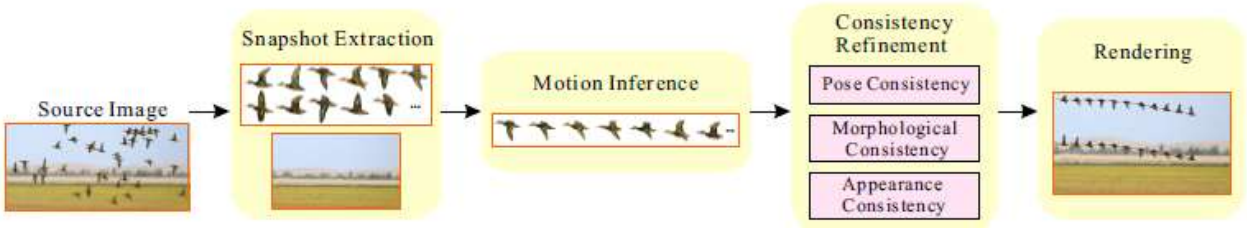

Рисунок *2 Алгоритм анімації тварин*

У праці [14] розглянуто реконструкцію тривимірних форм тих чи інших об’єктів (автомобілі) та їхнього повертання на фото.

Рисунок *3 Приведення до руху зображень людей*

Приведення у рух об’єктів-людей на зображеннях досліджене у [25]. Завдяки створеним алгоритмам людина з картинки може ходити, бігати, сидіти або ж навіть стрибати.

Алгоритм виокремлює на зображенні силует людини та сегментує його, співвідносячи намальовані частини тіла з вузлами мешу, за допомогою механізму Mask R-CNN. Далі будується SMPL-модель (Skinned Multy-Person Linear model — реалістична тривимірна модель людського тіла, побудована за принципом вершин), після чого ця модель деформується так, щоб її двовимірна проекція відповідала вхідній картинці. Деформація йде за двома

напрямками: деформується як карта глибин, так і карта шкіри, з яких будується підлаштований під вхідну картинку меш. Паралельно допоміжний механізм домальовує приховане силуетом тло. Далі вертекси мешу можна рухати запрограмованим чином, і відповідно буде рухатися силует людини.

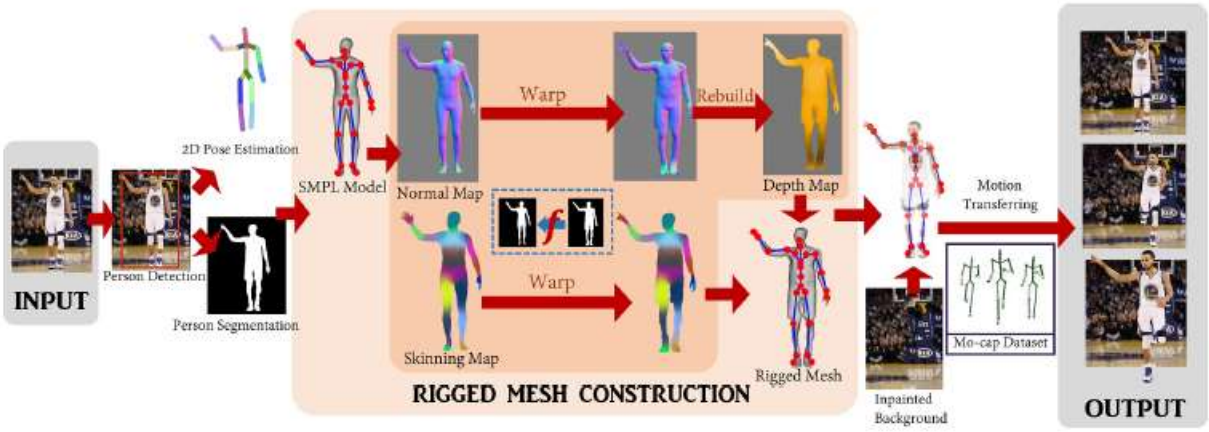

Рисунок *4 Алгоритм анімації зображень людей*

Також зазначимо дослідження, присвячені зміні виразів обличчя, зміні форм тіла [29], анімації людей на фото[12], анімації за допомогою 2d-картинок та 3d-моушн-даних [11] та анімації мови (на прикладі обличчя Барака Обами) [21].

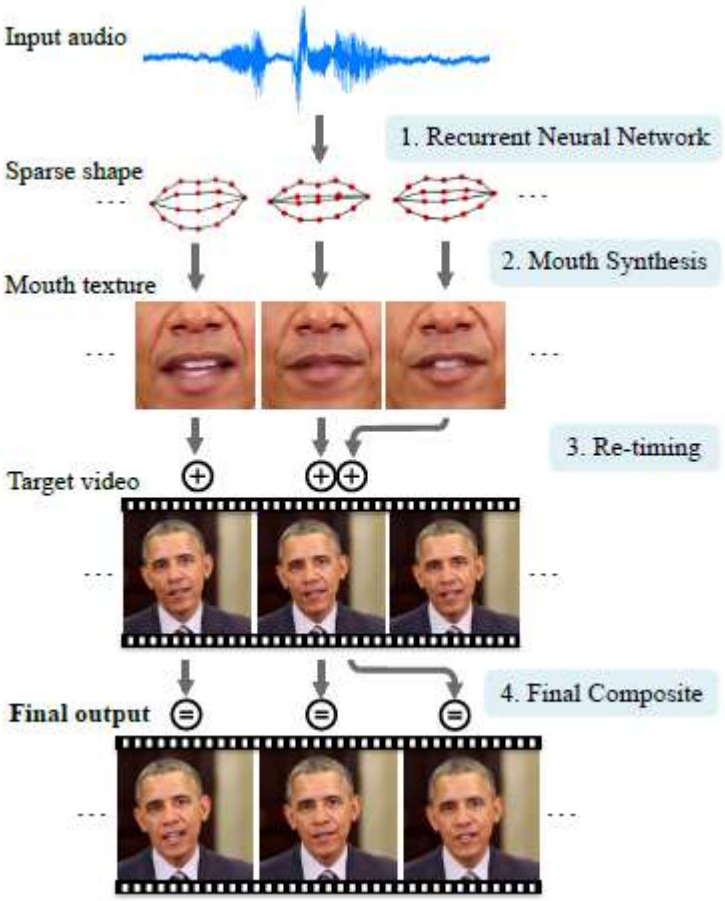

Рисунок 5 *Алгоритм анімації облич*

1.2 Алгоритми зміни стилістики зображення

Розв’язання проблеми зміни стилістики зображення, перемальовування зображення відповідно до задачі «як би це намалював Клод Моне», «як би це намалював Поль Сезанн» або «як би це намалював Хаяо Міядзакі», ускладнюється тим, що у такій задачі в нашому розпорядженні зазвичай нема парних картинок, за допомогою яких ми могли б навчати нейронну мережу.

Серед можливих розв’язків можна згадати, наприклад, варіант [29]. У даному дослідженні трансляція картинки з домену X до домену Y, мапінг G : X → Y відбувається через визначення спільних характеристик певного кластеру зображень та додання цих характеристик до іншого кластеру зображень за умов відсутності точних пар.

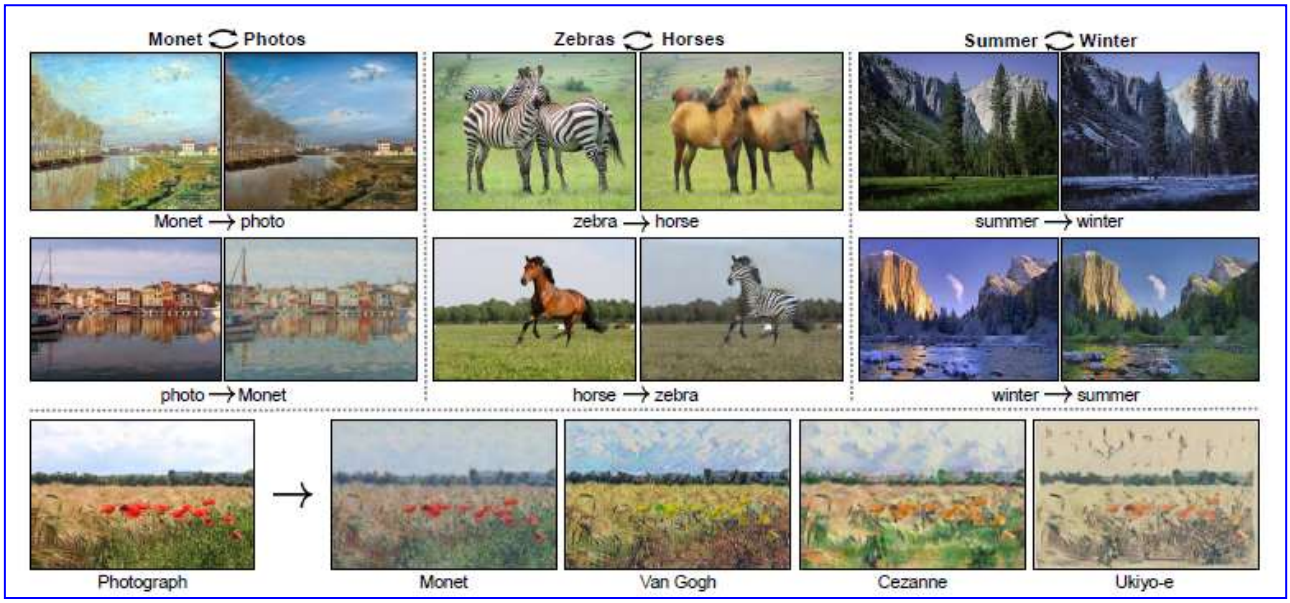

Рисунок 6 *Трансформація зображення за кластерним принципом*

Припускається, що існує деяке неявне відношення між доменами (наприклад, це два різних рендерингу однієї й тієї ж сцени), та йде пошук цього відношення. Відсутність прикладів точних пар картинок, які можна було б використовувати для навчання, компенсується навчанням на рівні множин: дається множина зображень у домені X та у домені Y. Можна навчати нейронну мережу такому перекладу $G : X \rightarrow Y$, щоб вивідне $\hat{y} = G(x)$, $x \in X$, розпізнавалося змагальною мережею (натренованою для відрізнення $\hat{y}$ від $y \in Y$) як таке, яке неможливо відрізнити від зображень y.

Однак на практиці подібне розрізнення не обов'язково буде іти осмисленими шляхами (можна уявити нескінченну множину ознак, які вирізняють множину Y), крім того, на практиці цілком можлива проблема колапсу, коли нейронна мережа видаватиме одне й те саме зображення у відповідь на будь-яке. Тому додається вимога про *циклічну несуперечливість* (cycle consistency) циклу: до функції G додається функція $F : Y \rightarrow X$, і важлива умова успіху — інвертованість цих функцій, так, щоб $F(G(x)) \approx x$ та $G(F(y)) \approx y$.

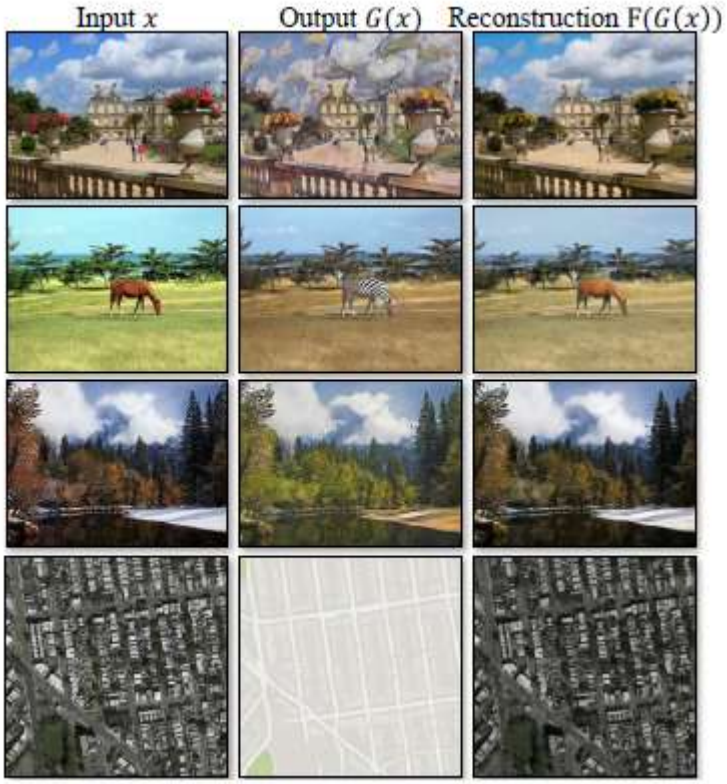

Рисунок *7. Приклад трансформації*

Даний алгоритм реалізовано за допомогою технології GAN (Generative Adversial Network, генеративна змагальна мережа). Це клас алгоритмів штучного інтелекту, що використовуються в навчанні без учителя, реалізовані системою двох штучних нейронних мереж, які змагаються одна з одною в рамках гри з нульовою сумою. Вони були запроваджені Яном Ґудфелоу в 2014 році. Ця методика дозволяє створювати фотографії, які для побіжного огляду людиною виглядають як справжні та мають багато реалістичних елементів (хоча в тестах люди можуть відрізнити реальні зображення від згенерованих у багатьох випадках). Одна мережа генерує кандидатів (генератор), а інша оцінює їх (дискримінатор) [10]. Як правило, генеративна мережа навчається будувати відповідності з латентного простору до певного розподілу даних, тоді як дискримінаційна мережа розрізняє представників справжнього розподілу даних та кандидатів, вироблених генератором. Метою тренувальної мережі є збільшення частоти помилок дискримінаційної мережі (тобто «обдурити» дискримінатор шляхом

створення нових синтезованих екземплярів, які повинні походити на представників справжнього розподілу даних).

На аналогічних принципах також ґрунтується робота [5], яка просто надає картинці мультиплікаційного стилю.

Рисунок *8. Трансформація стилю зображення*

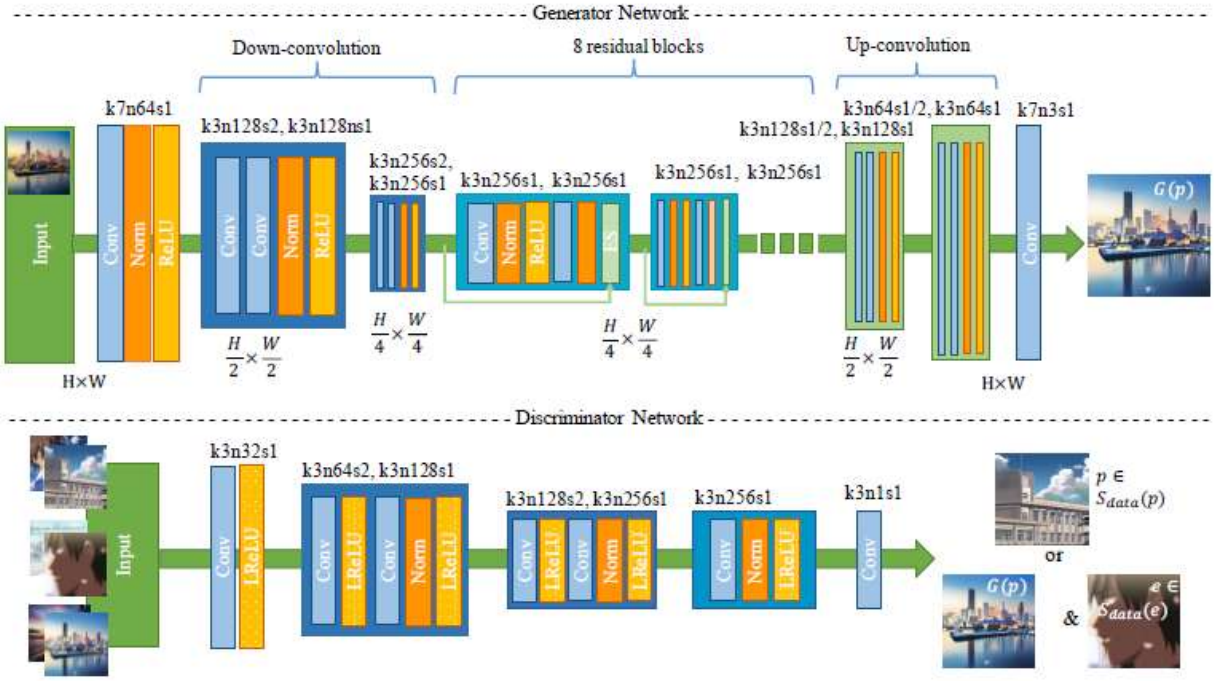

Рисунок *9. Алгоритм трансформації стилю зображення*

Дослідження [9] використовує для трансформації стилю зображення алгоритм, заснований на моделі DNN (Deep Neural Network, глибока нейронна мережа). Це штучна нейронна мережа (ШНМ) з декількома прихованими шарами вузлів між вхідним та вихідним шарами. Подібно до пласких ШНМ, ГНМ можуть моделювати складні нелінійні відношення. Архітектури ГНМ, наприклад, для виявлення об'єктів та граматичного аналізу, породжують композиційні моделі, де об'єкт виражається як шарувата композиція примітивів зображення. Додаткові шари дозволяють композиції включати ознаки з нижчих шарів, забезпечуючи потенціал для моделювання складних даних меншою кількістю вузлів, ніж настільки ж ефективна пласка мережа.

ГНМ зазвичай проектуються як мережі прямого поширення, але дослідження дуже успішно застосували рекурентні нейронні мережі, особливо ДКЧП, до таких задач, як моделювання мов. Згорткові глибинні нейронні мережі (ЗНМ, англ. convolutional deep neural networks, CNN) застосовуються в комп'ютерному зорі, де їхній успіх є добре задокументованим. ЗНМ також було застосовано до акустичного моделювання для автоматичного розпізнавання мовлення (АРМ, англ. automatic speech recognition, ASR), де вони продемонстрували переваги над попередніми моделями.

Ключове спостереження вищезазначеного дослідження полягає в тому, що під час розпізнання об’єктів на зображеннях зображення у нейронних мережах репрезентуються таким чином, що зміст (content) відокремлюється від стилю. А отже, виникає можливість їхнього рекомбінування у потрібному ключі. Задля репрезентації стилю використовується функція простору ознак, спершу спроектована для збереження інформації про текстуру.

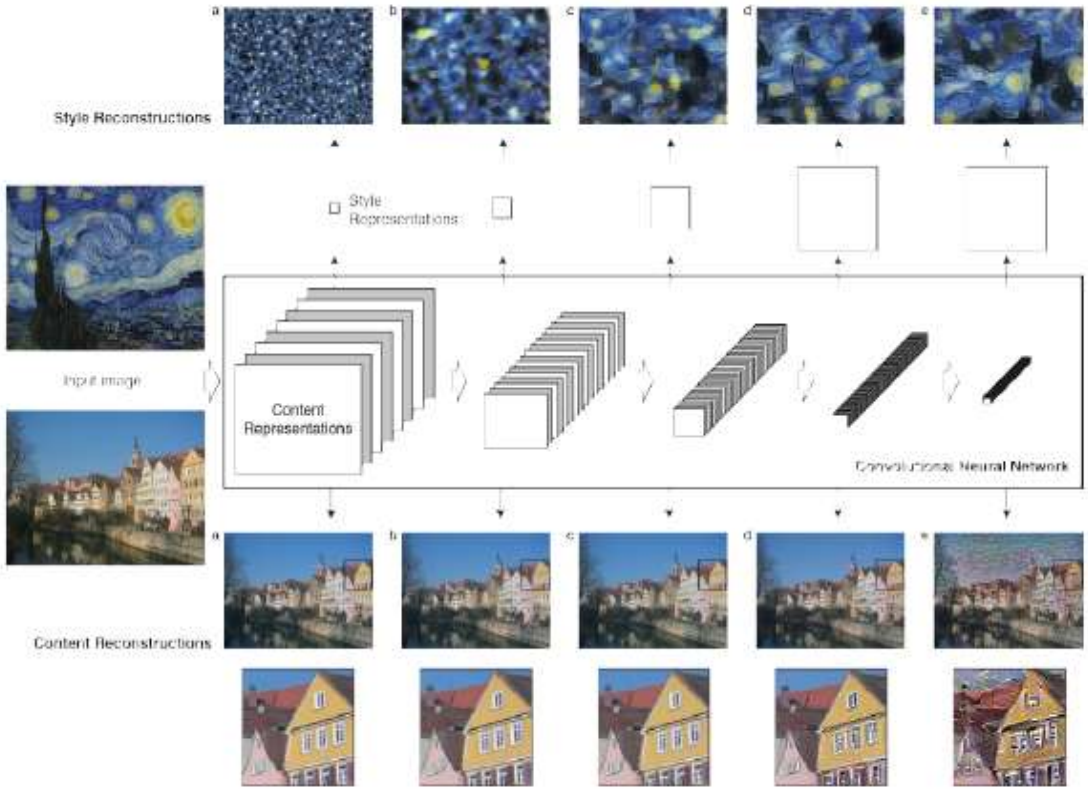

Рисунок *10. Алгоритм відокремлення стилю від змісту*

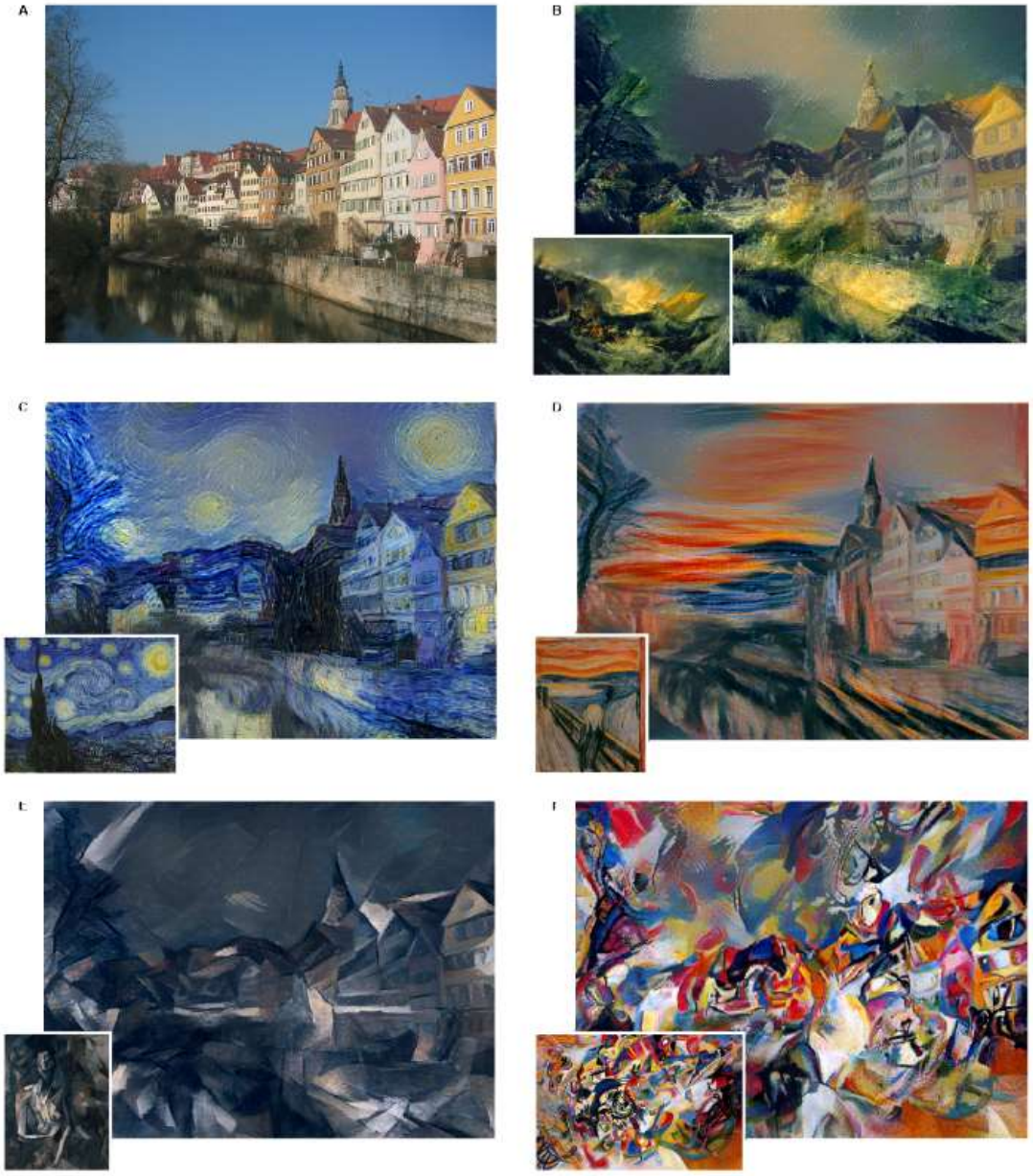

Рисунок *11. Приклади перевизначення стилю*

Модифікація зображень (людських облич) у ключі підкреслення певних специфічних рис обличчя у гумористичному ключі, Exaggerating the Difference from the Mean (EDFM), лежить в основі такого художнього явища, як карикатура. При цьому, звичайно, художній стиль будь-якого карикатуриста є унікальним. У праці [20] теоретично обґрунтовано та експериментально підтверджено можливість захоплення художнього стиля певного карикатуриста та створення карикатур у цьому стилі за фотографією.

В основу алгоритму закладено технологію каскадно-корелятивних нейронних мереж (CCNN): це генеративний алгоритм навчання нейронних мереж прямого поширення з учителем, де нейронна мережа будується безпосередньо під час навчання. Спершу структура дуже проста (один нейрон), далі поступово за умов необхідності додаються нові й нові нейрони. При цьому вага вже доданого та навченого нейрона стає константою й не може бути зміненою. Подібне збереження дозволяє CCNN акумулювати досвід після ініціального сеансу навчання. Крім того, цей алгоритм досить швидко вчиться (в 10 разів швидше, аніж більшість алгоритмів зворотного поширення), дозволяє будувати власну архітектуру та є корисним у кумулятивному навчанні, де один раз додана у систему інформація має зберігатися.

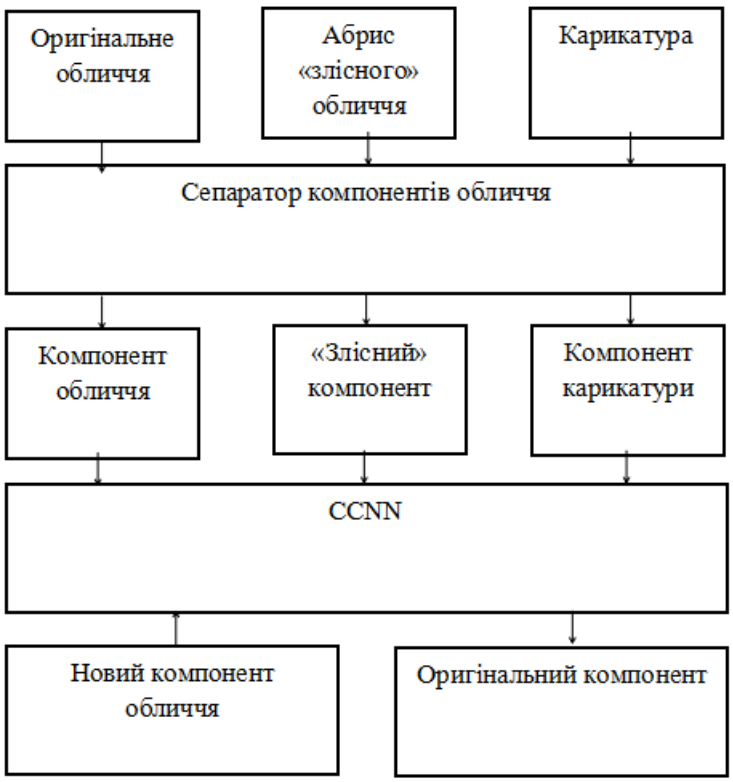

Рисунок *12 Алгоритм захоплення стилю карикатури*

У процесі навчання використовується сепаратор компонентів обличчя, який співвідносить між собою оригінал обличчя, карикатуру у певному стилі

та скетч «злісного» виразу обличчя (намальований програмістами власноруч або ж згенерований за допомогою певної програми), розбиваючи усі три на компоненти (очі, ніс, рот та ін.) та подаючи оригінальні компоненти та скетчі на вхід нейронної мережі, карикатурні ж — на вихід. Після такого навчання у нейронну мережу можна завантажити фото певної частини обличчя, і буде автоматично згенеровано її карикатурний аналог у певному стилі. Таким чином, хоча про повноцінне створення карикатур говорити ще рано, вагоме наближення до мети має місце.

1.3 Алгоритми модифікації відео

Deepfake, конкатенація слів «глибинне навчання» (англ. deep learning) та «підробка» (англ. fake) — методика синтезу зображення людини, яка базується на штучному інтелекті. Вона використовується для поєднання і накладення існуючих зображень та відео на вихідні зображення або відеоролики.

Мета, з якою цю технологію було розроблено, далеко не була етичною: Deepfake з'явився в Інтернеті в 2017 році, зокрема на Reddit, як програма для створення шок-контенту. На даний момент подібний створений за допомогою Deepfake контент суворо переслідується, однак deepfake-ролики прийнятного змісту можна легко знайти на популярних веб-сайтах потокового відео, таких як YouTube або Vimeo. Популярною програмою є FakeApp, який використовує TensorFlow.

Методи виявлення жестів та перетворення на цільове відео, яке схоже на цільову особу, були представлені в 2016 році і дозволяють створення в режимі реального часу підробних мімічних виразів в 2D-відео [23].

Поширення Deepfake викликає побоювання у багатьох сучасних діячів. Гарячим прихильником заборони Deepfake є, наприклад, відомий канадський психолог та громадський діяч Джордан Б. Пітерсон, який сам неодноразово страждав від фейків з його участю [19].

Через важкі етичні проблеми, пов’язані з алгоритмами модифікації відео, у даній праці ми вимушені не звертатися до цих засобів. Згадаємо лише роботу [2], присвячену зміні стиля мови.

1.4 Комбінації алгоритмів

Розглянувши різноманітні підходи до розв’язання підзадач, з яких складається поставлена нами проблема, перейдемо до аналізу підходів, де ці підзадачі так чи інакше скомбіновано. Нам вдалося знайти одну таку роботу [27]. Дана робота присвячена створенню моделей «балакучих голів» за фотографією. Потребуючи навчання на досить великому датасеті, створений алгоритм здатний далі створювати нейронні моделі «балакучих голів» попередньо невідомих системі людей лише за однією фотографією. Більш того, є можливість навіть знизити кількість даних, потрібних для тренування датасету.

Рисунок *13 Створення «балакучих голів»*

1.5 Допоміжні алгоритми

У перелічених вище працях неодноразово виникає проблема заповнення тла, прихованого фігурою. Сучасні нейронні мережі дозволяють інтерполювати прихований тим чи іншим силуетом фон, через швидкий

рандомізований підбір найближчих сусідніх відповідностей між фрагментами картинки [3].

По мірі розвитку цифрової та обчислювальної фотографії дослідники розробили методи редагування цифрових фотографій та відео на високому рівні для досягнення набору бажаних цілей. Наприклад, останні алгоритми ретаргетингу зображень дозволяють змінити розмір зображень до нового співвідношення сторін — комп'ютер автоматично створює хорошу подобу вмісту вихідного зображення, але з новими розмірами. Інші алгоритми для завершення зображення дозволяють користувачеві просто стерти небажану частину зображення, а комп'ютер автоматично синтезує область заповнення, яка правдоподібно відповідає решті зображення. Алгоритми перестановки зображень дозволяють захоплювати частини зображення та переміщувати їх навколо — комп'ютер автоматично синтезує залишок зображення так, щоб нагадувати оригінал, поважаючи переміщені регіони.

У кожному з цих сценаріїв взаємодія користувача є важливою з кількох причин: По-перше, ці алгоритми іноді вимагають втручання користувача для отримання найкращих результатів. Наприклад, алгоритми повторного націлення іноді надають користувачеві елементи керування, щоб вказати, що одну або кілька регіонів (наприклад, обличчя) слід залишати відносно незмінними. Так само найкращі алгоритми завершення пропонують інструменти для керування результатом, надаючи підказки для комп'ютера. Ці методи забезпечують такий контроль, оскільки користувач намагається оптимізувати набір цілей, відомих йому, а не комп’ютеру. По-друге, користувач часто навіть не може апріорі сформулювати ці цілі. Художній процес створення бажаного зображення вимагає використання проб і помилок, оскільки користувач прагне оптимізувати результат стосовно особистих критеріїв, характерних для розглянутого зображення.

Роль інтерактивності в художньому процесі передбачає дві властивості ідеальної рамки редагування зображень: (1) набір інструментів повинен забезпечити гнучкість для виконання широкого спектру безшовних операцій

редагування для користувачів, щоб вивчити свої ідеї; та (2) продуктивність цих інструментів повинна бути досить швидкою, щоб користувач швидко бачив проміжні результати в процесі проб і помилок. Більшість підходів до редагування на високому рівні відповідають лише одному з цих критеріїв. Наприклад, було показано, що одне сімейство алгоритмів, відоме вільно як непараметрична вибірка виправлень, виконує ряд завдань редагування, виконуючи перший критерій — гнучкість. Ці методи ґрунтуються на невеликих (наприклад, 7x7) щільно відібраних виправленнях у декількох масштабах і здатні синтезувати як текстуру, так і складні структури зображення, які якісно нагадують вхідні зображення. Через їх здатність зберігати структури, ми називаємо цей клас прийомів структурним редагуванням зображень. У контексті цих методів було розроблено алгоритм, який прискорює такі методи принаймні на порядок, дозволяючи застосовувати їх у структурі інтерактивного структурного редагування зображень.

Щоб зрозуміти цей алгоритм, ми повинні розглянути загальні компоненти цих методів:

Основним елементом методів непараметричного відбору виправлень є повторний пошук усіх патчів в одній області зображення за найбільш подібним патчем в іншій області зображення. Іншими словами, задані зображення або регіони A і B, знайдіть для кожного виправлення найближчий сусід у B під метрикою відстані патча, такою як Lp. Ми називаємо це відображення Найближчим сусіднім полем (NNF), схематично проілюстрованим на вставній фігурі.

Підхід до цієї проблеми при пошуку найсильнішої грубої сили є дорогим — O (mM2) для областей зображення та патчів розміром M та m пікселів відповідно. Навіть використовуючи методи прискорення, такі як приблизні найближчі сусіди та зменшення розмірності, цей крок пошуку залишається вузьким місцем непараметричних методів вибірки виправлень, не дозволяючи їм досягти інтерактивної швидкості. Крім того, ці структури

прискорення на основі дерев використовують пам'ять у порядку O (M) або вище з відносно великими константами, обмежуючи їх застосування для зображень з високою роздільною здатністю.

Для ефективного обчислення приблизних полів найближчого сусіда наш новий алгоритм спирається на три ключові спостереження щодо проблеми:

Розмір простору зміщення. По-перше, хоча розмірність простору патча велика (m розмірів), він малонаселений (O (M) патчів). Багато попередніх методів прискорили пошук найближчого сусіда, атакуючи розмірність простору патча за допомогою деревних структур (наприклад, kd-дерево, яке може здійснювати пошук за час O (mMlogM)) та методами зменшення розмірності (наприклад, PCA). На відміну від цього, наш алгоритм здійснює пошук у двовимірному просторі можливих зміщень патчів, досягаючи більшої швидкості та ефективності пам'яті.

Природна структура образів. По-друге, звичайний швидкий пошук кожного пікселя ігнорує природну структуру зображень. В алгоритмах синтезу виправлень патчів вихідний висновок зазвичай містить великі суміжні шматки даних із вхідних даних (як це спостерігалось Ашіхміном). Таким чином, можна підвищити ефективність, здійснюючи пошук сусідніх пікселів взаємозалежно.

Закон великої кількості. Нарешті, хоча будь-який вибір випадкового призначення патчів навряд чи буде хорошим здогадом, деяка нетривіальна частка великого поля випадкових призначень, ймовірно, буде хорошими здогадами. Із збільшенням цього поля шанс, що жоден патч не матиме правильного зміщення, стає зникаючим малим. На основі цих трьох спостережень запропоновано рандомізований алгоритм для обчислення приблизних NNF з використанням поступових оновлень. Алгоритм починається з початкової здогадки, яка може бути отримана з попередньої інформації або може бути просто випадковим полем.

Ітераційний процес складається з двох фаз: розповсюдження, в якому когерентність використовується для розповсюдження хороших розчинів суміжних пікселів у полі; та випадковий пошук, при якому поточний вектор зміщення обурений множинними масштабами випадкових зсувів. Ми показуємо як теоретично, так і емпірично, що алгоритм має хороші властивості конвергенції для тестованих зображень до 2 Мп, а наша реалізація процесора показує прискорення в 20-100 разів порівняно з kd-деревами з PCA. Крім того, ми пропонуємо реалізацію GPU, яка приблизно в 7 разів швидша, ніж версія CPU для подібних розмірів зображення. Наш алгоритм вимагає зовсім небагато додаткової пам’яті поза вихідним зображенням, на відміну від попередніх алгоритмів, які будують допоміжні структури даних для прискорення пошуку. Використовуючи типові параметри параметрів нашого алгоритму, час виконання — O (mMlogM), а використання пам'яті — O (M). Хоча це той самий асимптотичний час і пам’ять, як найбільш ефективні методи прискорення на основі дерев, провідних констант значно менше.

Система застосування цього алгоритму в контексті структурної програми редагування зображень з трьома режимами інтерактивного редагування: перенацілювання зображення, завершення зображення та перестановка зображень. включає набір інструментів, які пропонують додатковий контроль над попередніми методами, дозволяючи користувачеві обмежувати процес синтезу інтуїтивним та інтерактивним способом.

Внесок даної роботи включає швидкий алгоритм рандомізованого наближення для обчислення поля найближчого сусіда між двома ділянками зображення, що перебувають між собою; застосування цього алгоритму в рамках структурного редагування зображень, що забезпечує якісну інтерактивну перенацілювання зображення, завершення зображення та перестановку зображень; і набір інтуїтивних інтерактивних елементів керування, які використовуються для обмеження процесу оптимізації для отримання бажаних творчих результатів.

Також згадаємо алгоритм класифікації зображень за стилями [15], алгоритм відновлення фігури та пози за фотографією [4], алгоритм визначення пози [24].

1.6 Висновки

Як бачимо, тема створення анімації у заданому стилі комп’ютерними засобами досі не розроблялася науково. Однак наближення до неї є, а відтак обрана нами тема цілком актуальна.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ СТВОРЕННЯ АНІМАЦІЇ З МОЖЛИВІСТЮ НАЛАШТУВАННЯ СТИЛІСТИКИ ТА ЗОВНІШНЬОГО ВИГЛЯДУ ДЕКОРАЦІЙ ТА ПЕРСОНАЖІВ

Після побіжного огляду сучасного стану справ у алгоритмах створення анімації та налаштування стилістики спробуємо сформулювати, якою нам видається методологія вирішення поставленої задачі.

2.1 Створення анімації

Перший етап, який передує власне роботі програми — власне створення сценарію мультфільму та його покадрове прописування. В цьому контексті розглянемо та спробуємо порівняти деякі сучасні технології анімації.

2.1.1 Скелетна анімація

Скелетна анімація — спосіб анімації тривимірних моделей в мультиплікації і комп'ютерних іграх. Полягає в тому, що мультиплікатор або моделер створює скелет, який представляє собою як правило древообразную структуру кісток, в якій кожна наступна кістка «прив'язана» до попередньої, тобто повторює за нею руху і повороти з урахуванням ієрархії в скелеті. Далі кожна вершина моделі «прив'язується» до будь-якої кістки скелета. Таким чином, при русі окремої кістки рухаються і всі вершини, прив'язані до неї. Завдяки цьому завдання аніматора сильно спрощується, тому що відпадає необхідність анімувати окремо кожну вершину моделі, а досить лише задавати положення і поворот кісток скелета.

Також завдяки такому методу скорочується і обсяг інформації, необхідної для анімації. Досить зберігати інформацію про рух кісток, а руху вершин вираховуються вже виходячи з них.

Скелетна анімація з розважуванням — більш просунутий варіант скелетної анімації, в ній кожна вершина моделі може бути пов'язана не з однією, а з кількома кістками. При цьому для кожної кістки визначається свою вагу, тобто величина впливу цієї кістки на переміщення вершини. Чим більше вага якийсь кістки, тим сильніше вершина зміщується під її впливом.

Завдяки розважуванню можна анімувати плавні вигини поверхонь, руху тканин, прапори і т. п.

Також скелетна анімація як технологія використовується в двовимірної комп'ютерної мультиплікації, наприклад, у векторному редакторі мультиплікації Anime Studio (колишня Moho) або відкритому Synfig Studio. Принцип дії такий же: окремі фрагменти моделі «прив'язуються» до кісток, і далі мультиплікатор, замість того щоб малювати модель в кожному кадрі, задає руху кісткам. Модель персонажа рухається, повторюючи рух скелета.

Техніку використовують практично усі анімаційні системи, де спрощений користувацький інтерфейс дозволяє легко маніпулювати складними алгоритмами і великою кількістю геометрії; особливо вирізняється інверсна кінематика та інші техніки планування руху. Проте, імітація реальної анатомії чи фізичних процесів ніколи не були метою скелетної анімації, лише контроль деформації мешу.

Переваги:

Кістка представляє множину вершин (або інших об'єктів, які утворюють, наприклад, руку);

Аніматорові доводиться контролювати меншу кількість характеристик моделі;

Аніматор може зосередитись на рухові великих масштабів;

Кістки можуть рухатись незалежно;

Анімація визначається простим переміщенням кісток, а не повершинно(у випадку полігональної сітки).

Недоліки:

Не забезпечується реалістичний рух моделі та шкіри.

Можливе рішення — спеціальні м'язові контролери, приєднані до кісток.

Скелетна анімація — стандартний спосіб анімувати персонажів чи механічні частини на довгий проміжок часу(більше 100 кадрів). Її зазвичай використовують ігрові дизайнери та у кінематографі, але можна застосувати й до механічних пристроїв ті інших об'єктів з рухомими деталями.

Технологія захоплення руху дозволить прискорити створення скелетної анімації, водночас підвищуючи рівень реалізму.

Якщо захоплення руху використовувати занадто небезпечно, можна скористатись цифровим моделюванням, щоб автоматично розрахувати фізику руху і опору для скелета. Можна додати властивості, що відносяться до віртуальної анатомії, такі як вага кінцівок, м'язова реакція, міцність кісток і рухливість суглобів. Це дозволить виконувати віртуальні трюки — стрибки, вигини, переломи і т.д. Також віртуальна анатомія застосовується у військовій та медичній галузях. Віртуальних солдат, рятівників, пасажирів і потерпілих використовують для тренування, віртуальної інженерії та віртуального тестування спорядження. Технології віртуальної анатомії можуть бути об'єднані з штучним інтелектом для подальшого розвитку анімації та моделювання.

2.2 Надання стилістики

На цьому етапі відбувається покадрове стилістичне редагування анімації відповідно до обраної стилістики.

Побіжний огляд алгоритмів, які можуть підійти для такої мети, ми надали в 1.2. У даному розділі місце розглянути їх детальніше.

2.2.1 Алгоритм, заснований на моделі CCAN

Даний алгоритм, як ми зазначали вище, будується на встановленні взаємно однозначних відповідностей між доменами зображень X та Y, із

заданими навчальними зразками $\{x_i\}_{i=1}^{N}$, де $x_i \in X$, та $\{y_j\}_{j=1}^{M}$, де $y_j \in Y$. Позначаємо розподіл даних як $x \sim p_{data}(x)$ та $y \sim p_{data}(y)$. Як проілюстровано на малюнку 3 (а), наша модель включає два відображення — $G: X \to Y$ та $F: Y \to X$. Крім того, ми представляємо два змагальних дискримінатори D_X та D_Y, де D_X має на меті розрізнити зображення $\{x\}$ та перекладені зображення $\{F(y)\}$; таким же чином D_Y прагне розрізняти $\{y\}$ та $\{G(x)\}$. Алгоритм містить два типи термінів: змагальні втрати (*adversarial losses*) для узгодження розподілу сформованих зображень із розподілом даних у цільовому домені; і циклічні втрати консистенції (*cycle consistency losses*), щоб запобігти протиставленню вивчених відображень G і F.

Ми застосовуємо змагальні втрати [16] до обох функцій відображення. Для функції відображення $G: X \to Y$ та її дискримінатора D_Y ми виражаємо ціль як:

$$L_{GAN}(G, D_Y, X, Y) = E_{y \sim p_{data}(y)}[\log D_Y(y)] + E_{x \sim p_{data}(x)}[1 - \log D_Y(G(x))] \quad (1)$$

де *G* намагається генерувати зображення *G(x)*, які схожі на зображення з домену *Y*, тоді як D_Y має на меті розрізнити перекладені зразки *G(x)* та реальні зразки *y*. G прагне мінімізувати цю мету проти супротивника D, який намагається її максимізувати, тобто $\min_G \max_{D_Y} L_{GAN}(G, D_Y, X, Y)$.

Введемо подібні змагальні втрати для функції відображення *F: Y → X* та її дискримінатора D_X: $\min_F \max_{D_X} L_{GAN}(F, D_Y, Y, X)$.

В теорії змагання можна теоретично вивчити відображення G і F, які дають результати, ідентично розподілені як цільові доменів Y та X відповідно (строго кажучи, для цього G і F потребують стохастичних функцій). Однак, маючи достатньо велику пропускну здатність, мережа може зіставити один і той же набір вхідних зображень з будь-якою випадковою

перестановкою зображень у цільовому домені, де будь-яке з вивчених відображень може викликати вихідний розподіл, який відповідає цільовому розподілу. Таким чином, суперечливі втрати самі по собі не можуть гарантувати, що вивчена функція може відобразити окремий вхід x_i до бажаного результату y_i. Для подальшого зменшення простору можливих функцій відображення, ми стверджуємо, що вивчене відображення функції повинні відповідати циклу: для кожного зображення x із домену X цикл перекладу зображень повинен мати можливість повернути x до вихідного зображення, тобто $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. Ми називаємо це послідовністю прямого циклу. Подібним чином для кожного зображення y з домену Y, G та F також повинно задовольняти узгодженість циклу назад: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$.

Стимулюється така поведінка, використовуючи втрату послідовності циклу:

$$\begin{aligned} L_{cyc}(G,F) &= E_{x\sim p_{data}(x)}\left[\left\|F(G(x))-x\right\|_1\right] \\ &+ E_{y\sim p_{data}(y)}\left[\left\|G(F(y))-y\right\|_1\right] \end{aligned} \tag{2}$$

Повна мета:

$$\begin{aligned} L(G,F,D_X,D_Y) &= L_{GAN}(G,D_Y,X,Y) \\ &+ L_{GAN}(F,D_X,X,Y) \\ &+ \lambda L_{cyc}(G,F) \end{aligned} \tag{3}$$

де λ контролює відносну важливість двох цілей. Ми прагнемо вирішити:

$$G^*,F^* = \arg\min_{G,F}\max_{D_X,D_Y} L(G,F,D_X,D_Y) \tag{4}$$

Зверніть увагу, що модель можна розглядати як навчання двох «автокодерів»: ми вивчаємо один автокодер $F \circ G: X \rightarrow X$ спільно з іншим $G \circ F: Y \rightarrow Y$. Однак кожен з цих автокодерів має спеціальні внутрішні структури: вони накладають зображення на себе через проміжне

представлення, яке є перекладом зображення в інший домен. Така установка також може розглядатися як особливий випадок "змагальних автокодерів", які використовують суперечливу втрату, щоб навчити вузький вузол автокодера відповідати довільному цільовому розподілу. У нашому випадку цільовий розподіл для $X \rightarrow X$ автокодер — це домен Y [29].

2.2.2 Алгоритм, заснований на моделі DNN (алгоритм Ґетіса)

Даний алгоритм було сформовано на основі VGG-мережі, конволюційної нейронної мережі, яка конкурує з працею людини за загальним завданням розпізнавання візуальних об'єктів. Використовувався простір функцій, який надають 16 згорткових та 5 об'єднаних шарів 19 шару VGGNetwork.

Ми не використовуємо жодного з повністю пов'язаних шарів. Модель є загальнодоступною і може бути досліджена. Заміна операції максимального об'єднання середнім об'єднанням покращує градієнтний потік, а один отримує незначно більш привабливі результати, саме тому зображені зображення створюються при середньому об'єднанні.

Зазвичай кожен шар у мережі визначає нелінійний банк фільтрів, складність якого збільшується із положенням шару в мережі. Отже, задане вхідне зображення $\vec{x}$ кодується у кожному шарі CNN реакціями фільтра на це зображення. У шарі з різними фільтрами N_l є карти N_l, що мають розмір кожної з розмірів M_l, де M_l — висота, менша за ширину карти функції.

Отже, відповіді в шарі l можуть зберігатися в матриці $F^l \in R^{N_l \times M_l}$, де F_{ij}^l — активація і-го фільтра в положенні j в шарі l. Для візуалізації інформації про зображення, кодованої в різних шарах ієрархії (рис. 1, реконструкції вмісту), ми виконуємо градієнтний спуск на зображенні білого шуму, щоб знайти інше зображення, яке відповідає характеристикам характеристик вихідного зображення. Тож нехай $\vec{p}$ і $\vec{x}$ є оригінальним зображенням і зображенням, яке генерується, а P_l і F_l їх відповідним

поданням ознак у шарі l. Потім ми визначаємо втрату в квадраті помилки між двома представленнями функцій

$$L_{content}(\vec{p},\vec{x},l)=\frac{1}{2}\sum_{i,j}\left(F_{ij}^{l}-P_{ij}^{l}\right)^{2} . \quad (5)$$

Похідна цієї втрати щодо активацій у шарі l дорівнює

$$\frac{\partial L_{content}}{\partial F_{ij}^{l}}=\begin{cases}\left(F^{l}-P^{l}\right)_{ij}, \text{if } F_{ij}^{l}>0\end{cases} \quad (6)$$

з якого може бути обчислений градієнт відносно зображення $\vec{x}$, використовуючи стандартне поширення помилок зворотного зв'язку. Таким чином, ми можемо змінити початково випадкове зображення $\vec{x}$, поки воно не створить ту ж відповідь у певному шарі CNN, що і вихідне зображення $\vec{p}$.

Зверху на відповіді CNN у кожному шарі мережі ми побудували подання стилю, яке обчислює кореляції між різними відповідями фільтрів, де очікування приймається за просторове розширення вхідного зображення. Ці кореляційні ознаки задаються матрицею Грама $G^{l}\in R^{N_l\times N_l}$, де G_{ij}^{l} — внутрішній добуток між векторизованою картою ознак i та j у шарі l:

$$G_{ij}^{l}=\sum_{k}F_{ik}^{l}F_{jk}^{l} \quad (7)$$

Для створення текстури, яка відповідає стилю заданого зображення, ми використовуємо градієнтний спуск із зображення білого шуму, щоб знайти інше зображення, яке відповідає стильовому зображенню вихідного зображення. Це робиться шляхом мінімізації середньої квадратичної відстані між записами матриці Грама від вихідного зображення та матрицею Грама зображення, яке потрібно генерувати. Тож нехай $\vec{a}$ і $\vec{x}$ є оригінальним зображенням і зображенням, яке генерується, і A_l і G_l їх відповідні стильові зображення в шарі l. Тоді внесок цього шару в загальну втрату

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2 \tag{8}$$

і тотальна втрата

$$L_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l \ , \tag{9}$$

де w_l — коефіцієнти зважування внеску кожного шару у загальну втрату. Похідна E_l щодо активацій у шарі l може бути обчислена аналітично:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left(\left(F^l\right)^T \left(G^l - A^l\right)\right)_{ji} & \text{if } F_{ij}^l < 0 \end{cases} \tag{10}$$

Градієнти E_l щодо активації в нижніх шарах мережі можна легко обчислити, використовуючи стандартне поширення помилок назад.

Для створення зображень, що змішують вміст фотографії зі стилем картини, ми спільно мінімізуємо відстань зображення білого шуму від вмісту зображення фотографії в одному шарі мережі та подання стилю живопис у ряді шарів CNN. Тож нехай $\vec{p}$ буде фотографією, а $\vec{a}$ — художнім твором. Функція втрат, яку ми мінімізуємо, є

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x}) \tag{11}$$

де і є коефіцієнтами вагомості згідно до змісту та реконструкції стилів [9].

2.2.3 Алгоритм, заснований на моделі CCNN

Модуль витягування компонентів обличчя підрозділяє оригінальне зображення обличчя, його відповідну карикатуру, намальовану художником, і середнє обличчя, на помітні компоненти, такі як око, ніс, підборіддя, рот тощо. Згодом геометричні дані з даного компонента оригінального

зображення та даних з відповідного компонента середнього зображення вводяться як входи в модуль нейронної мережі. Відповідні дані з карикатурного компонента вводяться в модуль як вихідні дані.

Після того, як у навчальному процесі було використано достатньо точок даних, ми показуємо, що нейронна мережа здатна передбачити карикатуру нового зображення, що зображує той самий компонент обличчя, який був використаний у процесі тренування. Нижче наведено більш детальний опис задіяних процесів.

Крок 1. Створення середнього обличчя: Для цілей нашого сучасного дослідження, яке зосереджено лише на доказі концепції, середнє обличчя (і, отже, компоненти обличчя) було намальовано для експериментального використання та аналізу. Однак у реальній системі можна використовувати одну з багатьох чудових програм генератора середніх облич, доступних у Всесвітній павутині.

Крок 2. Витяг / розділення компонентів для обличчя: Розроблений простий інструмент аналізу зображень, який базується на виявленні країв, встановленні порогів та витонченні, щоб виділити / відокремити різні важливі компоненти обличчя, такі як вуха, очі, ніс та рот, від оригінального, середнього та карикатурного обличчя. зображення. Існує багато таких алгоритмів та комерційних програмних пакетів, які можуть ідентифікувати компоненти обличчя за зображеннями / ескізами.

Крок 3. Створення наборів даних для навчання нейронної мережі: Після вилучення компонентів обличчя оригінальні, середні та карикатурні зображення розглянутого компонента перекриваються, припускаючи відповідну загальну центральну точку. Наприклад, для ока центр райдужки можна вважати найбільш підходящою центральною точкою. Згодом за допомогою ліній поперечного перерізу з центром у вищезазначеній точці та розміщених на рівних кутових відстанях зазначаються координатні точки, в яких лінії перетинають компоненти. Це робиться за напрямком годинникової стрілки.

Таблиця *1*

Набір даних

	Original	Mean	Caricature
X1	25	37	13
Y1	99	99	99
X2	47	53	37
Y2	108	102	118
X3	56.8	56.8	56.8
Y3	109	102	125
X4	66	59	76
Y4	109	102	119
X5	86	75	100
Y5	99	99	99
X6	62	60	68
Y6	93	95	87
X7	56.8	56.8	56.8
Y7	92	95	86
X8	50	52	45
Y8	93	95	87

Крок 4. Таблицювання наборів даних: Після отримання координатних точок X-Y, як на кроці 3, вони складаються в таблицю, як показано в Таблиці 1. Чим більша кількість використовуваних ліній поперечного перерізу, тим точніше буде захоплена форма. Однак для наочності подання та простоти експериментів ми використовували лише чотири лінії поперечного перерізу на малюнку 3, що призводить до восьми наборів даних.

Крок 5. Введення даних: Беручи до уваги той факт, що нейронну мережу слід навчити автоматично створювати карикатуру на певний компонент обличчя, намальований певним художником, ми вважаємо точки даних, отримані з карикатурного зображення вище, вихідним набором навчальних даних нейронної мережі. Крім того, нейронна мережа забезпечена наборами даних, отриманими з вихідних та середніх зображень,

для формулювання вхідних даних. Це слідує загальновизнаній стратегії, яку використовує людський мозок для аналізу даного зображення обличчя порівняно з відомим середнім образом обличчя.

Крок 6. Налаштування нейронної мережі: Ми пропонуємо використовувати такі параметри навчання для простого, швидкого та ефективного навчального процесу.

Крок 7. Тестування: Після успішного завершення тренінгу, як описано вище, відповідний компонент обличчя нового оригінального зображення відбирається та подається як вхідний сигнал до навченої нейронної мережі разом із відповідними даними відповідного середнього компонента. У розділі-4 ми наводимо експериментальні докази на підтвердження нашого доказу концепції, що CCNN здатний відобразити стиль малювання карикатуриста [20].

2.3. Висновки

Підсумовуючи наш стислий огляд методів, зазначимо, що найдоречнішою нам здаються скелетна анімація, як така, що може бути легко реалізована за допомогою готових 3D-спрайтів та 3D-сценаріїв анімації (наприклад, в форматі *.fbx), які легко можна знайти в Інтернеті. Також нам видається доречним метод Л. Ґетіса, як такий, який найлегше реалізовується.

РОЗДІЛ 3 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ СТВОРЕННЯ АНІМАЦІЇ З МОЖЛИВІСТЮ НАЛАШТУВАННЯ СТИЛІСТИКИ ТА ЗОВНІШНЬОГО ВИГЛЯДУ ДЕКОРАЦІЙ ТА ПЕРСОНАЖІВ

3.1. Архітектура системи

Програмна система створення анімації з можливістю налаштування стилістики та зовнішнього вигляду декорацій та персонажів складається з трьох основних модулів: основного застосунку, модуля створення відео з налаштованими декораціями та персонажами і модуля надання стилістики. Трохи забігаючи наперед, скажемо, що в цілому архітектура системи виглядає так:

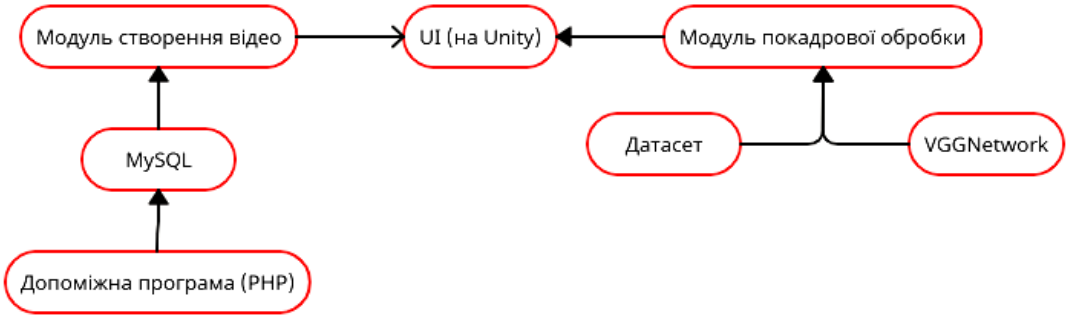

Рисунок *14. Архітектура системи*

Як бачимо, основний застосунок може бути написано мовою C# і найкраще реалізується в середовищі програмування Unity.

3.1.1 Архітектура UI на Unity

UI на Unity мало хто любить займатися, тому розробники часто вибирають вже існуючі архітектури і вперто захищають їх в палких суперечках. На ділі немає якоїсь ідеальної архітектури, придатною для всіх.

UI на Unity непоганий, але іноді може бути жахливим. Багато чого вже вбудовано в Unity, тому розробка інтерфейсу там досить проста, однак деякі сумнівні рішення і невеликі деталі можуть викликати фрустрацію.

Іноді доведеться створювати свої компоненти інтерфейсу. При цьому потрібно пам'ятати про доцільність цих дій — можливо, те, що вам потрібно, вже є на движку.

Глосарій.

- Екран (screen) — окрема замкнута частина інтерфейсу. Вони бувають двох видів: панелі (panels) і діалоги (dialogs).
- Одночасно можуть бути відкриті кілька панелей і діалогів, але тільки один діалог може бути інтерактивним в одиницю часу. Панелі можуть бути тільки відкриті або закриті, а діалоги мають історією.
- Віджет (widget) — частина екрану, яку можна використовувати кілька разів, значення в ньому можуть динамічно змінюватися, і можуть бути використані в інших елементах (наприклад, в діалозі з досягненнями).
- Шар (layer) — то, в чому знаходиться і чим контролюється екран.
- Діалог вибору рівня.
- Панель, що відображає кількість ресурсів користувача.
- Навігаційна панель.

Ієрархія. Ось приблизна будова ієрархії елементів інтерфейсу (в дужках перераховано, що в них зазвичай міститься).

UI [код UI Manager, основний полотно]

* Камера

* Шар діалогу (код шару діалогу)

— Діалог A (контролер діалогу A)

— Діалог B (контролер діалогу B)

* Шар панелей (код шару панелей)

— Панель A (контролер панелі A)

— Панель B (контролер панелі B)

Unity відмальовує елементи строго за ієрархією, тобто, нижні елементи рендеряться в останню чергу. У цьому прикладі панелі завжди будуть розташовані поверх діалогів.

Код системи інтерфейсу розділений на три частини: головний фасад, контролери шарів і контролери екранів [18].

Патерн MVC. Model-View-Controller (MVC, «Модель-Подання Контролер») — схема поділу даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер — таким чином, що модифікація кожного компонента може здійснюватися незалежно:

- Модель (Model) надає дані і реагує на команди контролера, змінюючи свій стан.
- Подання (View) відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі.
- Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

При цьому ще в описі оригінального MVC згадувалося, що відокремлення Контролера від Подання не надто важливе і вони цілком можуть бути інтегровані [30].

3.1.2 Архітектура 3D-додатків на Unity

Успадкування VS компоненти. У великих іграх досить складна архітектура. Складні суті і складну взаємодію між класами. Якщо намагатися розробляти гри, використовуючи стандартний ООП підхід, то гарантовані постійні переробки купа коду і сильне збільшення тривалості розробки.

Проблема криється в спадкуванні (проблема тендітних базових класів — ситуація, коли змінити реалізацію типу-предка неможливо, не порушивши коректність функціонування типів-нащадків). При знаходити власні шляхи розв'язання для боротьби з цією проблемою був сформований Компонентно-орієнтований підхід (КОП). Якщо коротко, суть КОП наступна:

«Є якийсь клас-контейнер, а також клас-компонент, який можна додати в клас-контейнер. Об'єкт складається з контейнера і компонентів в цьому контейнері».

Компоненти трохи нагадують інтерфейси. Але інтерфейси дозволяють тільки виділити у класів загальну сигнатуру функцій і властивостей, а компоненти дозволяють винести загальну реалізацію класів окремо.

В ООП підході об'єкт визначається описуваних його класом.

У КОП підході об'єкт визначається компонентами, з яких він складається. Не важливо, що це за об'єкт. Важливо, що у нього є і що він вміє робити.

КОП спрощує повторне використання написаного коду — використання одного компонента в різних об'єктах. Також з різних комбінацій вже існуючих компонентів, можна зібрати новий тип об'єкту.

Для прикладу, візьмемо об'єкт «персонаж». З точки зору ООП — це був би один великий клас, можливо, успадковані від чогось. З точки зору КОП — це набір компонентів, що складають об'єкт «персонаж». Наприклад: характеристики / статті персонажа — компонент «Stats», управління персонажем «CharacterController», анімація персонажа — «CharacterAnimationController», обробник зіткнень — «CharacterCollisionHandler».

Не варто відмовлятися від спадкування в іграх. Спадкування компонентів цілком нормальна практика. Та й в деяких ситуаціях, це буде більш правильним. Але, якщо видно, що буде кілька рівнів успадкування класів для опису об'єктів, то краще використовувати компоненти.

Складні ієрархії класів юнітів, предметів та іншого. Багато розробники, незнайомі або погано знайомі з КОП, роблять одну і ту ж помилку при проектуванні складних систем класів для юнітів, предметів. Або навіть правильно виділяють компоненти, але для чогось роблять спадкування AI компонентів.

Розглянемо докладніше ієрархію класів юнітів різних типів. Для скорочення тексту, юніти в даному контексті можуть бути і персонажами і будівлями.

Оскільки об'єкти збираються з компонентів, то вже немає проблеми, що при зміні одного об'єкта знадобиться змінювати інші. Також не треба більше ламати голову над створенням ієрархії класів, що підходить усім об'єктам.

Машини станів, дерева поводжень. Схема в попередній частині була дуже спрощеною. Насправді компонентів знадобиться набагато більше. Також знадобиться організовувати їх взаємодію один з одним. Для спрощення роботи з об'єктами зі складною поведінкою (юніти, персонажі) використовуються машини станів, дерева поводжень.

1. Машина станів. Логіка об'єкта розбивається на стану, події, переходи, а також може розбиватися ще й на дії. Варіації реалізації цих елементів можуть помітно відрізнятися.

Стан об'єкта — може бути як класом без ігрової логіки, просто зберігає якісь дані, наприклад, назва стану об'єкта: атака, переміщення. Або клас «стан» може описувати поведінку об'єкта в конкретному стані.

Дія — функція, яка може виконатися в даному стані.

Перехід — зв'язок між 2-ма станами. Вказує, з якого стану в яке можливий перехід.

Подія — якесь повідомлення / команда, що передається в машину станів або викликається всередині неї. Служить для вказівки, що треба виконати перехід в інший стан, якщо це можливо з поточного стану.

Цікаво реалізована машина станів в плагіні Behaviour Machine. Там станом є MonoBehaviour компонент, який відповідає за логіку роботи в цьому стані. Причому, станом також може бути дерево поведінки.

2. Ієрархічна машина станів. Коли станів багато, збільшується кількість зв'язків між ними, що ускладнює роботу з графом машини станів. Для спрощення роботи з ним, можна використовувати ієрархічну машину станів. Вона відрізняється тим, що в якості стану можна використовувати

вкладену машину станів. Таким чином виходить деревоподібна ієрархія станів.

3. Дерево поведінки. Для спрощення написання AI, в іграх використовуються дерева поводжень (Behavior Tree).

Дерево поведінки — це деревоподібна структура, як вузли якої виступають невеликі блоки ігровий логіки. З різних блоків логіки розробник конструює в візуальному редакторі деревоподібну структуру, налаштовує вузли дерева. Ця структура буде відповідати за прийняття рішень персонажем і його взаємодію з ігровим світом.

Кожен вузол повертає результат, від якого залежить, як будуть оброблятися інші вузли дерева. Варіанти повертається результату зазвичай такі: успіх, невдача, виконується.

Основні типи вузлів в дереві поведінки:

- Action Node (дія)

Просто деяка функція, яка повинна виконатися при відвідуванні даного вузла.

- Condition (умова)

Зазвичай служить для того, щоб визначити, виконувати чи ні наступні за ним вузли. При true поверне Success, а при false повертає Fail.

- Sequencer (послідовність)

Виконує всі вкладені вузли по порядку, поки будь-якої з них не завершиться невдачею (в такому випадку повертає Fail), або поки всі вони успішно не завершаться (тоді повертає Success).

- Selector (селектор)

На відміну від Sequencer, припиняє обробку, як тільки будь-який вкладений вузол поверне Success.

- Iterator (ітератор — виконує роль циклу for)

Використовується для виконання в циклі серії дій деяке число раз.

- Parallel Node

Виконує всі свої дочірні вузли «одночасно». Тут не мається на увазі, що вузли виконуються декількома потоками. Просто створюється ілюзія паралельного виконання, аналогічно корутінам в Unity3d.

Коли краще використовувати машину станів, а коли дерево поведінки?

У книзі «Artificial Intelligence for Games II» на 370-ій сторінці йдеться, що дерева поведінки складніше реалізувати, якщо їм потрібно реагувати на події ззовні. Також там пропонується можливе рішення — ввести поняття «завдання» (наприклад: патрулювання, переслідування противника, атака противника) в дерево поведінки. Тобто передбачається, що контролер дерева поведінки буде переходити на інший вузол-завдання, щоб змінити поведінку. Також в книзі пропонується альтернативний варіант — об'єднати дерево поведінки з машиною станів. До речі, в плагін Behaviour Machine це вже реалізовано.

Додамо, що якщо AI сам отримує дані зі світу і не отримує жодних команд, то Behavior Tree підходить для нього. Якщо ж AI чимось управляється — людиною або іншим AI (наприклад, юніт в стратегіях управляється гравцем-комп'ютером або загоном), то краще використовувати стейт-машину.

Абстракції ігрових об'єктів. Не варто розглядати, що керований гравцем персонаж — це об'єкт Player. Також не варто вважати, що цим персонажем може управляти тільки людина або тільки комп'ютер. Хто знає, як в майбутньому буде перероблений геймплей.

Player (може бути як комп'ютер, так і людина, в одному класі змішувати їх не варто) — це окремий об'єкт. Персонаж / юніт — також окремий об'єкт, яким може керувати будь-який гравець. В стратегічних іграх до того ж можна окремо винести об'єкт «загін».

Розділити ігрову логіку на подібні об'єкти не складно. До того ж вона буде застосовна до безлічі різних ігор.

Спрощення доступу до інших компонентів в об'єкті, сцені. При великій кількості компонентів в об'єкті, з'являється незручність при потребі звернення до них. Постійно доводиться заводити в кожному компоненті поля для зберігання посилань на інші компоненти або звертатися до них через GetComponent().

Патерн медіатор наштовхнув мене ввести якийсь компонент-посередник, через який компоненти могли б звертатися один до одного. До того ж це дозволить винести перевірку існування інших компонентів в даний компонент і код знадобитися писати тільки 1 раз. Такий компонент у різних типів об'єктів теж варто робити різним, тому що використовуються різні набори компонентів. В даному випадку — це не реалізація патерну медіатор, а просто кешування посилань в одному класі для зручності доступу до інших компонентів об'єкта.

У сценах аналогічна ситуація. Можна в певному об'єкті Сінглтон зробити посилання на часто використовувані компоненти, щоб в інспектора конкретних компонентів не доводилося постійно вказувати посилання на інші об'єкти.

Оскільки компоненти потрібно вказати тільки в одному об'єкті, а не в декількох, трохи зменшується обсяг роботи в редакторі, та й коду в цілому буде менше.

Складні складові ігрові об'єкти. Персонажі, UI елементи і деякі інші об'єкти можуть складатися з більшого числа компонентів-скриптів і безлічі вкладених об'єктів. Якщо погано продумана ієрархія подібних об'єктів, то це може сильно ускладнити розробку.

Розглянемо 2 випадки, коли важливо продумувати ієрархію об'єкта:

- окремі частини об'єкта повинні замінюватися іншими у процесі гри;
- частина скриптів об'єкта повинна працювати тільки в одній сцені, а інша частина — в інший.

Для початку розглянемо перший випадок.

Можна повністю замінювати об'єкт, але якщо він після заміни повинен знаходитися в тому ж стані і мати ті ж дані, що і до заміни, то завдання ускладнюється.

Для спрощення заміни будь-якої частини об'єкта, його структуру можна організувати, наприклад, так:

- Character.
- Data.
- ControlLogic (скрипти для управління персонажем).
- RootBone (коренева кістка персонажа; компоненти Animator і скрипти для роботи з IK повинні бути тут, інакше вони не будуть працювати).
- Animation (інші скрипти для роботи з анімацією).
- Model.

Подібне розбиття дозволить змінити зовнішній вигляд об'єкта або контролер анімації, не сильно зачіпаючи інші компоненти.

Тепер розглянемо другий випадок.

Наприклад, є якийсь об'єкт зі спрайтом і даними. При кліці на нього в сцені поліпшень, потрібно поліпшити даний об'єкт. А при кліці на нього в сцені гри повинно виконатися якесь ігрове дію.

Можна зробити 2 префаб, але тоді, якщо об'єктів багато, доведеться налаштовувати в 2 рази більше префабов.

Можна піти іншим шляхом і організувати структуру в такий спосіб:

- ObjectView (картинка об'єкта).
- Data (дані об'єкта, які використовуються в обох сценах).
- UpgradeLogic (кнопка і скрипти для сцени поліпшень).
- GameLogic (кнопка і скрипти для сцени гри).

Поле targetGraphic в кнопках має посилатися на картинку в ObjectView.

Даний підхід перевірявся в uGUI [33].

3.2 Засоби реалізації

3.2.1 Середовище розробки Visual Studio 2019

Microsoft Visual Studio — це інтегроване середовище розробки (IDE) від Microsoft. Застосовується для розробки комп'ютерних програм, а також веб-сайтів, веб-програм, веб-сервісів та мобільних додатків. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store та Microsoft Silverlight. Він може створювати як власний код, так і керований код.

Visual Studio включає редактор коду, що підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Інтегрований налагоджувач працює як налагоджувач вихідного рівня, так і налагоджувач машинного рівня. Інші вбудовані інструменти включають програміст коду, дизайнер для побудови графічних програм, веб-дизайнер, дизайнер класів та конструктор схем баз даних. Він приймає плагіни, які розширюють функціональність майже на кожному рівні, включаючи додавання підтримки для систем керування джерелами (таких як Subversion та Git) та додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для мов, що належать до домену, або наборів інструментів для інших аспектів розробки програмного забезпечення життєвий цикл (як клієнт Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування та дозволяє редактору коду та налагоджувачу підтримувати (різною мірою) майже будь-яку мову програмування, за умови, що існує спеціальна послуга. Вбудовані мови включають C, C++, C++ / CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML та CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M, серед інших, доступна через плагіни. Раніше підтримувались Java (і J#).

Найпростіший випуск Visual Studio, випуск Community, доступний безкоштовно. Гасло для видання Visual Studio Community — "Безкоштовна

повнофункціональна IDE для студентів, відкритих програм та індивідуальних розробників".

Остання версія Visual Studio — 2019 рік [32].

3.2.2 Мова програмування C#

C# (вимовляється Сі-шарп) — мультипарадигмальна мова програмування загального призначення, що охоплює статичну типізацію, сильну типізацію, лексичну область дії, імперативну, декларативну, функціональну, загальну, об’єктно-орієнтовану (на основі класів) та компонентно-орієнтовану парадигми програмування.

На сьогодні мова програмування C# — одна з найпотужніших та найзатребуваніших мов в IT-галузі. На даний момент на ньому пишуться найрізноманітніші програми: від невеликих десктопних програмок до великих веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів.

C# вже не молода мова, і, як і вся платформа .NET, вже пройшла великий шлях. Перша версія мови вийшла разом з релізом Microsoft Visual Studio .NET в лютому 2002 року. Поточною версією мови є C# 9.0, яка вийшла 10 листопада 2020 року разом з релізом .NET 5.

C# є мовою з Сі-подібним синтаксисом і близька в цьому відношенні до C++ і Java. Тому, якщо ви знайомі з однією з цих мов, то опанувати C # буде легше.

C# є об'єктно-орієнтованою і в цьому плані багато перейняла у Java і C++. Наприклад, C# підтримує поліморфізм, успадкування, перевантаження операторів, статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішити завдання з побудови великих, але в той же час гнучких, масштабованих і розширюваних додатків. І C# продовжує активно розвиватися, і з кожною новою версією з'являється все більше цікавих функциональностей, як, наприклад, лямбда, динамічне зв'язування, асинхронні методи і т.д. [32].

3.2.3 Середовище розробки ігор Unity

Unity — багатоплатформовий інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X, крос-платформний ігровий движок, розроблений Unity Technologies, вперше анонсований і випущений в червні 2005 року на всесвітній конференції розробників Apple Inc. як ексклюзивний ігровий движок для Mac OS X. Станом на 2018 рік движок був розширений для підтримки більш ніж 25 платформ. Механізм може бути використаний для створення тривимірних, двовимірних, ігор у віртуальну реальність та доповнену реальність, а також моделювання та іншого досвіду. Движок був прийнятий на виробництво за межами відеоігор, таких як кіно, автомобілебудування, архітектура, машинобудування та будівництво.

З моменту запуску було випущено кілька основних версій Unity. Остання стабільна версія, 2020.1.16, вийшла в грудні 2020 року.Створені за допомогою Unity застосунки працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360. Є можливість створювати інтернет-додатки за допомогою спеціального під'єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL.

Ігрова логіка пишеться за допомогою C#, раніше також була можливість використовувати Boo та JavaScript, але розробники відмовились від їх підтримки.

Технічні характеристики Unity:

- Ігровий рушій повністю пов'язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі;
- Робота з ресурсами можлива через звичайний Drag&Drop.
- Підтримка імпортування великої кількості форматів файлів;
- Вбудований генератор ландшафтів;

- Вбудована підтримка мережі;
- Інтерфейс редактора дуже гнучкий, є можливість писати свої вікна редактора та різноманітні розширення для нього;
- Існує рішення для спільної розробки — Asset Server. Також можна використовувати зручний для користувача спосіб контролю версій. Наприклад, SVN або Source Gear.

3.2.4 Microsoft Cognitive Toolkit

Microsoft Cognitive Toolkit (CNTK) — це інструментарій з відкритим кодом для розподіленого глибокого навчання комерційного рівня. Він описує нейронні мережі як ряд обчислювальних кроків за допомогою спрямованого графіка. CNTK дозволяє користувачеві легко реалізовувати та комбінувати популярні типи моделей, такі як DNN із зворотним зв'язком, згорткові нейронні мережі (CNN) та періодичні нейронні мережі (RNN / LSTM). CNTK реалізує навчання стохастичного градієнта (SGD, зворотне розповсюдження помилок) з автоматичною диференціацією та розпаралеленням на декількох графічних процесорах та серверах.

Це відео забезпечує огляд набору інструментів на високому рівні. Крім того, Microsoft пропонує вступний курс глибокого навчання за допомогою CNTK, Deep Learning Explained.

Останній випуск CNTK — 2.7.

CNTK може бути включений як бібліотека до ваших програм Python, C # або C ++ або використаний як самостійний інструмент машинного навчання за допомогою власної мови опису моделі (BrainScript). Крім того, ви можете використовувати функціонал оцінки моделі CNTK з ваших програм Java.

CNTK підтримує 64-розрядні операційні системи Linux або 64-розрядні Windows. Для встановлення ви можете вибрати попередньо скомпільовані двійкові пакети або скомпілювати набір інструментів із джерела, наданого в GitHub.

CNTK також є одним з перших наборів інструментів глибокого навчання, що підтримують формат ONNX Open Neural Network Exchange, представлену модель спільного використання з відкритим кодом для взаємодії платформи та спільної оптимізації. Спільно розроблений Microsoft та підтриманий багатьма іншими, ONNX дозволяє розробникам переміщати моделі між фреймворками, такими як CNTK, Caffe2, MXNet та PyTorch.

Останній випуск CNTK підтримує ONNX v1.0 [7].

Наведемо лістинг класу безпосередньої обробки зображення з метою корекції стилю, використовуваного в поточній роботі.

Лістинг 1 Клас безпосередньої обробки зображення

```
public static class StyleTransfer
    {
        // color channel offsets for the VGG19 network
        public static readonly float[] VGG19_Offsets = new
float[] { 103.939f, 116.779f, 123.68f };

        /// <summary>
        /// Convert an image to a 1-dimensional float
array. All color frames
        /// are laid out sequentially, one after the other.
        /// </summary>
        /// <param name="mat">The input image as an OpenCv
Mat object.</param>
        /// <param name="offsets">Offsets to apply to each
color channel.</param>
        /// <returns>An 1-dimensional float array
containing the image data.</returns>
        public static float[] FlattenByChannel(Mat mat,
float[] offsets)
        {
            var num_pixels = mat.Size().Height *
mat.Size().Width;
            float[] result = new float[num_pixels * 3];
            //using (MatOfByte3 mat3 = new MatOfByte3(mat))
            using (Mat<Vec3b> mat3 = new Mat<Vec3b>(mat))
            {
                var indexer = mat3.GetIndexer();
                var pos = 0;
                for (int y = 0; y < mat.Height; y++)
                {
                    for (int x = 0; x < mat.Width; x++)
                    {
                        var color = indexer[y, x];
```

```
                        result[pos] = color.Item0 —
offsets[0];
                        result[pos + num_pixels] =
color.Item1 — offsets[1];
                        result[pos + 2 * num_pixels] =
color.Item2 — offsets[2];
                        pos++;
                    }
                }
            }
            return result;
        }

        /// <summary>
        /// Convert an 1-dimensional float array to an
image.
        /// </summary>
        /// <param name="offsets,g">The float array to
process.</param>
        /// <param name="offsets">Offsets to apply to each
color channel.</param>
        /// <param name="scaling">The scaling factor to
apply.</param>
        /// <param name="invertOrder">Set to invert the
order of the channels.</param>
        /// <returns>An image converted from the 1-
dimensional float array .</returns>
        public static byte[] UnflattenByChannel(float[]
img, float[] offsets = null, float scaling = 1.0f, bool
invertOrder = false)
        {
            if (offsets == null) { offsets = new float[3];
}
            var img_data = new byte[img.Length];
            var image_size = img.Length / 3;
            for (int i = 0; i < img_data.Length; i += 3)
            {
                img_data[i + 1] = (byte)Math.Max(0,
Math.Min(scaling * img[i / 3 + image_size] + offsets[1],
255));
                if (invertOrder)
                {
                    img_data[i + 2] = (byte)Math.Max(0,
Math.Min(scaling * img[i / 3] + offsets[0], 255));
                    img_data[i] = (byte)Math.Max(0,
Math.Min(scaling * img[i / 3 + 2 * image_size] +
offsets[2], 255));
                }
                else
```

```
                {
                    img_data[i] = (byte)Math.Max(0,
Math.Min(scaling * img[i / 3] + offsets[0], 255));
                    img_data[i + 2] = (byte)Math.Max(0,
Math.Min(scaling * img[i / 3 + 2 * image_size] +
offsets[2], 255));
                }
            }
            return img_data;
        }

        /// <summary>
        /// Load an image from disk and return it as a 1-
dimensional float array
        /// with all color channels laid out sequentially.
        /// </summary>
        /// <param name="imagePath">The path of the file to
load.</param>
        /// <param name="width">the width of the
file.</param>
        /// <param name="height">The height of the
file.</param>
        /// <returns></returns>
        public static float[] LoadImage(
            string imagePath,
            int width,
            int height)
        {
            using (var mat = Cv2.ImRead(imagePath))
            {
                using (var mat2 = new Mat(height, width,
mat.Type()))
                {
                    Cv2.Resize(mat, mat2, new Size(width,
height));
                    return FlattenByChannel(mat2,
VGG19_Offsets);
                }
            }
        }

        /// <summary>
        /// Get a list of content and style layers from the
given neural network.
        /// </summary>
        /// <param name="input">The neural network to
process.</param>
        /// <returns>A list of content and style layers in
the neural network.</returns>
```

```
        public static List<CNTK.Variable>
GetContentAndStyleLayers(
            this CNTK.Function input)
        {
            var layers = new List<CNTK.Variable>();
            var layerNames = new string[] { "conv5_2",
"conv1_1", "conv2_1", "conv3_1", "conv4_1", "conv5_1" };
            foreach (var layerName in layerNames)
            {
                var layer = input.FindByName(layerName);
                layers.Add(layer);
            }
            return layers;
        }

        /// <summary>
        /// Create a style transfer base model from a VGG19
network.
        /// </summary>
        /// <param name="feature">The input feature to
use.</param>
        /// <returns>A neural network containing the
content and style layers of the VGG19 network.</returns>
        public static CNTK.Function StyleTransferBase(
            this CNTK.Variable input)
        {
            // extract all content and style layers
            var layers =
((CNTK.Function)input).GetContentAndStyleLayers();

            // return a model from these layers
            return CNTK.Function.Combine(layers,
"content_and_styles").Clone(CNTK.ParameterCloningMethod.Fre
eze);
        }

        /// <summary>
        /// Adds a dream layer to a neural network.
        /// </summary>
        /// <param name="input">The neural network to
extend.</param>
        /// <param name="image">The content image.</param>
        /// <param name="width">The width of the content
image.</param>
        /// <param name="height">The height of the content
image.</param>
        /// <returns>The neural network extended with a
dream layer.</returns>
        public static CNTK.Function DreamLayer(
```

```
            this CNTK.Function input,
            float[] image,
            int width,
            int height)
        {
            // set up the dream layer
            var dream_weights_init = new
CNTK.NDArrayView(new int[] { width, height, 3 }, image,
NetUtil.CurrentDevice);
            var dream_weights = new
CNTK.Parameter(dream_weights_init, "the_dream");
            var dummy_features =
CNTK.Variable.InputVariable(new int[] { 1 },
CNTK.DataType.Float, "dummy_features");
            var dream_layer =
CNTK.CNTKLib.ElementTimes(dream_weights, dummy_features,
"the_dream_layer");

            // combine the dream layer with the content and
style layers
            var replacements = new
Dictionary<CNTK.Variable, CNTK.Variable>() { {
input.Arguments[0], dream_layer.Output } };
            var model =
input.Clone(CNTK.ParameterCloningMethod.Freeze,
replacements);

            // return the finished model
            var all_outputs = new List<CNTK.Variable>() {
dream_layer };
            all_outputs.AddRange(model.Outputs);
            return CNTK.Function.Combine(all_outputs, name:
"overall_model");
        }

        /// <summary>
        /// Calculate the output labels for style transfer.
        /// </summary>
        /// <param name="model">The neural network to
use.</param>
        /// <param name="contentImage">The content image to
use.</param>
        /// <param name="styleImage">The style image to
use.</param>
        /// <returns></returns>
        public static float[][]
CalculateLabels(CNTK.Function model, float[] contentImage,
float[] styleImage)
        {
```

```
            // make sure the content image dimensions match
the neural network input size
            // make sure the content and style images are
the same size
            var input_shape =
model.Arguments[0].Shape.Dimensions.ToArray();
            System.Diagnostics.Debug.Assert(input_shape[0]
* input_shape[1] * input_shape[2] == contentImage.Length);

System.Diagnostics.Debug.Assert(contentImage.Length ==
styleImage.Length);

            // set up a batch with the content and the
style image
            var batch_buffer = new float[2 *
contentImage.Length];
            Array.Copy(contentImage, 0, batch_buffer, 0,
contentImage.Length);
            Array.Copy(styleImage, 0, batch_buffer,
contentImage.Length, contentImage.Length);
            var batch_nd = new CNTK.NDArrayView(new int[] {
model.Arguments[0].Shape[0], model.Arguments[0].Shape[1],
model.Arguments[0].Shape[2], 1, 2 }, batch_buffer,
NetUtil.CurrentDevice);
            var batch = new CNTK.Value(batch_nd);

            // let the model evaluate the batch
            var inputs = new Dictionary<CNTK.Variable,
CNTK.Value>() { { model.Arguments[0], batch } };
            var outputs = new Dictionary<CNTK.Variable,
CNTK.Value>();
            foreach (var output in model.Outputs)
            {
                outputs.Add(output, null);
            }
            model.Evaluate(inputs, outputs,
NetUtil.CurrentDevice);

            // collect and return the model outputs
            float[][] labels = new
float[model.Outputs.Count][];
            labels[0] =
outputs[model.Outputs[0]].GetDenseData<float>(model.Outputs
[0])[0].ToArray();
            for (int i = 1; i < labels.Length; i++)
            {
                labels[i] =
outputs[model.Outputs[i]].GetDenseData<float>(model.Outputs
[i])[1].ToArray();
```

```
            }
            return labels;
        }

        /// <summary>
        /// Create a loss function that can compare the
model output with the style image.
        /// </summary>
        /// <param name="model">The model to use.</param>
        /// <param name="outputs">The model
outputs.</param>
        /// <param name="labels">The labels to use.</param>
        /// <returns>The loss function to use for
training.</returns>
        public static CNTK.Function CreateLossFunction(
            CNTK.Function model,
            IList<CNTK.Variable> outputs,
            IList<CNTK.Variable> labels)
        {
            var lossFunction =
ContentLossFunction(outputs[0], labels[0]);
            for (int i = 1; i < outputs.Count; i++)
            {
                var sl = StyleLossFunction(outputs[i],
labels[i]);
                lossFunction =
CNTK.CNTKLib.Plus(lossFunction, sl);
            }
            return lossFunction;
        }

        /// <summary>
        /// Create a batch for training.
        /// </summary>
        /// <param name="model">The model to use.</param>
        /// <param name="labels">The label data to
use.</param>
        /// <returns>A batch for training.</returns>
        public static Dictionary<CNTK.Variable, CNTK.Value>
CreateBatch(CNTK.Function model, float[][] labels)
        {
            var dictionary = new Dictionary<CNTK.Variable,
CNTK.Value>();
            for (int i = 0; i < model.Arguments.Count; i++)
            {
                var loss_input_variable =
model.Arguments[i];
                if (loss_input_variable.Name ==
"dummy_features")
```

```
                {
                    var dummy_scalar_buffer = new float[] {
1 };
                    var dummy_scalar_nd = new
CNTK.NDArrayView(new int[] { 1 }, dummy_scalar_buffer,
NetUtil.CurrentDevice, readOnly: true);
                    dictionary[loss_input_variable] = new
CNTK.Value(dummy_scalar_nd);
                }
                else
                {
                    var cs_index =
Int32.Parse(loss_input_variable.Name.Substring("content_and
_style_".Length));
                    var nd = new
CNTK.NDArrayView(loss_input_variable.Shape,
labels[cs_index], NetUtil.CurrentDevice, readOnly: true);
                    dictionary[loss_input_variable] = new
CNTK.Value(nd);
                }
            }
            return dictionary;
        }

        /// <summary>
        /// Infer the image from the trained model.
        /// </summary>
        /// <param name="model">The model to use.</param>
        /// <param name="batch">The evaluation batch to
use.</param>
        /// <returns>The image inferred from the
model.</returns>
        public static byte[] InferImage(
            this CNTK.Function model,
            Dictionary<CNTK.Variable, CNTK.Value> batch)
        {
            var outputs = new Dictionary<CNTK.Variable,
CNTK.Value>() { { model.Outputs[0], null } };
            model.Evaluate(batch, outputs,
NetUtil.CurrentDevice);
            var img =
outputs[model.Outputs[0]].GetDenseData<float>(model.Outputs
[0])[0].ToArray();
            return UnflattenByChannel(img, VGG19_Offsets);
        }

        /// <summary>
        /// A content loss function for style transfer.
        /// </summary>
```

```
        /// <param name="x">The current content image
tensor.</param>
        /// <param name="y">The desired content image
tensor.</param>
        /// <returns>A function that can calculate the
content loss.</returns>
        private static CNTK.Function
ContentLossFunction(CNTK.Variable x, CNTK.Function y)
        {
            var diff_ = CNTK.CNTKLib.Minus(x, y, name:
"content_loss_diff_");
            var square_ = CNTK.CNTKLib.Square(diff_, name:
"content_loss_square_");
            var sum_ = CNTK.CNTKLib.ReduceSum(square_,
CNTK.Axis.AllStaticAxes(), name: "content_loss_sum_");
            var scaling = CNTK.Constant.Scalar((float)(1.0
/ x.Shape.TotalSize), NetUtil.CurrentDevice);
            sum_ = CNTK.CNTKLib.ElementTimes(sum_, scaling,
name: "content_loss_");
            return sum_;
        }

        /// <summary>
        /// A style loss function for style transfer.
        /// </summary>
        /// <param name="style">The current style image
tensor.</param>
        /// <param name="combination">The
combination.</param>
        /// <returns>A function that can calculate the
style loss.</returns>
        private static CNTK.Function
StyleLossFunction(CNTK.Variable style, CNTK.Variable
combination)
        {
            var style_gram = GramMatrix(style);
            var combination_gram = GramMatrix(combination);
            var diff_ = CNTK.CNTKLib.Minus(style_gram,
combination_gram, name: "style_loss_diff_");
            var square_ = CNTK.CNTKLib.Square(diff_, name:
"style_loss_square_");
            var sum_ = CNTK.CNTKLib.ReduceSum(square_,
CNTK.Axis.AllStaticAxes(), name: "style_loss_reduce_sum_");
            var max_ = CNTK.CNTKLib.ReduceMax(style_gram,
CNTK.Axis.AllStaticAxes(), name: "style_loss_reduce_max");
            var style_gram_total_size =
style_gram.Output.Shape.Dimensions[0] *
style_gram.Output.Shape.Dimensions[1];
```

```
            var scaling_factor =
CNTK.Constant.Scalar((float)style_gram_total_size,
NetUtil.CurrentDevice);
            var result = CNTK.CNTKLib.ElementDivide(sum_,
scaling_factor, name: "style_loss_result_");
            result = CNTK.CNTKLib.ElementDivide(result,
max_, name: "style_loss_");
            return result;
        }

        /// <summary>
        /// Return the gramm matrix.
        /// </summary>
        /// <param name="x">The input.</param>
        /// <returns>The gram matrix of the
input.</returns>
        private static CNTK.Function
GramMatrix(CNTK.Variable x)
        {
            var x_shape = x.Shape.Dimensions.ToArray();
            var features = CNTK.CNTKLib.Reshape(x, new
int[] { x_shape[0] * x_shape[1], x_shape[2] }, name:
"gram_matrix_reshape_");
            var gram =
CNTK.CNTKLib.TransposeTimes(features, features, name:
"gram_matrix_transpose_times_");
            return gram;
        }

    }
```

3.2.5 Класифікатор зображень VGG19

VGG19 — це варіант моделі VGG, яка коротше складається з 19 шарів (16 згорткових шарів, 3 повністю зв’язаних шару, 5 шарів MaxPool та 1 шар SoftMax). Є й інші варіанти VGG, такі як VGG11, VGG16 та інші. VGG19 має 19,6 мільярда FLOP.

Простою мовою VGG — це глибокий CNN, який використовується для класифікації зображень. Шари в моделі VGG19 такі:

- Conv3x3 (64).
- Conv3x3 (64).
- MaxPool.

- Conv3x3 (128).
- Conv3x3 (128).
- MaxPool.
- Conv3x3 (256).
- Conv3x3 (256).
- Conv3x3 (256).
- Conv3x3 (256).
- MaxPool.
- Conv3x3 (512).
- Conv3x3 (512).
- Conv3x3 (512).
- Conv3x3 (512).
- MaxPool.
- Conv3x3 (512).
- Conv3x3 (512).
- Conv3x3 (512).
- Conv3x3 (512).
- MaxPool.
- Повністю підключений (4096).
- Повністю підключений (4096).
- Повністю підключений (1000).
- SoftMax.

Архітектура. Фіксований розмір (224 * 224) зображення RGB був наданий як вхід для цієї мережі, що означає, що матриця мала форму (224 224,3).

Єдиною попередньою обробкою, яку було зроблено, було те, що вони віднімали середнє значення RGB з кожного пікселя, обчислене протягом усього навчального набору.

Використовувані ядра розміром (3 * 3) із розміром кроку 1 піксель, це дозволило їм охопити цілі поняття зображення.

Просторове заповнення використовувалось для збереження просторової роздільної здатності зображення.

Максимальне об'єднання було виконано у вікнах 2 * 2 пікселі з sride 2.

За цим йшла випрямлена лінійна одиниця (ReLu), щоб ввести нелінійність, щоб зробити модель класифікацією кращою, та покращити обчислювальний час, оскільки попередні моделі використовували функції tanh або sigmoid, це виявилося набагато кращим за ті.

Реалізовано три повністю з'єднаних шари, з яких перші два мали розмір 4096, а потім шар з 1000 каналами для 1000- смугової класифікації ILSVRC, а кінцевий шар є функцією softmax.

Стовпець E у наступній таблиці стосується VGG19 (інші стовпці — для інших варіантів моделей VGG):

Таблиця *2*

Фактична конфігурація мереж

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Порівняння та пояснення стовпців (Архітектура) у таблиці 2. Спочатку пояснимо стовпець E, оскільки це архітектура VGG19. Він містив 16 шарів CNN та три повністю зв'язаних шари та кінцевий шар для функції softmax, повністю пов'язані шари та кінцевий шар залишаться однаковими для всіх архітектури мережі.

B : Містить 8 шарів CNN, отже, загалом 11 шарів, включаючи повністю зв’язані (FC) шари, і не має внутрішньої різниці, крім кількості шарів.

A-LRN : Це також схоже на колонку A, але має один додатковий крок локальної нормалізації відповіді (LRN), який реалізує бічне гальмування в

шарі, під яким я маю на увазі, що він робить значний пік і, таким чином, створює локальний максимум, який збільшує сенсорне сприйняття, яке нам може знадобитися в нашому CNN, але було видно, що для цього конкретного випадку, тобто ILSVRC, це не підвищувало точності, і загальна мережа займала більше часу для навчання.

C : Він містить 13 шарів CNN та 16, включаючи шари FC, У цій архітектурі автори використовували фільтр conv (1 * 1) лише для того, щоб ввести нелінійність і, таким чином, кращу дискримінацію.

B і D : Ці стовпці просто додають додаткові шари CNN і складаються з 13 шарів та 16 шарів відповідно.

За цим ми можемо бачити, що глибокі CNN можуть досягти справді кращої точності, впроваджуючи більше шарів, і можуть зменшити обчислення, застосовуючи шари 3 * 3 конв. Фільтра, а не просто 7 * 7 фільтр, який з часом зменшив кількість параметрів і, отже, обчислювальний час .

Використання нейронної мережі VGG. Основною метою, для якої була розроблена мережа VGG, було виграти ILSVRC, але вона використовувалася багатьма іншими способами.

Використовувані як хороша архітектура класифікації для багатьох інших наборів даних, і оскільки автори робили доступними моделі для широкого загалу, вони можуть використовуватися як є, так і з модифікацією для інших подібних завдань.

Трансферне навчання: також може бути використане для розпізнавання обличчя.

Ваги легко доступні з іншими фреймворками, такими як Keras, щоб їх можна було повозити та використовувати як завгодно [13].

3.2.6 Система управління базами даних MySQL

MySQL — вільна реляційна система управління базами даних (СУБД). На сьогодні це одна з найпопулярніших систем управління базами даних.

Початковим розробником даної СУБД була шведська компанія MySQL AB. У 1995 році вона випустила перший реліз MySQL. У 2008 році компанія MySQL AB була куплена компанією Sun Microsystems, а в 2010 році вже компанія Oracle поглинула Sun і тим самим придбала права на торговельну марку MySQL. Тому MySQL на сеголняшней день розвивається під егідою Oracle.

Поточної актуальною версією СУДБ є версія 8.0, яка вийшла в січні 2018 року.

Спільнотою розробників MySQL створені різні відгалуження коду, такі як Drizzle, OurDelta, Percona Server і MariaDB. Всі ці відгалуження вже існували на момент поглинання компанії Sun корпорацією Oracle. MySQL є кросплатформеною, є дистрибутиви під найрізноманітніші ОС, в тому числі найбільш популярні версії Linux, Windows, MacOS [34].

3.2.7 Середовище розробки MySQL Workbench

MySQL Workbench — це уніфікований візуальний інструмент для архітекторів баз даних, розробників та адміністраторів баз даних. MySQL Workbench забезпечує моделювання даних, розробку SQL та комплексні інструменти адміністрування для конфігурації сервера, адміністрування користувачів, резервного копіювання та багато іншого. MySQL Workbench доступний у Windows, Linux та Mac OS X.

MySQL Workbench пропонує візуальні інструменти для створення, виконання та оптимізації запитів SQL. Редактор SQL забезпечує виділення синтаксису кольорів, автозаповнення, повторне використання фрагментів SQL та історію виконання SQL. Панель підключень до бази даних дозволяє розробникам легко керувати стандартними підключеннями до бази даних, включаючи MySQL Fabric. Браузер об’єктів забезпечує миттєвий доступ до схеми бази даних та об’єктів.

MySQL Workbench забезпечує візуальну консоль для легкого адміністрування середовищ MySQL та отримання кращої видимості баз

даних. Розробники та адміністратори баз даних можуть використовувати візуальні інструменти для налаштування серверів, адміністрування користувачів, виконання резервного копіювання та відновлення, перевірки даних аудиту та перегляду стану бази даних.

MySQL Workbench надає набір інструментів для підвищення продуктивності програм MySQL. Бази даних адміністраторів можуть швидко переглядати ключові показники ефективності за допомогою панелі інструментів. Звіти про ефективність забезпечують легку ідентифікацію та доступ до гарячих точок введення-виведення, дорогі оператори SQL тощо. Крім того, за допомогою одного кліка розробники можуть побачити, де оптимізувати свій запит, завдяки вдосконаленому та простому у використанні плану візуального пояснення [17].

3.2.8 Мова програмування PHP

PHP (англ. PHP: Hypertext Preprocessor — «PHP: препроцесор гіпертексту»; спочатку PHP / FI (Personal Home Page / Form Interpreter), а пізніше названий Personal Home Page Tools — «Інструменти для створення персональних веб-сторінок») — скриптова мова загального призначення, інтенсивно застосовується для розробки веб-додатків. В даний час підтримується переважною більшістю хостинг-провайдерів і є одним з лідерів серед мов, що застосовуються для створення динамічних веб-сайтів.

Мова і її інтерпретатор (Zend Engine) розробляються групою ентузіастів в рамках проекту з відкритим кодом. Проект поширюється під власною ліцензією, несумісною з GNU GPL.

3.2.9 Збірка веб-сервера XAMPP

XAMPP — безкоштовна багатоплатформова збірка веб-сервера з відкритим початковим кодом, що містить HTTP-сервер Apache, базу даних MariaDB, MySQL й інтерпретатори скриптів для мов програмування PHP та

Perl, а також додаткові бібліотеки, що дозволяють запустити повноцінний веб-сервер.

XAMPP — це акронім: 'X' (будь-яка з чотирьох операційних систем), 'A'pache, 'M'ySQL, 'P'HP, 'P'erl. Спочатку XAMPP створювався як інструмент для розробників, дозволяючи веб-дизайнерам та програмістам тестувати свою роботу, не використовуючи Інтернет. Для спрощення роботи деякі можливості та налаштування безпеки відключені за замовчуванням, і в цілому XAMPP рекомендується до використання тільки в дуже дружньому оточенні.

3.3 Модулі і алгоритми

Під час розробки програмного застосунку для створення анімації з можливістю налаштування стилістики та зовнішнього вигляду декорацій та персонажів було створено наступні модулі:

- модуль інтерфейсу;
- модуль створення відео;
- модуль покадрової обробки.

3.3.1 Модуль інтерфейсу

Інтерфейс розроблений за допомогою патерну MVC, у його некласичному варіанті, де контролер інтегровано до подання. Для кожного вікна існує свій клас-подання, також є клас-модель, який зберігає дані та посилання на підключення до бази даних MySQL.

Лістинг 2 Клас-модель

```
public static class App
{
    public static MySqlConnection connection; // з’єднання
з базою даних MySQL

    public static string connectionString; // параметри
з’єднання (сервер, логін, пароль та ін.)
```

```
    public static string forEpisodes = ""; // місце
зберігання списку епізодів

    public static Dictionary<string, PictureStore>
settings; // місце зберігання налаштувань зовнішнього
вигляду персонажів

    public static Sprite style; // місце зберігання
картинки загального стилю

    public static int iCurrentEpisode = -1; // індекс
епізода, який записується в поточний момент

    public static string folderPath; // посилання на
каталог модуля покадрової обробки відео

    public static void Connect(); // функція підключення до
бази даних

    public static void Disconnect(); // функція відключення
від бази даних

    public static void StartRecording() // функція
завантаження нового епізоду для запису та/або виходу з
режиму запису та переходу до покадрової обробки
    {
        iCurrentEpisode++;
        forEpisodes.Replace(" ", "");
        string[] episodes = forEpisodes.Split(',');
        if (iCurrentEpisode < episodes.Length)
        {
            SceneManager.LoadScene("Episode_" +
episodes[iCurrentEpisode], LoadSceneMode.Single);
        }
        else
        {
            Debug.Log("!");
            SaveTextureAsPNG();
            StartRestyle();
            Application.Quit();
        }
    }

    public static void SaveTextureAsPNG(); // функція
завантаження текстури обраної стилістики до каталога модуля
покадрової обробки

    public static void StartRestyle(); // активація модуля
покадрової обробки
```

```
}
```

Детально інтерфейс описано в розділі 3.5, нижче ж ми описуємо деталі його технічної реалізації.

Лістинг 3 Клас вікна вибору епізодів

```
public class SelectEpisodesScript : MonoBehaviour
{
    // елементи керування
    public Canvas selectEpisodesCanvas;
    public RectTransform episodes;
    public Toggle toggle; // шаблон чекбокса для епізода

    void Start(); // завантаження списку епізодів з бази
даних та їхній вивід на екран

    public void nextButtonClick() // обробка списку
епізодів та перехід до наступного вікна
    {
        int i = 1;
        foreach (Transform child in episodes)
        {
            if (child.GetComponent<Toggle>().isOn)
            {
                App.forEpisodes += (i.ToString() + ", ");
// тут відбувається оформлення списку епізодів у форматі,
придатному для використання в запитах MySQL
            }
            i++;
        }
        if (App.forEpisodes == “”)
        {
            EditorUtility.DisplayDialog("Помилка", "Не
обрано жоден епізод!", "OK");
        }
        else
        {
            App.forEpisodes = App.forEpisodes.Substring(0,
App.forEpisodes.Length – 2);

selectEpisodesCanvas.gameObject.SetActive(false); //
дезактивація форми вибору епізодів

this.GetComponent<SetCharactersScript>().setCharactersCanva
s.gameObject.SetActive(true); // активація форми
налаштування персонажів
```

```
this.GetComponent<SetCharactersScript>().SetLevel(1); //
вивід персонажів першого рівня на екран

this.GetComponent<SetCharactersScript>().GetQuantity();
        }
    }

    public void selectAllButton(bool fl); // функція
керування інтерфейсом (одночасний вибір/скасування вибору
усіх епізодів)
}
```

Лістинг 4 Клас вікна налаштування персонажів

```
public class SetCharactersScript : MonoBehaviour
{
    // елементи керування
    public Image image;
    public Text text;
    public RectTransform content;
    public RectTransform imageContent;
    public GameObject imageView;
    public Canvas setCharactersCanvas;
    public Sprite noneSprite;

    public Text mainText;
    public Text decoText;
    public Text episText;

    public Text characterText;
    public GameObject pictureText;

    // зберігання даних
    public Dictionary<string, PictureStore> settings = new
Dictionary<string, PictureStore>(); // збереження
налаштувань персонажів
    string currentCharacter; // персонаж, який
налаштовується в поточний момент

    int quantity; // загальна кількість персонажів (для
перевірки, чи всіх персонажів налаштовано)

    string sqlCharactersRequest; // запит, який повертає
список задіяних персонажів (наведено в розділі 3.3.5)
    string sqlPicturesRequest; // запит, який повертає
асоційовані з персонажем ілюстрації
    string sqlQuantityRequest; // запит на загальну
кількість персонажів
```

```
    public void SetLevel(int level); // вивід задіяних
персонажів певного рівня

    public void SetCharacter(string character); // вивід
картинок, асоційованих з певним персонажем

    public void SetPicture(Sprite picture, string painter)
    {
        image.sprite = picture;
        text.text = painter;
        settings[currentCharacter] = new PictureStore {
painter = painter, picture = picture };
        content.GetComponentsInChildren<Text>().Select(p =>
p).Where(p => p.text ==
currentCharacter).ToArray()[0].fontStyle =
FontStyle.Normal;
    }

    public void ClearImageView(); // очищення поля виводу
картинок

    public void ClearImage(); // очищення поля виводу
обраної картинки

    public void GetQuantity(); // отримання загальної
кількості персонажів

    public void Submit(); // збереження налаштувань та
перехід до наступної сцени
}
```

Лістинг 4.1 Структури для зберігання даних про вибір персонажа під час налаштування

```
public struct PictureStore
{
    public string painter;
    public Sprite picture;
}
```

Лістинг 5 Клас вікна вибору стилістики

```
public class SetStylisticsScript : MonoBehaviour
{
    // елементи керування
    public Canvas setStylisticsCanvas;
    public RectTransform imageView;
    public GameObject pictureText;
```

```
    public Image forStyleImage;
    public Text forStyleText;

    public void GetStyles() // отримання списку стилів та
їхній вивід на екран
    {
        setStylisticsCanvas.gameObject.SetActive(true);
        bool fl = true;
        string selectStylesString = "SELECT * from Стили";
        MySqlCommand selectStylesCommand = new
MySqlCommand(selectStylesString, App.connection);
        MySqlDataReader selectStylesDataReader =
selectStylesCommand.ExecuteReader();
        while (selectStylesDataReader.Read())
        {
            GameObject newPictureText = fl ? pictureText :
UnityEngine.Object.Instantiate(pictureText, imageView);
            Texture2D tex = new Texture2D(256, 256,
TextureFormat.RGBA32, false);

tex.LoadImage((byte[])selectStylesDataReader[2]);
            tex.Apply();

newPictureText.transform.GetChild(0).GetComponent<Image>().
sprite = Sprite.Create(tex, new Rect(0, 0, tex.width,
tex.height), new Vector2(0f, 0f));

newPictureText.transform.GetChild(1).GetComponent<Text>().t
ext = selectStylesDataReader[1].ToString();
            EventTrigger.Entry entry = new
EventTrigger.Entry();
            entry.eventID = EventTriggerType.PointerClick;
            entry.callback.AddListener((data) => {
SetStyle(newPictureText.transform.GetChild(0).GetComponent<
Image>().sprite,
newPictureText.transform.GetChild(1).GetComponent<Text>().t
ext); });

newPictureText.transform.GetChild(0).GetComponent<EventTrig
ger>().triggers.Add(entry);

newPictureText.transform.GetChild(1).GetComponent<EventTrig
ger>().triggers.Add(entry);
            if (fl) fl = false;
        }
        selectStylesDataReader.Close();
    }
```

```
    void SetStyle(Sprite picture, string style) // вибір
стилю та його вивід на екран
    {
        forStyleImage.sprite = picture;
        forStyleText.text = style;
    }

    public void Submit() // збереження налаштувань,
від'єднання від бази даних та початок запису
    {
        if (forStyleText.text != "")
        {
            App.style = forStyleImage.sprite;
            App.Disconnect();
            App.StartRecording();
        }
        else
        {
            EditorUtility.DisplayDialog("Помилка", "Не
обрано стиль!", "OK");
        }
    }

}
```

3.3.2 Модуль отримання списку персонажів та прев'ю

Суворо кажучи, це підрозділ попереднього модуля, хоча частково й винесений поза межі Unity та реалізований мовою SQL (хоча у зв'язку з тим, що в MySQL неможливо написати функцію, де параметром був би список параметрів, один із запитів довелося прописати безпосередньо у Unity):

Лістинг 6 Запит на отримання списку персонажів

```
SELECT Персонаж FROM Персонажи p WHERE p.idПерсонажи =
ANY(SELECT Персонаж FROM Участие WHERE Эпизод IN ({0})) AND
IF(((SELECT MIN(Ранг) FROM Участие u WHERE p.idПерсонажи =
u.Персонаж) = 1 AND(SELECT AVG(Ранг) FROM Участие u WHERE
p.idПерсонажи = u.Персонаж) >= 2.8), 3, (SELECT MIN(Ранг)
FROM Участие u WHERE p.idПерсонажи = u.Персонаж)) = {1}
```

Лістинг 7 Запит на отримання списку прев'ю

```
SELECT
    (SELECT Художник FROM Художники WHERE idХудожники =
Картины.Художник),
    Картина FROM Картины
   WHERE Персонаж = Charact;
```

3.3.3 Модуль створення відео

Даний модуль реалізовано за допомогою середовища розробки Unity Editor та, на жаль, для коректної праці потребує наявності цього середовища та запуска нашого додатка як проекта цього середовища. Він включає в себе 3D-сцену, скрипти, які надають їй заданого користувачем вигляду, та скрипт, який створює об’єкт Unity Recorder (який виконує запис того, що відбувається на сцені).

Лістинг 8 Клас, який працює з об’єктом Unity Recorder:

```
public class RecorderScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        var controllerSettings =
ScriptableObject.CreateInstance<RecorderControllerSettings>
();
        var TestRecorderController = new
RecorderController(controllerSettings);

        var videoRecorder =
ScriptableObject.CreateInstance<MovieRecorderSettings>();
        videoRecorder.name = "My Video Recorder";
        videoRecorder.Enabled = true;
        videoRecorder.VideoBitRateMode =
VideoBitrateMode.High;

        videoRecorder.ImageInputSettings = new
GameViewInputSettings
        {
            OutputWidth = 640,
            OutputHeight = 360
        };

        videoRecorder.AudioInputSettings.PreserveAudio =
true;
        videoRecorder.OutputFile = "C:\\Users\\Pushka
PC\\Documents\\try";

controllerSettings.AddRecorderSettings(videoRecorder);
        controllerSettings.SetRecordModeToFrameInterval(0,
59); // 2s @ 30 FPS
        controllerSettings.FrameRate = 30;

        RecorderOptions.VerboseMode = false;
```

```
        TestRecorderController.PrepareRecording();
        TestRecorderController.StartRecording();
    }


    public void OnSceneEnd(); // функція, яка завершує
сцену та завантажує нову (через функцію моделі
App.StartRecording() )
}
```

3.3.4 Модуль покадрової обробки відео

В основу модуля покадрової обробки відео задля надання потрібної стилістики покладено реалізацію алгоритму Л. Ґетіса мовою C#, розроблену М. Фаррагхером [8]. Цей модуль широко використовує інструментарій Microsoft CNTK, з чим пов’язана певна його проблемність (CNTK на даний момент поступово застаріває). Тим не менш для наших завдань цей модуль цілком підійшов.

Наведемо лістинг головних класів, де власне відбувається обробка:

Лістинг 9 Клас завантаження відео та перебору кадрів

```
class Program
    {
        // paths to the content and style images
        static readonly string styleImagePath =
"style.png";
        static readonly string inputVideoPath =
"Episode_{0}.mp4";
        static readonly string outputVideoPath =
"StylizedEpisode_{0}.mp4";

        // path to the episodes list
        static readonly string episodesListPath =
"episodes.dat";

        // the width and height to resize the images to
        static readonly int imageHeight = 300;
        static readonly int imageWidth = 400;

        /// <summary>
        /// The main program entry point.
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {
```

```
            // check compute device
            Console.WriteLine($"  Using
{NetUtil.CurrentDevice.AsString()}");

            var styleImage =
StyleTransfer.LoadImage(styleImagePath, imageWidth,
imageHeight);

            StreamReader sr = new
StreamReader(episodesListPath);
            string episodes = sr.ReadToEnd();
            string[] episodesArray = episodes.Split(',');

            for (int i = 0; i < episodesArray.Length; i++)
            {

                VideoFileReader reader = new
VideoFileReader();
                reader.Open(string.Format(inputVideoPath,
episodesArray[i]));
                int fcc = VideoWriter.FourCC('M', 'P', '4',
'V'); //'M', 'J', 'P', 'G'
                float fps = 30F;
                VideoWriter writer = new
VideoWriter(string.Format(outputVideoPath,
episodesArray[i]), fcc, fps, new
OpenCvSharp.Size(imageWidth, imageHeight), true);
                //writer.Open(CariProjeAdres +
"\\result.mp4", imageWidth, imageHeight, 30,
VideoCodec.MPEG4, 1000000);
                for (int j = 0; j < reader.FrameCount; j++)
                {
                    Bitmap videoFrame =
reader.ReadVideoFrame();
                    var matrix = videoFrame.ToMat();
                    videoFrame.Dispose();
                    var matrix2 = new Mat(imageHeight,
imageWidth, matrix.Type());
                    Cv2.Resize(matrix, matrix2, new
OpenCvSharp.Size(imageWidth, imageHeight));
                    var contentImage =
StyleTransfer.FlattenByChannel(matrix2,
StyleTransfer.VGG19_Offsets);

                    // create the feature variable
                    var featureVariable =
CNTK.Variable.InputVariable(new int[] { imageWidth,
imageHeight, 3 }, CNTK.DataType.Float);
```

```
                    // create the neural network base (just
the content and style layers)
                    var model = featureVariable
                        .VGG19(freeze: true)
                        .StyleTransferBase();

                    // calculate the labels
                    var labels =
StyleTransfer.CalculateLabels(model, contentImage,
styleImage);

                    // add the dream layer
                    model = model.DreamLayer(contentImage,
imageWidth, imageHeight);

                    // create the label variable
                    var contentAndStyle =
model.GetContentAndStyleLayers();
                    var labelVariable = new
List<CNTK.Variable>();
                    for (int k = 0; k < labels.Length; k++)
                    {
                        var shape =
contentAndStyle[k].Shape;
                        var input_variable =
CNTK.Variable.InputVariable(shape, CNTK.DataType.Float,
"content_and_style_" + k);
                        labelVariable.Add(input_variable);
                    }

                    // create the loss function
                    var lossFunction =
StyleTransfer.CreateLossFunction(model, contentAndStyle,
labelVariable);

                    // set up an AdamLearner
                    var learner = model.GetAdamLearner(10,
0.95);

                    // get the model trainer
                    var trainer = model.GetTrainer(learner,
lossFunction, lossFunction);

                    // create the batch to train on
                    var trainingBatch =
StyleTransfer.CreateBatch(lossFunction, labels);

                    // train the model
```

```
                    Console.WriteLine("Training the
model...");
                    var numEpochs = 300;
                    for (int k = 0; k < numEpochs; k++)
                    {

trainer.TrainMinibatch(trainingBatch, true,
NetUtil.CurrentDevice);
                        if (k % 50 == 0)
                            Console.WriteLine($"epoch {k},
training loss = {trainer.PreviousMinibatchLossAverage()}");
                    }

                    // create a batch to evaluate the model
on
                    var evaluationBatch =
StyleTransfer.CreateBatch(model, labels);

                    // infer the image from the model
                    Console.WriteLine("Inferring
transformed image...");
                    var img =
model.InferImage(evaluationBatch);

                    // show image
                    var mat = new Mat(imageHeight,
imageWidth, MatType.CV_8UC3, img, 3 * imageWidth);
                    writer.Write(mat);
                    model.Dispose();
                }
                reader.Close();
                writer.Dispose();
            }
        }
    }
```

3.3.5 Модуль завантаження зображень до бази MySQL

Цей модуль не входить до основного тексту програми, однак був незамінним під час її написання. MySQL Workbench не можна напряму підключити до диска та завантажити прев'ю спрайта персонажа або стилю, тож це робилося через *.html та *.php-файли, запущені через XAMPP.

Лістинг 10 Html-файл

```
<html>
```

```
<body>
<form method="POST" action="getdata.php"
enctype="multipart/form-data">
<label>Painter</label>
<input type="number" name="painter"><br>
<label>Character</label>
<input type="text" name="character"><br>
<input type="file" name="myimage">
<label>Is Style?</label>
<input type="checkbox" name="isStyle"><br>
<input type="submit" name="submit_image" value="Upload">
</form>
</body>
</html>
```

Лістинг 11 Php-файл

```
<?php
$host; // хост
$user; // логін
$pass; // пароль
$link = mysqli_connect($host, $user, $pass);
mysqli_select_db($link, 'mim_anim');
//Получаем содержимое изображения и добавляем к нему слеш
$imagetmp=addslashes(file_get_contents($_FILES['myimage']['
tmp_name']));
$painter = $_POST["painter"];
$character = $_POST["character"];
//Вставляем имя изображения и содержимое изображения в
image_table
$insert_image="INSERT INTO `картины` (`персонаж`,
`художник`, `картина`) VALUES('$painter', '$character',
'$imagetmp')";
if ($_POST["isStyle"]) $insert_image="INSERT INTO `стили`
(`название`, `пример`) VALUES('$character', '$imagetmp')";
$result = mysqli_query($link, $insert_image);
?>
```

3.4 Структури даних

Зберігання даних про майбутню анімацію (епізоди, персонажі, їхні можливі спрайти) є ресурсомісткою задачею. При розробці інтерфейсу було вирішено застосувати базу даних MySQL для покращення якості зберігання і швидкості доступу до даних. На Рисунок *15* зображена діаграма бази даних.

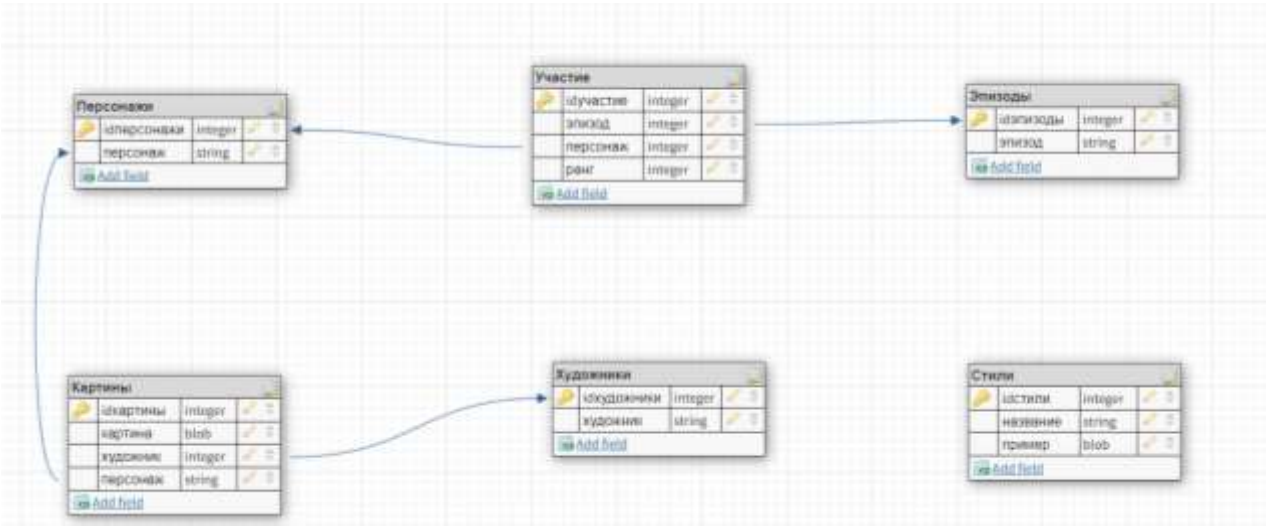

Рисунок *15. Схема бази даних*

Основні сутності — «Участие», яка зберігає дані про участь персонажа або декорації в тому чи іншому епізоді, а також про ранг участі (головний персонаж / декорація / епізодичний персонаж), та «Картина», яка зберігає прев'ю зовнішнього вигляду персонажів. Сутність «Эпизоды» зберігає список епізодів та їхні умовні назви (для спрощення компонування змісту мультфільму користувачем), «Персонажи» — список імен персонажів, «Художники» — список імен художників. «Стили» — зразки стилістики, які потім використовуватимуться для покадрової обробки, та їхні назви. В кожній таблиці поле, яке так чи інакше стосується іншої сутності, має властивість Foreign Key (виняток — «Стили»).

3.5 Проект інтерфейсу

Програму обладнано інтерфейсом з трьох вікон. На першому відбувається вибір епізодів, які ввійдуть до кінцевого відео. Вікно зроблене в формі списку чекбоксів, до яких також додано кнопки «Обрати все» та «Скасувати все», а також кнопка продовження роботи.

Рисунок *16 Вікно вибору епізодів*

У випадку спроби продовження роботи без вибору жодного епізоду програма видасть повідомлення про помилку (у даній роботі, зважаючи на неминучість використання інструментарію UnityEditor, ми вважаємо за можливим скористатися MessageBox'ом з цього простору імен):

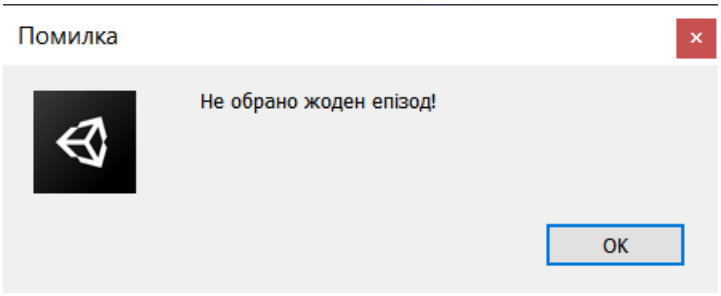

Рисунок *17 Вікно помилки при виборі епізодів*

Друге вікно — вікно налаштування зовнішнього вигляду персонажів та декорацій майбутнього мультфільму. Персонажів відсортовано за частотою появи в загальному об'ємі відео за категоріями: Головні персонажі — Епізодичні персонажі.

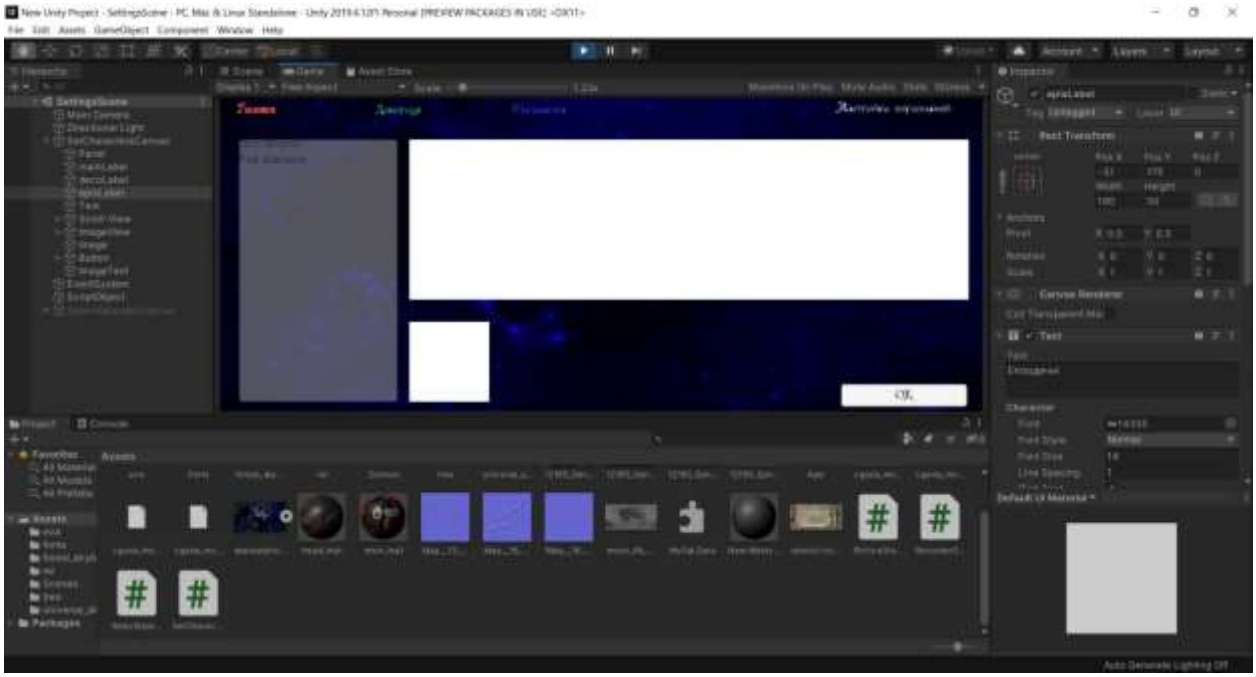

Рисунок *18 Вікно вибору епізодів*

Натискання на персонажа викликає список можливих його втілень на екрані, з прев’ю та іменем художника.

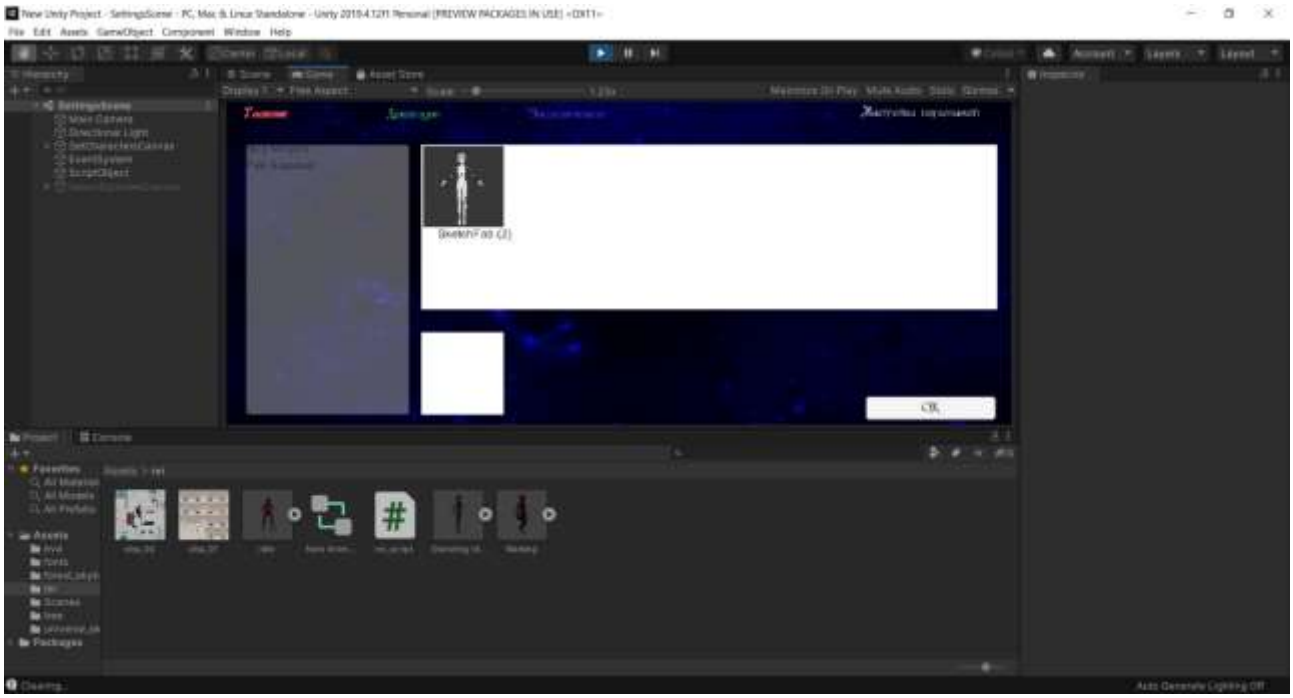

Рисунок *19 Вікно вибору персонажів (з прев'ю)*

Далі користувачеві надається вибір (на жаль, через брак 3D-моделей у даній роботі ми змогли реалізувати його лише теоретично; та що там, навіть Снігуроньку довелося замінити на Аянамі Рей, чий спрайт було знайти значно легше…) Обрана картинка відображається в нижній частині екрану та запам’ятовується програмою (якщо під час налаштування звернутися до цього персонажа повторно, попередній вибір відобразиться). З метою поліпшення роботи персонажі, чий вигляд ще не вибрано, підсвічуються жирним шрифтом.

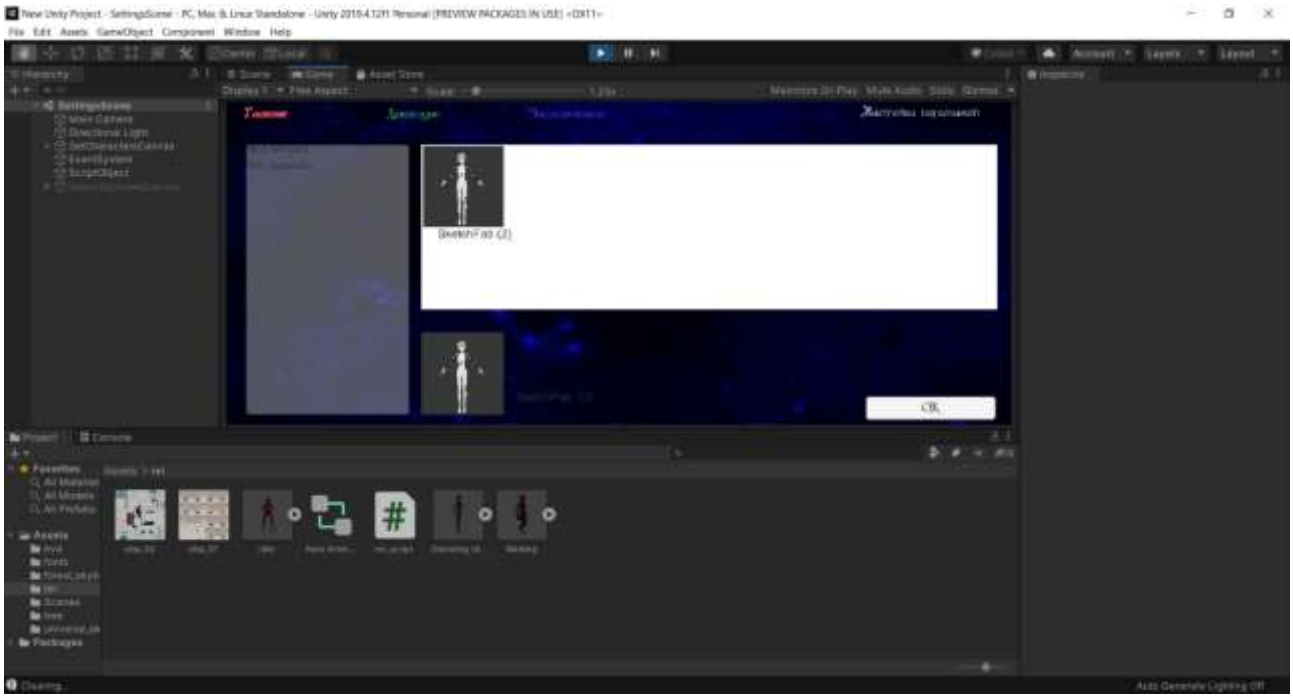

Рисунок *20 Вікно вибору персонажів (з обраним прев'ю)*

Якщо буде спроба піти далі, не завершивши налаштування — також буде повідомлення про помилку.

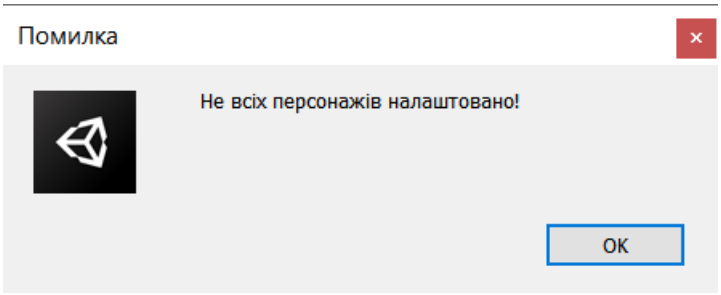

Рисунок *21 Вікно помилки під час вибору персонажів*

Третє вікно — вибір стилістики. Як і у попередньому вікні, у верхній частині відображено загальний список, у нижній — обраний стиль. За спроби піти далі, не обравши стилістику, буде виведено повідомлення про помилку.

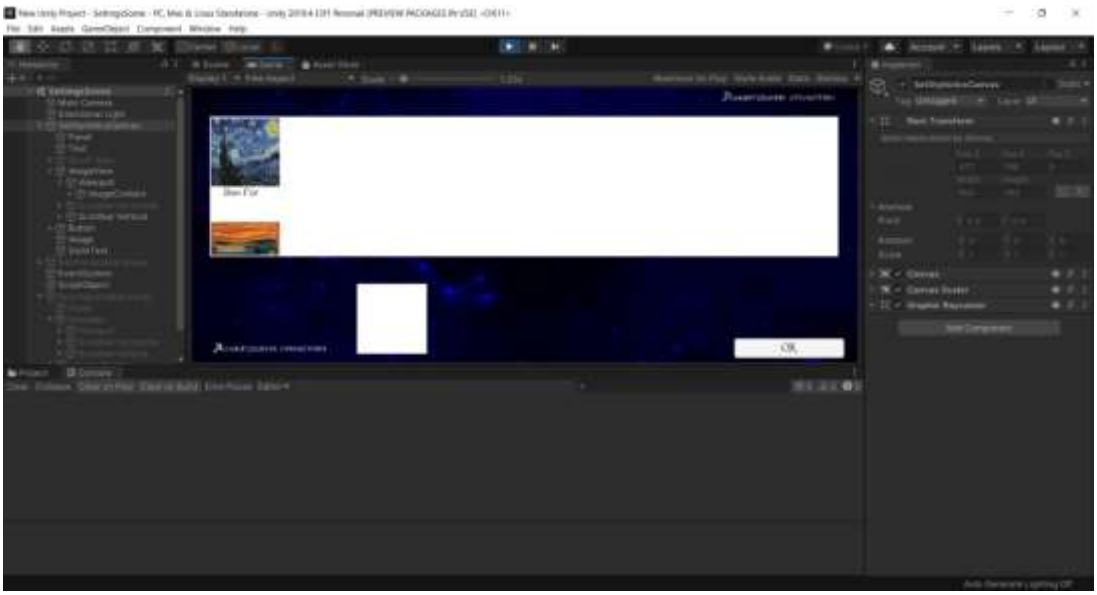

Рисунок *22. Вікно вибору стилістики*

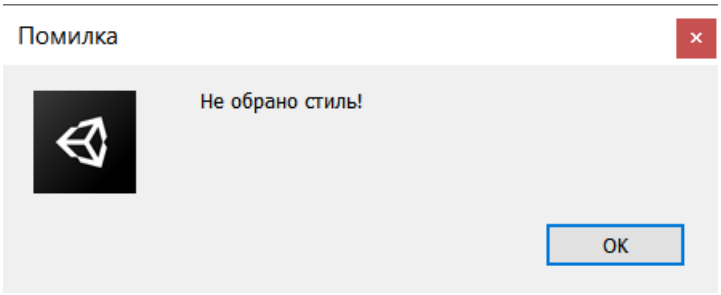

Рисунок *23. Вікно помилки під час вибору стилістики*

3.6 Вимоги до апаратного забезпечення

Основною задачею застосунку є рендеринг 3D-об’єктів, відео та покадрова обробка відео за допомогою нейронних мереж. Тому були визначені наступні мінімальні вимоги до застосунку.

Мінімальні вимоги до апаратного забезпечення:

- Оперативна пам’ять 16 Гб
- Чотирьох-ядерний процесор з базовою тактовою частотою 3,33 GGz;
- Відеоплата (GPU) NVIDIA GeForce GTX

3.7 Опис функціональних можливостей

Оскільки основною метою кваліфікаційної роботи є дослідження стану питання та наявних алгоритмів та методів вирішення проблеми, а не розробка кінцевого програмного продукту, функціональні можливості системи орієнтовані на зручність проведення досліджень, а не для використання у якості готової програмної системи для створення анімації із можливістю налаштування зовнішнього вигляду персонажів та загальної стилістики. Однак вже є ескіз дружнього графічного користувацького інтерфейсу, який цілком можна використовувати в потенційній готовій програмній системі. Кінцевою метою роботи програми на даний момент є анімаційні ролики невеличкої тривалості та, на жаль, поки що (через брак апаратно-програмних ресурсів) невисокої якості, але вже з потрібними сюжетом, зовнішнім виглядом персонажів та стилем зображення.

РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ СТВОРЕННЯ АНІМАЦІЇ З МОЖЛИВІСТЮ НАЛАШТУВАННЯ СТИЛІСТИКИ ТА ЗОВНІШНЬОГО ВИГЛЯДУ ДЕКОРАЦІЙ І ПЕРСОНАЖІВ

4.1 Дослідження результатів корекції стилістики за допомогою наявної реалізації алгоритму Ґетіса

Під час розробки модуля корекції стилістики відео цей модуль було протестовано на різноманітних зображеннях, з варіюванням розміру та кількості епох.

4.1.1 Дослідження впливу кількості епох на результат

Перша серія тестів була пов’язана з визначенням оптимальної кількості епох обробки зображення нейронною мережею. М. Фаррагхер пропонував в своїй роботі значення 300 епох [8].

Розглянемо вплив кількості епох на конкретних прикладах.

Рисунок *24 Перший експеримент (оригінал зображення)*

Рисунок *25 Перший експеримент (джерело стилю)*

Рисунок *26. Результат першого експерименту (300 епох)*

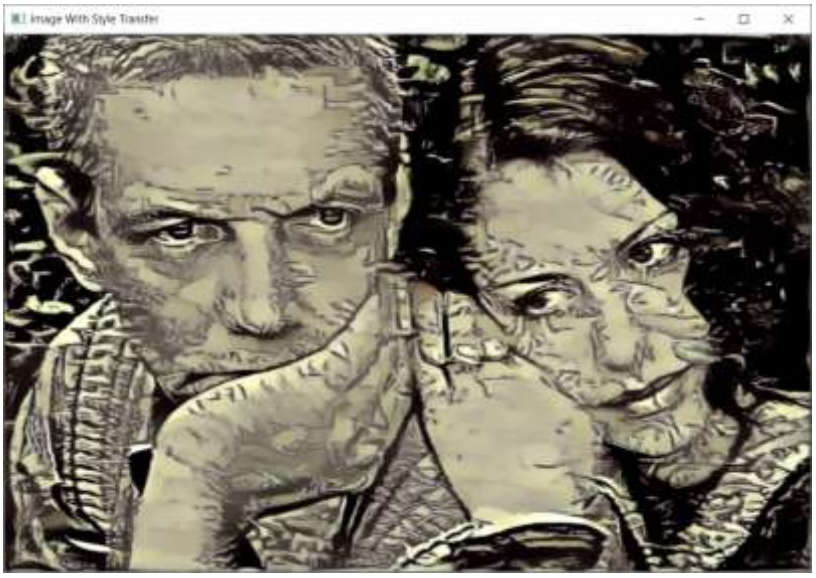

Рисунок *27 Результат першого експерименту (3000 епох)*

Як бачимо, збільшення кількості епох веде до більшої рівномірності кольору на зображенні та більш виявленому гравюрному стилю. Цілком аналогічні результати експерименту з переведенням цього ж зображення в малюнок олівцем.

Рисунок *28 Інше джерело стилю*

Рисунок *29. Результат(з 300 епохами)*

Рисунок *30 Результат (з 3000 епохами)*

Тут різниця, втім, менше кидається у вічі, аніж у випадку з гравюрою.

Однак далеко не у всіх випадках збільшення кількості епох веде до зменшення функції втрат. Під час наших експериментів були випадки, коли така під час нейромережної обробки збільшувалася. Крім того, необхідно враховувати загальні витрати часу на обробку зображень. Тому ми вирішили зупинитися на значенні 300 епох, яке пропонував М. Фаррагхер.

4.1.2 Дослідження оптимального розміру зображення

Другий (в принципі, інтуїтивно зрозумілий) важливий аспект проблеми — розмір зображення. Що більше зображення, тим більше деталей зберігається й тим менше зміст зображення спотворюється.

Рисунок *31 Оригінал зображення*

Рисунок *32 Джерело стилю*

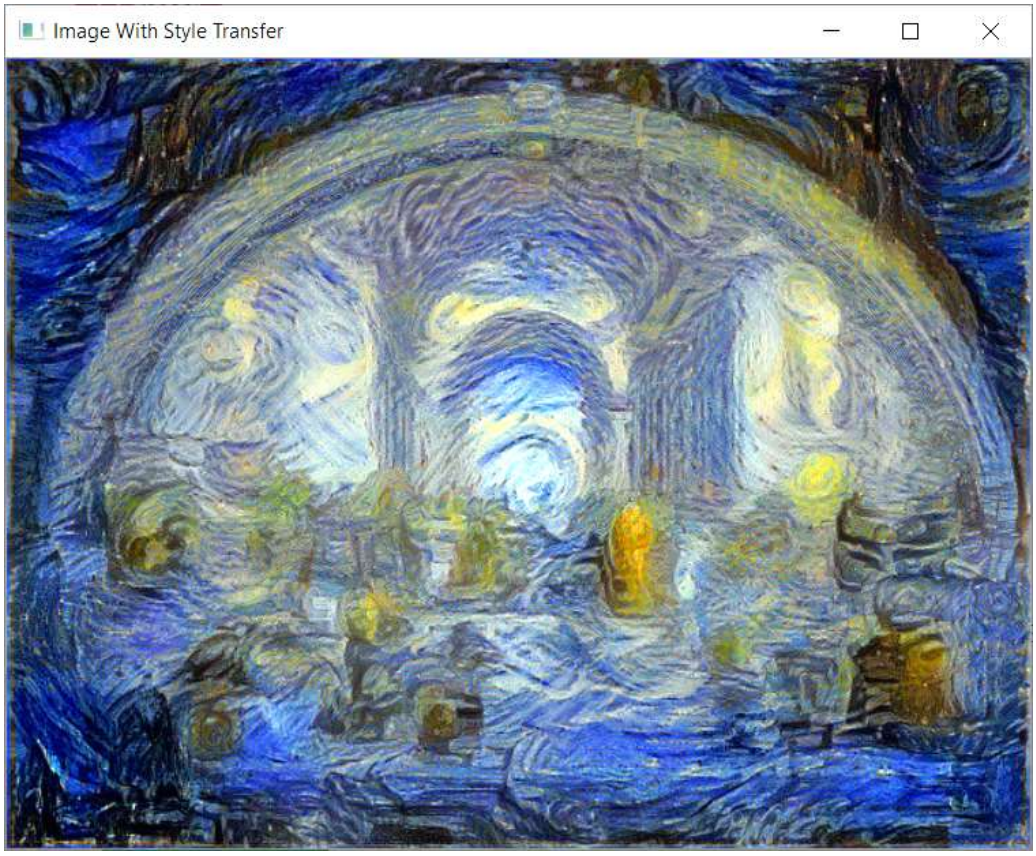

Рисунок *33 Результат першого експерименту (755 * 621)*

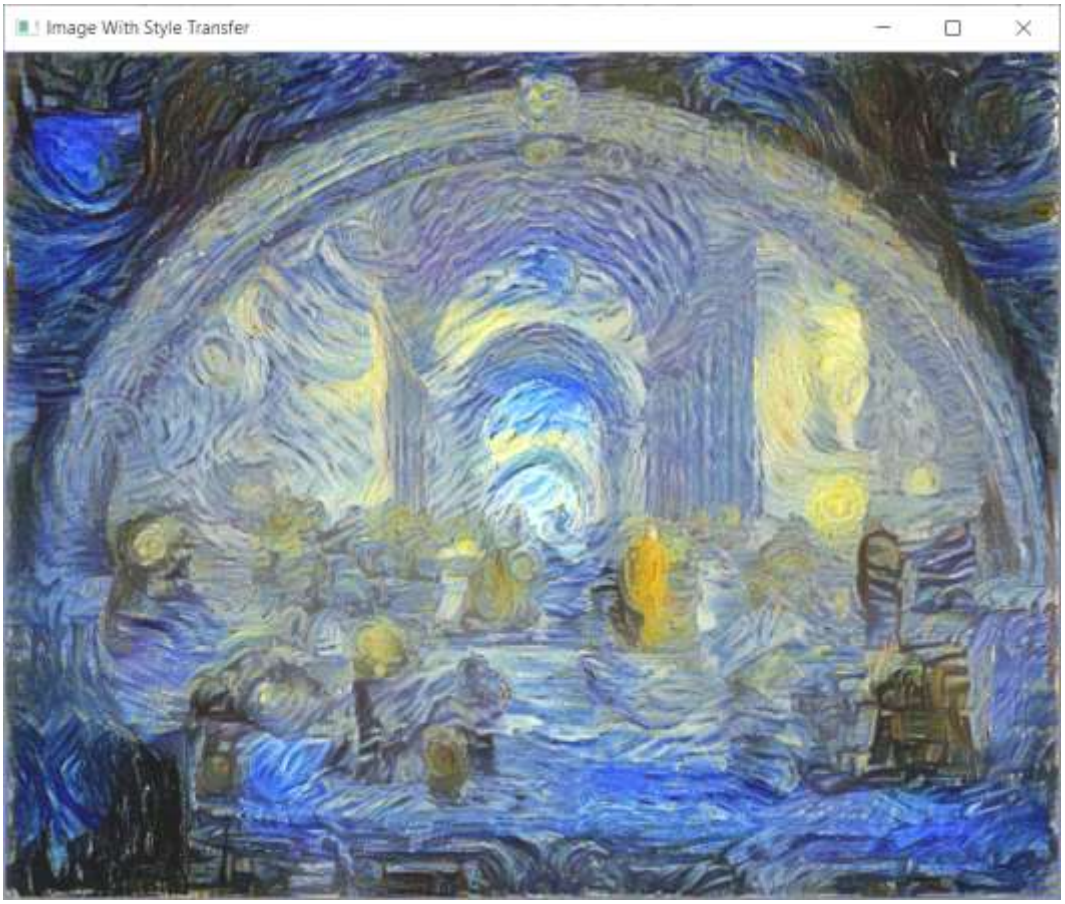

Рисунок *34 Результат другого експерименту (880 * 746)*

Як бачимо, інтуїтивно зрозуміла кореляція між розміром зображення та точністю відтворення цілком підтверджується.

Однак, на жаль, через апаратні характеристики нашого комп’ютера існує певна межа зображення, до якого можна пристосувати зображення — близько 500 000 пікселів. У випадку ж створення відео ця межа знижується ще більше, до 200 000. Таким чином, ми зупинилися на розмірі 300*400.

4.1.3 Дослідження можливостей зміни контурів зображення

У алгоритма Ґетіса існують певні обмеження в застосовності. Наприклад, він практично безсилий в зміні характера промальовки контуру. Наведемо приклад.

Рисунок *35 Оригінал зображення*

Рисунок *36 Джерело стилю*

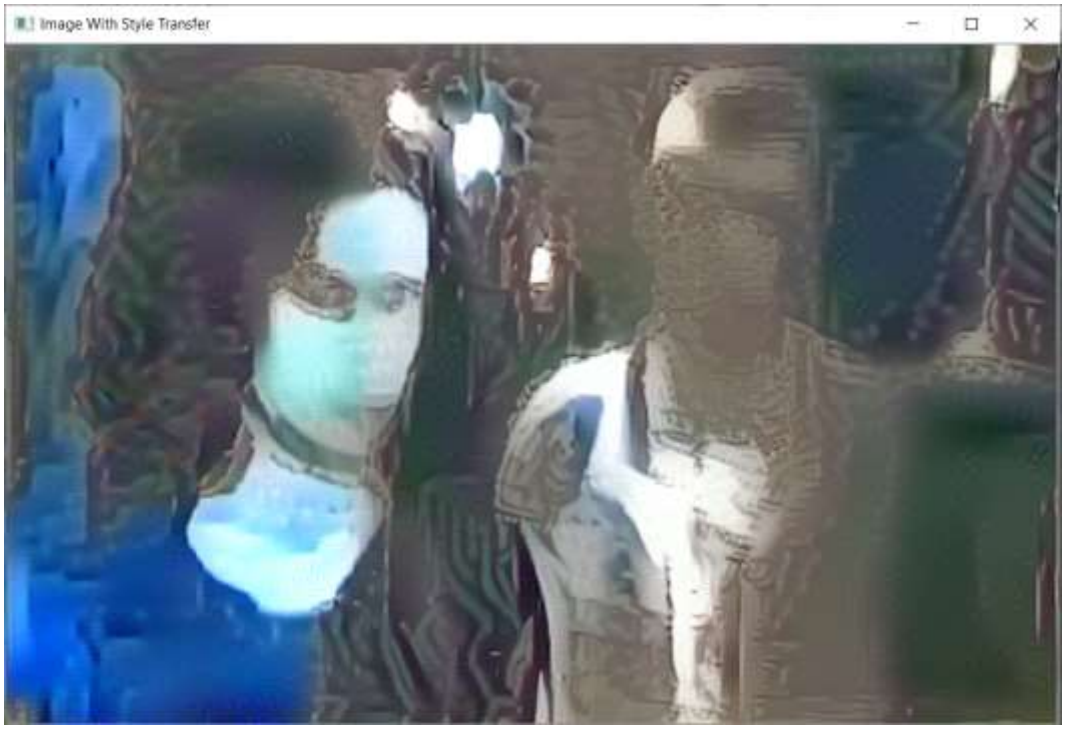

Рисунок *37 Результат роботи*

Таким чином, для зміни принципів промальовки контуру (наприклад, з фотостилю на аніме-стиль) цей алгоритм не підходить — хоча можливості імітації нереалістичного стилю (малюнок олівцем, гравюра, стиль Моне чи Ван Гога) у цього алгоритма чудові [31].

ВИСНОВКИ

1. Встановлено актуальність розробки системи створення анімації з можливістю налаштування зовнішнього вигляду персонажів та загальної стилістики зображення. Досліджено основні підходи до цієї проблеми та методи їх реалізації, проведено їх порівняльний аналіз.

2. Досліджена можливість використання алгоритму Ґетіса у анімації та покадровій обробці відео.

3. Основним інструментом реалізації програмної системи обрано середовище створення ігор Unity, що надає великий спектр можливостей.

4. Розроблено інтуїтивно зрозумілий користувачеві інтерфейс налаштування зовнішнього вигляду персонажів.

5. Розроблено модулі створення відео з заздалегідь налаштованим зовнішнім виглядом персонажів та покадрової обробки відео з наданням йому заздалегідь налаштованої стилістики.

6. Проведено аналіз отриманих результатів і виконано тестування модуля надання стилістики як на окремих зображеннях, так і на відео.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Averbuch-Elor H., Cohen-Or D., Kopf J., and Cohen M. F. Bringing portraits to life. ACM Transactions on Graphics (Proceeding of SIGGRAPH Asia 2017), 36(4):to appear, 2017.
2. Bansal Aayush, Ma Shugao, Ramanan Deva, and Sheikh Yaser. Recycle-GAN: Unsupervised Video Retargeting. ECCV 2018 // arXiv:1808.05174 [cs.CV].
3. Barnes Connelly, Shechtman Eli, Finkelstein Adam, Goldman Dan B. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. ACM Transactions on Graphics (Proc. SIGGRAPH) 28(3), August 2009.
4. Bogo F., Kanazawa A., Lassner C., Gehler P., Romero J., and Black M. J. Keep it SMPL: Automatic estimation of 3D human pose and shape from a single image. In Computer Vision. ECCV 2016, Lecture Notes in Computer Science. Springer International Publishing, Oct. 2016.
5. Chen Y., Lai Yu-Kun, Liu Yong-Jin. CartoonGAN: Generative Adversarial Networks for Photo Cartoonization. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).
6. Chuang Y.-Y., Goldman D. B., Zheng K. C., Curless B., Salesin D. H., Szelisk R. Animating pictures with stochastic motion textures. ACM Transactions on Graphics, Vol 24, No 3, to appear, (Proceedings of ACM SIGGRAPH 2005, July 2005, Los Angeles).
7. CNTK v2.7 Release Notes, URL: https://docs.microsoft.com/en-us/cognitive-toolkit/releasenotes/cntk_2_7_release_notes (дата звернення 12.12.2020).
8. Farragher M. Artistic Style Transfer With C# And A Neural Network, URL: https://medium.com/machinelearningadvantage/artistic-style-transfer-with-c-and-a-cntk-neural-network-2e53e29c194e (дата звернення 12.12.2020).
9. Gatys Leon A., Ecker Alexander S., Bethge Matthias. A Neural Algorithm of Artistic Style // arXiv:1508.06576 [cs.CV].

10. Goodfellow, Ian J.; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). «Generative Adversarial Networks». arXiv:1406.2661 [stat.ML].
11. Hornung A., Dekkers E., and Kobbelt L. Character animation from 2d pictures and 3d motion data: ACM Transactions on Graphics (TOG), 26(1), 2007.
12. Jain A., Thorm¨ahlen T., Seidel H.-P., and Theobalt C. Moviereshape: Tracking and reshaping of humans in videos. In ACM Transactions on Graphics (TOG), volume 29, page 148. ACM, 2010.
13. Kaushik A. Understanding the VGG19 Architecture, URL: https://iq.opengenus.org/vgg19-architecture/ (дата звернення 12.12.2020).
14. Kholgade N., Simon T., Efros A., and Sheikh Y. 3d object manipulation in a single photograph using stock 3d models: ACM Transactions on Graphics (TOG), 33(4), 2014,pp.127.
15. Krizhevsky, A., Sutskever, I. & Hinton, G. E. Imagenet classification with deep convolutional neural networks. Neural Information Processing Systems 25, 2012/01/01, doi: 10.1145/3065386.
16. Loper Matthew, Mahmood Naureen, Romero Javier, Pons-Moll Gerard, Black Michael J. SMPL: A Skinned Multi-Person Linear Model. ACM Transactions on Graphics, October 2015, Article No.: 248. URL: https://doi.org/10.1145/2816795.2818013 (дата звернення 12.12.2020).
17. MySQL :: MySQL Workbench, URL: https://www.mysql.com/products/workbench (дата звернення 12.12.2020);
18. Oliveyra Y. A UI System Architecture and Workflow for Unity, URL: https://www.gamasutra.com/blogs/YankoOliveira/20180108/312617/An_UI_System_Architechture_and_Workflow_for_Unity.php (дата звернення 12.12.2020).

19. Peterson Jordan B. I didn’t say that,
URL: https://www.jordanbpeterson.com/blog-posts/i-didnt-say-that/ (дата звернення 12.12.2020).
20. Shet R.N., Lai K.H., Edirisinghe E.A., Chung P.W.H. Use Of Neural Networks In Automatic Caricature Generation: An Approach Based On Drawing Style Capture. IEE International Conference on Visual Information Engineering (VIE 2005), Glasgow, UK, 2005, pp. 1-7.
21. Suwajanakorn Supasorn, Seitz Steven M., and Kemelmacher-Shlizerman Ira. Synthesizing Obama: Learning Lip Sync from Audio. ACM Trans. Graph. 36, 4, Article 95 (July 2017), 13 pages. DOI. URL: http://dx.doi.org/10.1145/3072959.3073640 (дата звернення 12.12.2020).
22. Tseng Chien-Chung and Lien Jenn-Jier James. Synthesis of Exaggerative Caricature with Inter and Intra Correlations. In: Yagi Y., Kang S.B., Kweon I.S., Zha H. (eds) Computer Vision – ACCV 2007. ACCV 2007. Lecture Notes in Computer Science, vol 4843. Springer, Berlin, Heidelberg.
23. Thies J., Zollhofer M., Stamminger M., Theobalt C., Nießner M. Face2Face: Real-time Face Capture and Reenactment of RGB Videos // Proc. Computer Vision and Pattern Recognition (CVPR), IEEE
24. Wei Shih-En, Ramakrishna Varun, Kanade Takeo, Sheikh Yaser. Convolutional Pose Machines // Machines, arXiv:1602.00134 [cs.CV].
25. Weng Chung-Yi, Curless Brian, Kemelmacher-Shlizerman Ira. PhotoWake-Up: 3D Character Animation from a Single Photo, arXiv:1812.02246v1 [cs.CV] 5 Dec 2018.
26. Xu X., Wan L., Liu X., Wong T.-T., Wang L., Leung C.-S. Animating animal motion from still. ACM Transactions on Graphics 27, 2008, 117 p.
27. Zakharov E., Shysheya A., Burkov E., Lempitsky V. Few-Shot Adversarial Learning of Realistic Neural Talking Head Models, arXiv:1905.08233 [cs.CV]
28. Zhou S., Fu H., Liu L., Cohen-Or D., and Han X.. Parametric reshaping of human bodies in images. In ACM Transactions on Graphics (TOG), volume 29: ACM, 2010. 126 p.

29. Zhu Jun-Yan, Park Taesung, Isola Phillip, Efros Alexei A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 2242-2251.
30. Бодягин И. Model-View-Controller в .Net. Model-View-Presenter и сопутствующие паттерны,
URL: http://rsdn.org/article/patterns/modelviewpresenter.xml (дата звернення 14.12.2020);
31. Комар А. П., студент магістратури ФЕЕІТ ІІ ЗНУ. Наук. кер.: к.т.н., доц. Заяц В.І. «ДОСЛІДЖЕННЯ АЛГОРИТМІВ МОДИФІКАЦІЇ СТИЛІВ ЗОБРАЖЕННЯ». Збірник наукових праць студентів, аспірантів і молодих вчених «Молода наука-2020» : у 5 т. Запорізький національний університет. Запоріжжя: ЗНУ, 2020. Т.5. С. 165;
32. Полное руководство по языку программирования C# 9.0 и платформе .NET 5, URL: https://metanit.com/sharp/tutorial/ (дата звернення 12.12.2020).
33. Приемы при проектировании архитектуры игр,
URL: https://habr.com/ru/post/255561/ (дата звернення 12.12.2020);
34. Руководство по MySQL,
URL: https://metanit.com/sql/mysql/ (дата звернення 12.12.2020).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Комар Андрій Павлович, студент _2_ курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти sp115-01@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Створення анімації за заданим стилем**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2020 Підпис _____ Комар Андрій Павлович (студент)

Дата 30.11.2020 Підпис _____ Заяц Валерій Іванович (науковий керівник)