

ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики

«До захисту допущено»

Завідувач кафедри ПМІ

(підпис) О.А. Дмитрієва
(ініціали, прізвище)

«____» _____ 2021 р.

Випускна кваліфікаційна робота
магістра
(освітній ступінь)

на тему

«Створення 3D-моделі обличчя по її 2D зображенням»

Виконав: студент 2 курсу, групи ПЗм-20
(шифр групи)

напряму підготовки (спеціальності) спеціальності «121 Інженерія програмного
забезпечення» (шифр і назва напряму підготовки)

Погосян А. М.

(прізвище та ініціали)

(підпис)

Керівник д.т.н., проф. Башков Є. О.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

*Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.*

Студент

(підпис)

Покровськ - 2021

ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

Освітньо-кваліфікаційний рівень (освітній ступінь) магістр

Напрямок підготовки (спеціальність) 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри

Дмитрієва О. А.

/_____/

“___” _____ 2021 р.

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Погосян Арарат Мелікович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Створення 3D-моделі обличчя по її 2D зображенням

Керівник проекту (роботи) Башков Євген Олександрович, д.техн.н., проф.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від “22” вересня 2021 року № 570

2. Строк подання студентом проекту (роботи) _____

3. Вихідні дані до проекту (роботи) результати з науково-дослідних робіт, матеріали наукових видань, матеріали переддипломної практики, відомості з літератури, довідників, інтернет ресурси.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) дослідження сутності й актуальності генерації 3D обличчя; постановка завдання та визначення критеріїв оцінки; проектування архітектури моделі і програмного додатку; проектування програмної системи пошуку, вилучення характеристики, та генерації 3D обличчя; розробка програмного засобу; оформлення правил користування програмним забезпеченням; тестування програмного додатку; оцінка результатів програми;

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

1) блок-схеми розроблених алгоритмів;

3) екранні форми;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Підпис, дата	Підпис, дата

7. Дата видачі завдання 22 вересня 2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Дослідження сутності й актуальності генерації обличчя	12.10.2021	
2.	Постановка завдання та визначення критеріїв оцінки	15.10.2021	
3.	Проектування архітектури моделі і програмного додатку	28.10.2021	
4.	Розробка системи розпізнавання обличчя	02.11.2021	
5.	Розробка системи вилучення характеристик обличчя	8.11.2021	
6	Розробка системи генерації моделі обличчя	12.11.2021	
6.	Розробка програмної системи	17.11.2021	
7.	Тестування програмного додатку	19.11.2021	
8	Оцінка результатів програми	21.11.2021	
9.	Оформлення пояснювальної записки	01.12.2021	
10.	Підготовка слайдів презентації	05.12.2021	
11.	Підготовка демонстраційної версії розробленого програмного продукту	6.12.2021	
12.	Написання доповіді	7.12.2021	

Студент

Керівник роботи

(підпис) Погосян А.М.
(прізвище та ініціали)

(підпис) Башков Є. О.
(прізвище та ініціали)

АНОТАЦІЯ

Погосян А. М. Створення 3D-моделі обличчя по її 2D зображенню / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення». – ДВНЗ ДонНТУ, Покровськ, 2021.

Керівник д. техн. н., проф. Башков Євген Олександрович.

Об'єктом дослідження є процеси 3D-моделювання обличчя за допомогою підходу 3D Morphable Model.

Предмет роботи пов'язано з удосконаленням методу головних компонентів (PCA, principal component analysis) при моделюванні зображень обличчя людини.

Мета роботи полягає у розробці, обґрунтуванні та програмній реалізації системи для створення тривимірної моделі обличчя по її двовимірним зображенням (фотографіям).

Наукова новизна роботи полягає в отриманні удосконаленого алгоритму PCA (principal component analysis – метод головних компонентів).

Практична цінність роботи полягає в універсальності розробленого алгоритму та програмної реалізації, яка дозволить моделювати різні обличчя людей з кращою якістю.

Ключові слова: ГЕНЕРАЦІЯ МОДЕЛІ ОБЛИЧЧЯ, РОЗПІЗНАВАННЯ ОБЛИЧЧЯ, НЕЙРОННІ МЕРЕЖІ, АНАЛІЗ ВЛАСТИВОСТЕЙ ОБЛИЧЧЯ

SUMMARY

Poghosyan A. M. Creation of a 3D-model of the face on its 2D image / Final qualification work for the degree of "master" in the specialty 121 "Software Engineering". - SHEI DonNTU, Pokrovsk, 2021.

Head of Tech. n., prof. Bashkov Eugene Alexandrovich.

The object of research is the processes of 3D modeling of the face using the 3D Morphable Model approach.

The subject of the work is related to the improvement of the method of principal components (PCA, principal component analysis) in the modeling of images of the human face.

The purpose of the work is to develop, substantiate and software implementation of the system for creating a three-dimensional model of the face on its two-dimensional images (photographs).

The scientific novelty of the work is to obtain an advanced algorithm PCA (principal component analysis - the method of principal components).

The practical value of the work lies in the versatility of the developed algorithm and software implementation, which will model different faces of people with the best quality.

Keywords: FACE MODEL GENERATION, FACE RECOGNITION, NEURAL NETWORKS, ANALYSIS OF FACE PROPERTIES

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ	5
Вступ.....	6
Розділ 1 Аналіз та постановка задачі	12
1.1 Аналіз методів розпізнавання обличчя.....	12
1.1.1 На основі знань	12
1.1.2 PCA.....	12
1.1.3 PCA з використанням штучних нейронних мереж	16
1.1.4 Штучна бджолина колонія	16
1.1.5 Генетичний алгоритм	17
1.1.6 Оптимізація рою частинок	18
1.1.7 Deep Dense Face Detector	19
1.1.8 Radial Basis Function Neural Networks.....	19
1.1.9 Convolutional Neural Network Cascade	21
1.1.10 Bilinear CNNs	21
1.1.11 Multi-task Cascaded Convolutional Networks	22
1.2 Виявлення орієнтирів на обличчі.....	24
1.2.1 Значення лицьових орієнтирів	24
1.2.2 Проблеми виявлення орієнтирів на обличчі.....	25
1.2.3 Типи орієнтирів.....	26
1.2.4 Критерії оцінки обчислення орієнтирів обличчя	27
1.2.5 Попередня обробка для вилучення орієнтирів.....	28
1.2.6 Методи розмітки обличчя	29
1.2.7 Розмітка 3D-граней.....	30

1.3	Методи реконструкції 3D–моделі обличчя	31
1.4	Результат аналізу.....	33
1.4.1	Формулювання мети наукового дослідження	34
1.4.2	Перелік основних задач дослідження	34
1.4.3	Очікувані наукові та практичні результати роботи	35
1.5	Висновки до розділу	35
Розділ 2 Проектування архітектури		36
2.1	Вимоги до архітектури програмної системи	36
2.2	Методи розпізнавання обличчя.....	36
2.3	Методи розмітки обличчя.....	39
2.4	Визначення способу генерації моделі.....	39
2.5	Визначення методу зафарбування моделі	41
2.6	Висновки до розділу	42
Розділ 3 Реалізація програмного додатку		43
3.1	Засоби реалізації програмної системи	43
3.2	Середовище розробки та тестування	44
3.3	Опис формату збереження моделі	46
3.4	Реалізація програмної системи.....	47
3.5	Висновки до розділу	52
Розділ 4 Тестування та оцінювання результатів роботи нейронної мережі масштабування зображень.....		53
4.1	Тестування програмної системи.....	53
5.1	Оцінка роботи програми.....	60
5.2	Висновки до розділу	62
Висновки.....		63
Перелік використаних джерел.....		64
Додаток А Зауваження нормоконтролера.....		69
Додаток Б Лістинг основних програмних модулів.....		70
Додаток В Матеріали презентації		91

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І
ТЕРМІНІВ

CNN	Convolutional neural network
PCA	Principal component analysis
PSO	Particle Swarm Optimization
3DMM	3DMorphable model
RGB	Red green blue
CUDA	Compute Unified Device Architecture
MTCNN	Multi-task Cascaded Convolutional Networks
FCN	Fully Convolutional Network
GA	Genetic Algorithm
RBF	Radial Basis Functions Neural Networks
IOD	Inter-Ocular Distance
AAM	Active Appearance Model
ASM	Active Shape Model
HOG	Histogram Of Gradients
AI	Artificial Intelligence

ВСТУП

Тривимірна реконструкція обличчя – важливе завдання в області комп'ютерного зору і графіки. Відновлення тривимірної геометрії обличчя з зображення знаходить застосування в різних областях, це можуть бути ігри, доповнена реальність, і медицина.

Існуючі методи реконструкції обличчя, засновані на декількох зображеннях, дають хороші результати, але для однієї реконструкції моделі по одному зображенню це все ще проблема, особливо при часткових оклюзіях і нестандартних положеннях обличчя.

Для створення 3D-моделей також використовують 3D-лазерне сканування, завдяки чому, є можливість створити більш точний людський облік для проектування та оцінки продукту. Пристрій для сканування з високою роздільною здатністю може записувати величезну кількість докладної інформації про геометрію людського обличчя.

Фактором, що обмежує широке поширення систем обробки тривимірних зображень обличчя, є суттєві недоліки, пов'язані з сучасним рівнем технологій тривимірного сканування, ці недоліки включають:

- вимогу до контрольованих умов освітлення в процесі сканування;
- неможливість одночасного захоплення 3D-даних з декількох об'єктів;
- неточні дані, отримані з невідображаючих ділянок обличчя;
- нездатність працювати в режимі реального часу;
- у випадку 3D лазерних сканерів, використання лазерного світла під час процесу сканування може бути небезпечним для очей;
- вимоги до людського втручання під час процесу сканування і пост обробки;
- розмір і вартість 3D сканеру є обмежуючими факторами для використання в реальних додатках.

Існують різні методи реконструкції моделей, фокус може бути на швидкість реконструкції, точності підсумкової моделі та можливості побудови розподіленої системи.

Тривимірна реконструкція обличчя полягає у відновленні тривимірної геометрії обличчя в необмежених ситуаціях з двомірних зображень, що також є важливим завданням в області комп'ютерного зору і графіки. З розвитком комп'ютерного зору і машинного навчання з'явилися нові методи, для вирішення цього завдання.

Підходи до вирішення цієї проблеми можна розділити на дві категорії, а саме:

- метод на основі морфованної моделі (3DMM) [1];
- метод на основі згорткової нейронної мережі (CNN) [7].

З появою морфованної моделі стали активно розвиватися нові напрямки автоматичної генерації, реконструкції та анімації обличчя. Основна ідея використання морфованної моделі полягає в розкладанні довільної поверхні обличчя в лінійну комбінацію базисних обличь. За рахунок апроксимації довільні моделі зменшуються до необхідного розміру заздалегідь підготовленої бази, а висока деталізація вихідних моделей дозволяє синтезувати реалістичні обличчя. Підхід лінійної апроксимації вимагає високої обчислювальної потужності, тому застосовується навіть для генерації моделей в реальному часі [8] і дає реалістичні результати при побудові анімації обличчя.

Звичайний 3DMM вивчається з уже готового набору 3D – сканування обличь з відповідними якісними 2D–зображеннями обличь. Традиційно 3DMM вивчається під наглядом, виконуючи зменшення розмірів, зазвичай аналіз головних компонентів (PCA) на навчальному наборі спільно зроблених 3D сканувань обличчя і 2D–зображень. При використанні лінійної моделі, такої як PCA, нелінійні перетворення і лицьові варіації не можуть бути зафіксовані за допомогою 3D моделі. Для моделювання дуже мінливих 3D

форм необхідні великі обсяги високоякісних 3D даних обличч, однак виконання цієї вимоги є дорогим, оскільки сканування обличчя дуже трудомістке як на етапі збору даних, так і на етапі пост обробки.

Як згадувалося раніше, лінійна 3DMM має такі проблеми як:

- необхідність 3D-сканування обличчя для навчання з вчителем;
- неможливість використовувати масивні зображення осіб в природних умовах для навчання;
- обмежена потужність уявлення через лінійну модель (PCA). Как понимают «потужність» уявлення

Перша 3DMM [2] була побудована на основі сканування 200 суб'єктів схожою етнічною та віковою групами. Їх також знімали в добре контрольованих умовах, тільки з нейтральним виразом обличчя.

Отже, такі моделі вчаться зображати текстуру обличчя тільки в схожих, а не в природних умовах. Це істотно обмежує сценарії його застосування. Варіації обличчя мають нелінійний характер так як варіації в різних виразах обличчя або позах нелінійні, що порушує лінійне припущення моделей. Таким чином, модель PCA не може досить добре інтерпретувати лицьові варіації. Особливо це стосується текстури обличчя. Широко використовувана Базельська модель обличчя (BFM) [3] також побудована тільки з 200 об'єктів з нейтральним виразом обличчя.

Традиційні методи тривимірної реконструкції обличчя [3, 4] в основному засновані на алгоритмах оптимізації, наприклад, ітеративна найближча точка [4], щоб отримати коефіцієнти для моделі тривимірної трансформованою моделі (3DMM) і візуалізувати відповідні тривимірні обличчя з одного зображення обличчя [5]. Однак такі методи зазвичай вимагають багато часу через високу складність оптимізації і страждають від локальних оптимальних рішень і поганих ініціалізацій. Таким чином, в недавніх роботах пропонується використовувати CNN [7], щоб навчитися регресувати коефіцієнти 3DMM і значно поліпшити якість і ефективність реконструкції.

Після більш ніж десятиліття майже всі існуючі моделі використовують не більше 300 тренувальних сканувань. Таких невеликих навчальних наборів недостатньо для опису всього різноманіття людських обличч [6]. До недавнього часу, доклавши значних зусиль, а також розробивши новий автоматизований і надійний конвеєр для побудови моделей, який побудував першу великомасштабну 3DMM з сканованих зображень $\sim 10\,000$ об'єктів.

Текстурна модель зазвичай будується з невеликої кількості 2D-зображень обличч, захоплених разом з 3D скануванням, в добре контрольованих умовах. Незважаючи на те, що за останні кілька років пристрої 3D-зйомки значно покращилися, ці пристрої як і раніше не можуть працювати в природних умовах. Таким чином, всі поточні набори 3D-даних обличч були отримані в лабораторних умовах.

У 3DMM геометрія обличчя представлена як вектор форми, $S = (x_1, y_1, z_1, \dots, x_n, y_n, z_n)^T R^{3n}$, який містить координати x, y, z його n вершин. Для простоти передбачається, що кількість допустимих значень текстури в карті текстури дорівнює кількості вершин, отже, можна використовувати вектор текстури $T = (R_1, G_1, B_1, \dots, R_n, G_n, B_n)^{TR3}$ для подання текстури обличчя, в якій значення кольорів R, G і B можуть бути побудовані, з використанням набору даних зразкових граней, кожен M представлений своїм вектором форми S_i і вектором текстури T_i : $M = (S, T)$. Тривимірне обличчя можна описати за допомогою базового перетворення аналізу головних компонентів (PCA):

$$\begin{aligned} T &= \bar{T} + A_t a_t \\ S &= \bar{S} + A_s a_s \end{aligned}$$

де S – тривимірне обличчя, \bar{S} – середня форма, A_s – основа принципу форми, навчена на тривимірному скануванні обличчя з нейронним виразом, і a_s як коефіцієнт подання форми; T – це середня текстура, A_t – це основа принципу текстури, навчена на 3D сканування обличчя з нейтральним виразом обличчя, a_t – коефіцієнт уявлення текстури.

РОЗДІЛ 1

АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз методів розпізнавання обличчя

1.1.1 На основі знань

Метод, заснований на знаннях, залежить від набору правил і заснований на людських знаннях для виявлення обличчя. Приклад: обличчя повинно мати ніс, очі і рот на певних відстанях і позиціях один щодо одного.

Проблемою цих методів є складність побудови відповідного набору правил. Якщо правила були занадто загальними або занадто докладними, могло бути багато помилкових спрацьовувань. Одного цього підходу недостатньо, і він не може знайти багато обличч на кількох зображеннях.

1.1.2 PCA

Аналіз основних компонентів (PCA) використовується для спрощення даних шляхом лінійного перетворення та формування нових координат з максимальним відхиленням. Це колекція векторів із власного обличчя. Крім того, він обчислює матрицю коваріації з кількох частин колекції навчальних зображень обличчя. Можливість вилучення цієї функції може бути використана для розпізнавання зображень обличчя.

Мета полягає в тому, щоб реалізувати систему (модель) для певного обличчя та відрізнити її від великої кількості збережених обличч із деякими варіаціями в реальному часі. Це дає ефективний спосіб знайти простір нижчих розмірів. Далі цей алгоритм можна розширити для розпізнавання статі обличчя або для інтерпретації виразів обличчя людини. Можливість вилучення цієї функції може бути використана для розпізнавання зображень обличчя.

Перевага цього підходу перед іншими системами розпізнавання обличч полягає в його простоті, швидкості та нечутливості до дрібних або поступових

змін на обличчі. Проблема обмежується файлами, які можна використовувати для розпізнавання обличчя. А саме, зображення повинні мати вертикальні фронтальні види людських облич [19]. Алгоритм PCA:

Двовимірне зображення обличчя можна представити як одновимірний вектор, об'єднавши кожен рядок (або стовпець) у довгий тонкий вектор. Припустимо, що є M векторів розміру $N = (\text{рядки зображення} \times \text{стовпці зображення})$, що представляють набір зображень із вибіркою, p_i представляють значення пікселів.

$$x_i = [p_i \dots p_N]^T, i = 1, \dots, M$$

Середнє значення зображень центрується шляхом віднімання середнього зображення з кожного вектора зображення. Нехай m представляє середнє зображення.

$$m = \frac{1}{M} \sum_{i=1}^M x_i$$

І нехай w_i визначається як середнє центроване зображення:

$$w_i = x_i - m$$

Мета полягає в тому, щоб знайти набір e_i , які мають найбільшу можливу проекцію на кожну з w_i . Треба знайти набір M ортонормованих векторів e_i , для яких величина:

$$\lambda_i = \frac{1}{M} \sum_{n=1}^M (e_i^T w_n)^2$$

максимізується з обмеженням ортонормованості:

$$e_l^T e_k = \delta_{lk}$$

Було показано, що e_i та λ_i задаються власними векторами та власними значеннями коваріаційної матриці:

$$C = WW^T$$

де W – матриця, що складається з векторів-стовпців w_i , розміщених поруч. Розмір C дорівнює $N \times N$, що може бути величезним. Наприклад, зображення розміром 64×64 створюють коваріаційну матрицю розміром 4096×4096 . Непрактично вирішувати власні вектори C безпосередньо. Загальна теорема лінійної алгебри стверджує, що вектори e_i і скаляри λ_i можна отримати шляхом розв’язання власних векторів і власних значень матриці W^TW $M \times M$. Нехай d_i та μ_i – власні вектори та власні значення W^TW відповідно.

$$W^TWd_i = \mu_i d_i$$

Помноживши ліворуч в обидві сторони на W

$$WW^T(Wd_i) = \mu_i(Wd_i)$$

це означає, що перші $M - I$ власних векторів e_i та власних значень λ_i WW^T задані Wd_i та μ_i відповідно. Wd_i потрібно нормалізувати, щоб дорівнювати e_i . Оскільки потрібно підсумовувати лише скінченну кількість векторів зображень M , ранг коваріаційної матриці не може перевищувати $M - I$.

Власні вектори, що відповідають відмінним від нуля власним значенням коваріаційної матриці, створюють ортонормований базис для підпростору, в якому більшість даних зображення можуть бути представлені з невеликою похибкою. Вектори сортуються від високого до низького згідно з їх відповідних значень. Вектор, пов’язаний з найбільшим значенням, є таким, який відображає найбільшу дисперсію в зображенні. Тобто найменше власне значення пов’язане з власним вектором, який знаходить найменшу дисперсію. Вони зменшуються експоненціально, що означає, що приблизно 90% загальної дисперсії міститься в перших 5 – 10% вимірах.

Зображення обличчя можна спроектувати на розміри M_0 за допомогою обчислень:

$$\Omega = [v_1, v_2 \dots v_{M'}]^T$$

де $v_i = e_i^T w_i$. v_i – як координата зображення обличчя в новому просторі, який став основним компонентом. Вектори e_i також є зображеннями, так званими власними зображеннями, або власними [20]. Отже, Ω описує внесок кожної власної грані у відображення зображення обличчя, розглядаючи власні грані як базовий набір для зображень обличчя. Найпростіший метод визначення, який клас обличчя забезпечує найкращий опис вхідного зображення обличчя, це знайти клас обличчя k , який мінімізує евклідову відстань.

$$e_k = ||\Omega - \Omega_k||$$

де Ω_k – вектор, що описує k -ий клас граней. Якщо e_k менше за деякий попередньо визначений поріг θ_e , грань класифікується як належний до класу k .

Як тільки власні грані будуть розраховані, можна ухвалити кілька типів рішень від програми.

Те, що визначається під розпізнаванням обличчя, є широким терміном, який може включати такі завдання:

- ідентифікація, для отримання індивідуальних міток обличчя;
- реідентифікація;
- категоризація, віднесення обличчя до певного класу.

РСА обчислює основу простору, яка представлена його навчальними векторами. Ці базові вектори, фактично власні вектори, обчислені РСА, знаходяться в напрямку найбільшої дисперсії навчальних векторів. Кожне власне обличчя можна переглянути як функцію. Коли конкретне обличчя проектується на простір обличчя, його вектор у простір обличчя описує

важливість кожної з цих ознак обличчя. Грань виражається в просторі граней за допомогою коефіцієнтів (або ваг) власних граней.

Можна обробляти великий вхідний вектор, зображення обличчя, лише взявши його малий вектор ваги в просторі обличчя. Це означає, що можна відновити вихідне обличчя з деякою помилкою, оскільки розмірність простору зображення набагато більша, ніж простору обличчя. Розглянуто лише ідентифікацію обличчя. Кожне обличчя в навчальному наборі перетворюється в простір обличч, а його компоненти зберігаються в пам'яті. Простір обличч має бути заповнений цими відомими гранями. Вхідна грань надається системі, а потім проектується на простір граней. Система обчислює відстань від усіх збережених граней.

1.1.3 PCA з використанням штучних нейронних мереж

PCA це метод зменшення розмірності заснований на перетворенні Карунена-Лоєва. Метою даного методу є значне зменшення розмірності простору, для кращого опису типових рис обличчя. У задачах для розпізнавання обличчя його використовують для представлення зображення обличчя у вигляді вектору малої розмірності.

Застосовуючи цей метод можна визначити особливості навчальної вибірки зображень та описати їх за допомогою ортогональної проекції. В даному методі для кожної людини використовуються своя нейронна мережа, що і є його недоліком, але сам метод має досить високу точність.

1.1.4 Штучна бджолина колонія

Штучна бджолина колонія (ABC) – це один із алгоритмів на основі роїв, розроблених з урахуванням дій медоносних бджіл. Чотири компоненти поведінкової моделі ABC – це в основному джерело їжі, бджоли-розвідники, бджоли-спостерігачі та бджоли-найманці. Джерело їжі позначає можливе вирішення проблеми кластеризації, оскільки здійснюється глобальний пошук. Цей пошук виконується стохастично, тоді як спостерігач і бджола шукають сусідні рішення.

Застосовані бджоли згодом оцінюють точність розчину з попередньо збережених у пам'яті розчинів. Ця інформація послідовно передається бджолам-глядачам у зоні танцю. Це гарантує, що вибрано найкраще джерело їжі, а застійні джерела їжі в рамках вже встановленого циклу залишаються і замінюються новими джерелами. Процес повторюється доки не буде зближення для отримання оптимального рішення [21]

1.1.5 Генетичний алгоритм

Генетичний алгоритм (GA), з іншого боку, заснований на генетиці та теорії природного відбору. Це стохастичний алгоритм, який знаходить найкраще рішення, ефективно знаходячи глобальний оптимум у більшому просторі. Невід'ємне значення пристосованості отримується за допомогою функції пристосованості. Це значення використовується для підсумовування того, наскільки оптимальне рішення є близьким до найкращого у світі [22].

Генетичний алгоритм починається з генерування випадкових чисел (званих хромосомами) з розміром популяції n . Кожна хромосома має обчислене значення придатності, і перевіряється її критерій зупинки. Додатково пояснюються такі оператори GA, як відбір, кросовер і мутація, щоб підштовхнути хромосоми до конвергенції.

Оператор виділення створює потомство з існуючої популяції за допомогою процесу, порівнянного з природним відбором у біологічних формах життя. Відбір знову акцентує увагу на кращих показниках окремих людей у популяції. Це сприяє очікуванню того, що їх потомство матиме ймовірність перенесення генетичної інформації наступному поколінню. На конвергенцію сильно впливає масштаб процесу відбору. Критерії відбору повинні запобігати передчасній конвергенції, підтримуючи різноманітність популяції та баланс з операціями схрещування та мутації.

Оператор кросовера змішує інформацію між двома батьками таким чином, що відповідає статевому розмноженню. Метою процедури

схрещування є “народження” покращеного потомства. Це досягається шляхом вивчення різних частин простору пошуку.

Процедура мутації змінює значення випадково вибраного біта в кожному рядку, таким чином запобігаючи застряганню GA на локальному мінімумі через розсіювання генетичних даних, таким чином підтримуючи варіацію в популяції. Цей процес повторюється до тих пір, поки не буде досягнуто оптимальне рішення або не пройде задана кількість поколінь.

1.1.6 Оптимізація рою частинок

Оптимізація рою частинок (PSO) також є алгоритмом оптимізації, на який впливає біологія. Він був отриманий шляхом спостереження за колективною поведінкою та роїнням зграї птахів і риб [23]. Алгоритм складається з рішень, відомих як сукупність, кожне з яких має ряд параметрів, які представляють координати в просторі з кількома вимірами. Крім того, сукупність цих частинок перетворюється на сукупність частинок, які пробують простір пошуку, щоб знайти оптимальне рішення. Кожна частинка відстежує своє колишнє оптимальне рішення в пам'яті, а потім позначає ці рішення як особисті найкращі та глобальні найкращі. Тоді геометричне місце i -ї частинки визначається в D -багатомірному просторі як:

$$x_i = [x_{i1}, x_{i2}, x_{i3}, \dots, x_{iD}]$$

а популяція рою як:

$$X = [x_1, x_2, x_3, \dots, x_N]$$

Частинки потім ітеративно оновлюють свої відповідні положення в просторі параметрів під час пошуку оптимального рішення за допомогою:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

де v_i – компоненти швидкості i -ої частинки вздовж D виміру з t і $t+1$, що вказує на подвійний послідовний хід процесу. Швидкість i -ої частинки

визначається в наступному рівнянні трьома доданками: перший – це інерція, яка запобігає різкій зміні напрямку частинок, другий доданок описує здатність частинок повертатися в раніше відоме найкраще положення, а останній член описує частинки, які рухаються (рій) ближче до найкращого положення:

$$v_i(t + 1) = v_i(t) + c_1 (p_i - x_i(t)) R_1 + c_2 (g - x_i(t)) R_2$$

де p_i – особистий найкращий показник частки, g – глобальний найкращий, а c_1 та c_2 , в діапазоні $0 \leq 1$, $c_2 \leq 4$ – когнітивний та соціальний коефіцієнти відповідно. Нарешті, R_1 та R_2 – це дві діагональні матриці, випадково згенеровані з рівномірного розподілу в $[0,1]$. Це гарантує, що соціальний та когнітивний компоненти мають випадковий вплив на оновлення швидкості. Оскільки частинки є похідними від зближення особистого та глобального найкращих рішень, стохастичний вага двох прискорюючих членів і траєкторій є напіввипадковими. Це вимагає, щоб рівняння повторювалися до тих пір, поки не буде виконано критерій зупинки.

1.1.7 Deep Dense Face Detector

Deep Dense Face Detector (DDFD) не вимагає анотації пози/орієнтиру і здатний виявляти обличчя в широкому діапазоні орієнтацій за допомогою однієї моделі. Запропонований метод має мінімальну складність, оскільки на відміну від останніх методів виявлення об'єктів глибокого навчання, таких як R-CNN BB [24], він не потребує додаткових компонентів для сегментації, регресії обмеженої рамки або класифікаторів SVM. У порівнянні з попередніми згортковими детекторами обличчя на основі нейронної мережі, такими як у [25], використовувана мережа глибша і навчається на значно більшому навчальному наборі [26].

1.1.8 Radial Basis Function Neural Networks

Штучна нейронна мережа з радіальною базовою функцією є трирівнева нейронна мережа, що складається з вхідного слою, єдиного прихованого слою, що містить нейрони RBF [27], і вихідного шару, що містить лінійні або

нелінійні нейрони. Передавальна функція прихованих нейронів RBF – це ослаблення, центрально-радіальна симетрія, невід’ємна, нелінійна функція. Усі функції, які залежать тільки від відстані центрального вектору, радіально симетричні щодо цього вектору. Лінійна передатна функція вихідного шару використовується для апроксимації задач функцій. У цьому типі архітектури кожен нейрон підключений до всіх нейронів на наступному рівні, але немає жодних зв’язків між нейронами на тому ж рівні.

Інформація, що передається на вхідний рівень, який не виконує передавальну функцію для вхідного сигналу, а передає тільки вхідні дані в мережу з навколишнього середовища, поширюється через прихований шар у прямому напрямку до його вихідного рівня. Інформація, що передається з лінійного вхідного шару в прихований нелінійний шар, обробляється таким чином, що вихідний сигнал кожного прихованого нейрона назад пропорційний евклідовій відстані від вхідного вектора до центру RBF цього нейрона. У разі функції активації за Гауссом ваги нейронів прихованого шару є центром симетричної гаусової кривої дзвона. Ці ваги заздалегідь визначені таким чином, що весь вхідний простір покривається полем функцій активації Гаусса, а ваги синоптичних зв’язків між прихованими і вихідними нейронами шукаються в процесі навчання мережі. Вихідний шар обчислює суму терезів і зміщень всіх вихідних значень RBF.

Однак під час навчання всі входи RBF-ANN подаються на прихований рівень без будь-якої ваги, і тільки ваги між прихованим і вихідним шарами повинні бути змінені з використанням сигналу помилки. Отже, RBF-ANN вимагає набагато коротшого часу навчання проти багат шаровими ANN з прямим зворотним поширенням. Базова архітектура радіальної базисної функціональної мережі приведено на рисунку 1.1.

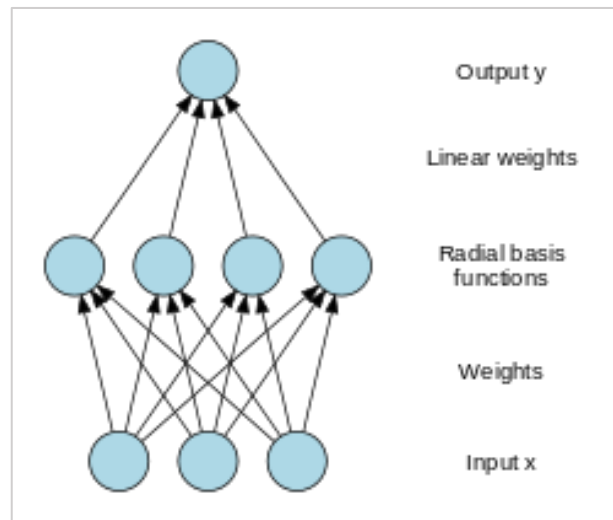


Рисунок 1.1 – Архітектура радіальної базисної функціональної мережі[27]

1.1.9 Convolutional Neural Network Cascade

У роботі [46] запропонували згорткову нейронну мережу «Cascade» для розпізнання обличч. Цей метод має потужні здатності розділення ознак, зберігаючи при цьому високу продуктивність. Запропонований CNN швидко відкидає фонові області на швидких етапах з низьким дозволом, і ретельно оцінює невелику кількість кандидатів в останній стадії.

Для підвищення ефективності, а також для зменшення кількості кандидатів на більш пізніх етапах, автор ввів етап калібрування CNN. У результаті генерування калібрувальної сітки отримується точніша локалізація обличчя, використовуючи більші вікна сканування по меншій кількості параметрів [55].

1.1.10 Bilinear CNNs

Запропоновано білінійні моделі, архітектуру розпізнавання, яка складається з двох ознак, вихідні дані яких перемножуються за допомогою зовнішнього продукту в кожному місці зображення та об'єднуються для отримання дескриптора зображення. Ця архітектура може моделювати локальні попарні взаємодії функцій трансляційним інваріантним чином, що особливо корисно для дрібнозернистої категоризації. Він також узагальнює

різні безпорядкові дескриптори текстур, такі як вектор Фішера, VLAD і O2P. Представляються експерименти з білінійними моделями, де екстрактори ознак засновані на згорткових нейронних мережах. Дволінійна форма спрощує обчислення градієнта і дозволяє наскрізне навчання обох мереж лише за допомогою міток зображень.

Результати показують [28], що архітектура вигідно порівнюється з існуючим рівнем техніки на ряді дрібнозернистих наборів даних, водночас є значно простішою та легшою для навчання.

1.1.11 Multi-task Cascaded Convolutional Networks

Каскадна загорткова мережа (MTCNN) розроблена для виявлення обличчя та вирівнювання обличчя. Процес виявлення складається з трьох етапів згорткових мереж, які розпізнають обличчя та її орієнтири, такі як рот, ніс та очі.

MTCNN пропонується як спосіб інтеграції обох завдань (розпізнавання та вирівнювання) за допомогою багатозадачного навчання. На першому етапі він використовує неглибокий CNN для швидкого створення вікон–кандидатів. На другому етапі він уточнює запропоновані вікна–кандидати за допомогою більш складного CNN. На третьому етапі використовується більш складний, ніж інші, для отримання уточнення результату та виведення орієнтирів обличчя.

Три етапи MTCNN:

Етап 1: Мережа пропозицій (P-Net).

Цей етап є повністю згортковою мережею (FCN). Різниця між CNN і FCN полягає в тому, що у повністю згорткової мережі не використовується щільний шар в архітектурі. Ця мережа пропозицій необхідна для отримання вікон–кандидатів та їх векторів регресії обмежувальної рамки.

Регресія обмежувальної рамки – це популярний метод прогнозування локалізації блоків, коли метою є виявлення об'єкта певного попередньо визначеного класу, у цьому випадку граней. Після отримання векторів

обмежувальної рамки виконується деяке уточнення, щоб об'єднати області, що перекриваються. Підсумковим результатом етапу є всі вікна кандидатів після уточнення, що зменшує кількість кандидатів.

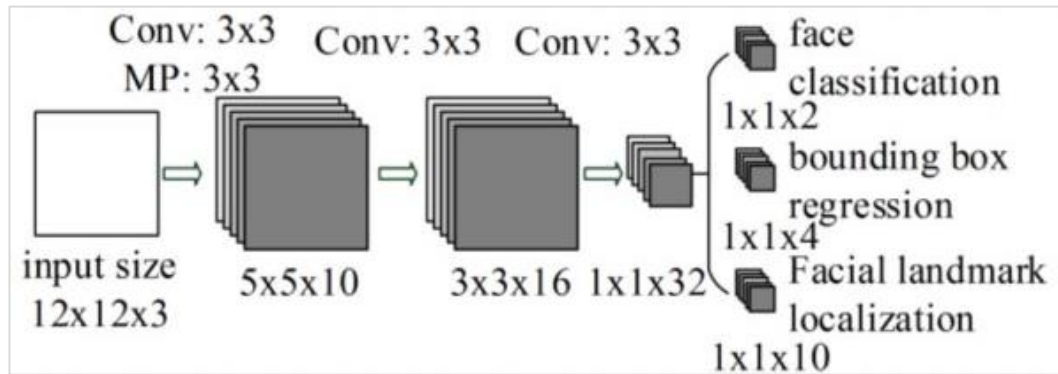


Рисунок 1.2 – Мережа пропозицій (P-Net) [29]

Етап 2: Мережа для уточнення (R-Net)

Усі знайдені кандидати з попереднього етапу надходять у наступну мережу. R-Net ще більше зменшує кількість кандидатів, виконує калібрування з регресією обмежувальної рамки та використовує немаксимальне придушення (NMS) для злиття перекривання кандидатів зменшуючи кількість обличч.

Поточна мережа (R-Net) виводить, незалежно від того, чи є вхід обличчям чи ні, вектор із 4 елементів, який є обмежувальною рамкою для обличчя, і вектор із 10 елементів для локалізації орієнтира обличчя.

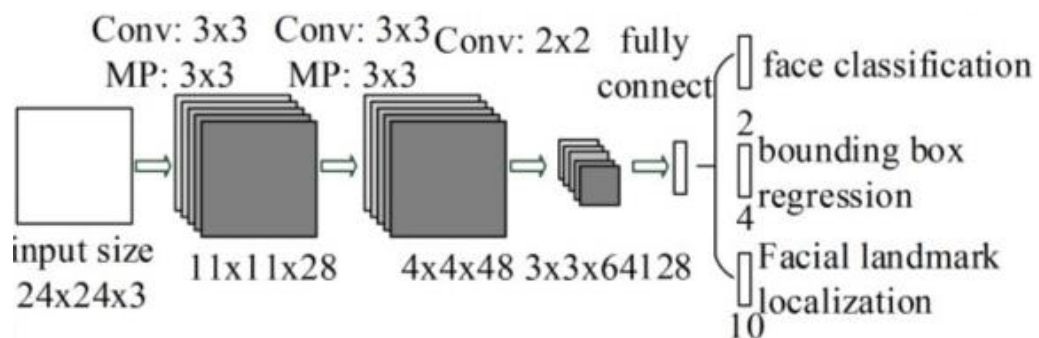


Рисунок 1.3 – Мережа пропозицій (R-Net) [29]

Етап 3: Вихідна мережа (O-Net)

Етап вихідної мережі подібний до R-Net, але ця вихідна мережа більш детально описує обличчя та повернути п'ять орієнтирів обличчя для очей, рота та носа.

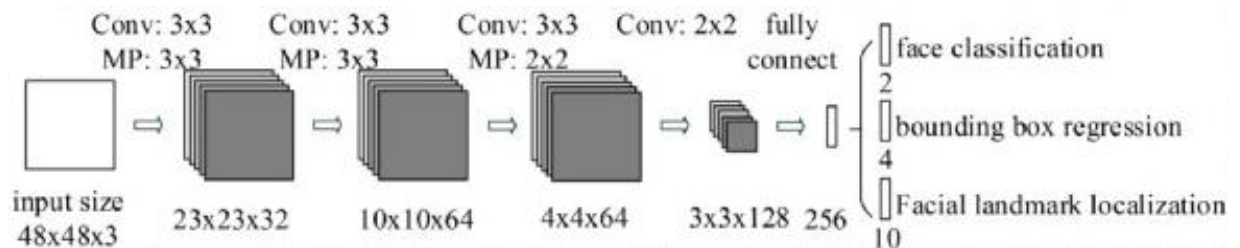


Рисунок 1.4 – Вихідна мережа (O-Net) [29]

1.2 Виявлення орієнтирів на обличчі

1.2.1 Значення лицьових орієнтирів

Орієнтири обличчя – це візуально помітні точки на обличчі, такі як кінець носа, кінці брів і рота. Парні позиції кожного з двох орієнтирів або локальна текстура орієнтира використовується як вектор ознак для розпізнавання емоцій обличчя. Загалом підходи для знаходження орієнтирів обличчя можна класифікувати на чотири типи відповідно до способів генерування таких моделей: модель на основі фігури (ASM), модель на основі зовнішнього вигляду (AAM), модель на основі регресії з комбінацією локальних і глобальних моделей і моделі на основі згорткових нейронних мереж [30].

Точне орієнтування обличчя та визначення рис обличчя є важливими операціями, які впливають на завдання, які зосереджені на обличчі, такі як: кодування, розпізнавання обличчя, вираз і/або розуміння жестів, виявлення погляду, анімація, відстеження обличчя тощо. Визначається орієнтир обличчя як помітна ознака, яка може відігравати дискримінаційну роль або служити опорними точками на графіку граней. Зазвичай вживаними орієнтирами є

куточки очей, кінчик носа, куточки ніздрів, куточки рота, кінцеві точки надбрівних дуг, мочки вух, носа, підборіддя тощо.

Вважається за краще використовувати термін *лицьовий компонент* для позначення всієї семантичної області обличчя, наприклад, вся область ока або очей, область носа, рота, підборіддя, щік або брів. Відомо, що на такі орієнтири, як куточки очей або кінчик носа, вираз обличчя мало впливає, тому вони більш надійні і насправді називаються *опорними точками*. Довідкові точки в системах зображень відносяться до позначок, навмисно розміщених у сцені, щоб функціонувати як орієнтир або міра. У розширенні, відносно стабільних або надійних орієнтирів на обличчі, такі як куточки очей або кінчик носа, також називаються *опорними точками* або *орієнтирами* з обробки обличчя в літературі. Моделі орієнтирів обличчя – це моделі, які навчаються за зовнішнім виглядом. Спочатку модель отримує приблизну грубу ініціалізацію. Потім початкову форму переміщують в кращу позицію крок за кроком до повного зближення з правильним варіантом.

1.2.2 Проблеми виявлення орієнтирів на обличчі

Є фактори, які погіршують ефективність визначення орієнтирів на обличчі: Зовнішній вигляд орієнтирів відрізняється через внутрішні фактори, такі як мінливість обличчя між людьми, а також через зовнішні фактори, такі як часткова оклюзія, освітлення, вираз обличчя, поза та роздільна здатність камери.

Орієнтири на обличчі іноді можуть спостерігатися лише частково через оклюзію волосся, рухи рук або самооклюзію або через інтенсивні обертання голови. Два інші основні варіанти, які ставлять під загрозу успіх виявлення орієнтирів – артефакти освітлення та вирази обличчя.

Алгоритм орієнтації обличчя, який добре працює під і поперек усіх внутрішніх варіацій граней, і який надає цільові точки ефективним часом, поки що неможливий. Як і у випадку з розпізнаванням обличчя, умови зйомки, такі як освітлення, роздільна здатність, безлад фону, можуть впливати на

продуктивність локалізації орієнтиру. Це підтверджується тим фактом, що орієнтири локалізації, навчені в одній базі даних, зазвичай мають нижчу продуктивність при тестуванні в іншій базі даних.

1.2.3 Типи орієнтирів

Орієнтири обличчя зручно розглядати у двох групах, позначених як довідкові та допоміжні, або первинні та вторинні орієнтири. Це дещо штучне розрізнення засноване на великій кількості та надійності зображень, які допомагають їх виявити. Наприклад, куточки очей, рота, кінчик носа, а іноді і брови можна відносно легко виявити за допомогою низькорівневих функцій зображення, таких як інформація про градієнт, кути або локальна інформація, наприклад, за допомогою інваріантного масштабного перетворення ознак. Гістограма градієнтів [32] та загальна інформація про морфологію обличчя. Ці безпосередньо виявлені орієнтири називаються основними або довідковими, і вони відіграють більш визначальну роль у ідентифікації та відстеження обличчя.

Орієнтири вторинної категорії, такі як ніздрі, підборіддя, носова частина, контури щок, точки без кінцівок на губах або середині брів, повіки тощо. Друга група орієнтирів відіграє більш помітну роль у виразі обличчя, хоча розмежування між цими двома завданнями не завжди є чітким. Основні та вторинні орієнтири, які найчастіше використовуються в літературі, показані на рисунку 1.5.

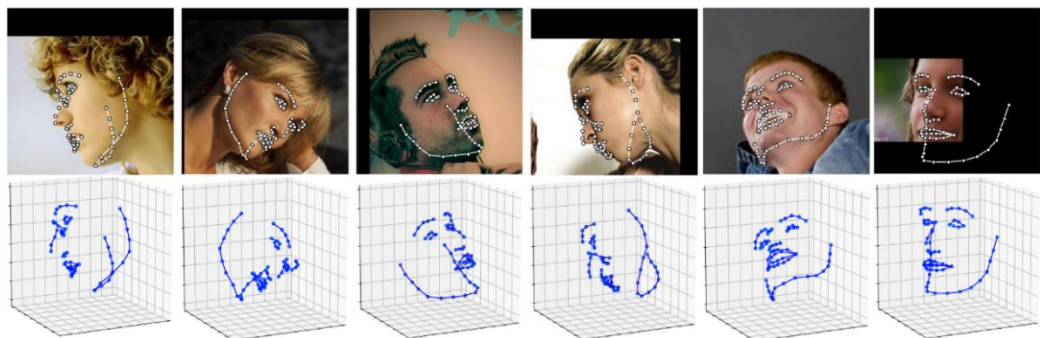


Рисунок 1.5 – 3D орієнтирі обличчя

1.2.4 Критерії оцінки обчислення орієнтирів обличчя

Можна визначити дві різні метрики для оцінки ефективності орієнтування:

- помилка локалізації на основі правдивої оцінки;
- виконання, орієнтоване на завдання.

Простий спосіб оцінити ефективність виявлення орієнтирів і локалізації орієнтирів – це використовувати ручні анотовані наземні істини. Якщо доступні позиції основної істини, ефективність локалізації може бути виражена в термінах нормалізованої середньоквадратичної помилки [33]. Середньоквадратичну помилку можна обчислити за орієнтиром, або дані можуть бути усереднені для всіх орієнтирів, щоб отримати глобальну точну цифру. Нормалізація, як правило, виконується щодо IOD – між очної відстані, яка визначається як відстань між двома очними центрами. Нормалізація помилок локалізації орієнтирів виконується шляхом поділу, IOD робить показник продуктивності незалежним від фактичного розміру обличчя або коефіцієнта масштабування камери.

Можна оголосити, що орієнтир буде виявлений, коли помилка локалізації залишається нижче відповідно вибраного порогу помилки. Похибки орієнтирів вважаються ізотропними, так що навколо кожного орієнтира наземної істини можна уявити коло виявлення з радіусом, рівним порогу помилки. Якщо евклідова відстань передбачуваного орієнтиру нижча за порогове значення, орієнтир вважається виявленим. Інакше, незалежно від значення помилки локалізації, вона оголошується як пропущений орієнтир.

Гарний спосіб проілюструвати ефективність виявлення – побудувати відсоток випадків виявлення певного орієнтира в межах заданого радіусу помилки. Точність локалізації, таким чином, обчислюється як евклідова відстань d між координатами наземної істини (x, y) і оціночними координатами (\tilde{x}, \tilde{y}) , нормалізованими за допомогою d_{norm} . Помилка визначається як:

$$\delta_i^k = \frac{d\{(x_i^k, y_i^k), (\tilde{x}_i^k, \tilde{y}_i^k)\}}{IOD}$$

де, верхній індекс k вказує на один з орієнтирів (наприклад, куточок ока, кінчик носа), а нижній індекс i – індекс зображення. Статистика виявлення орієнтирів може характеризуватися ймовірністю перевищення помилки локалізації. Загальна згода в літературі полягає в тому, що $\delta_i^k < 0.1$ є прийнятним критерієм помилки, тому орієнтир вважається виявленим, коли він знаходиться в межах однієї десятої між очної відстані від його справжнього положення. Точніше, розраховується продуктивність за орієнтиром:

$$P(k) = 100 \frac{\sum_{i=1}^I [i : \delta_i^k < Th]}{I}$$

де $[i : \delta_i^k < Th]$ – індикаторна функція, яка припускає 1, якщо відхилення менше порогового значення, інакше її значення дорівнює 0, а I позначає кількість тестових зображень. Загальна продуктивність усереднена для всіх типів орієнтирів:

$$P = 100 \frac{\sum_{k=1}^K \sum_{i=1}^I [i : \delta_i^k < Th]}{K \times I}$$

Цільовим заходом орієнтації може бути його вплив на виконання завдань. Деякі приклади цільових додатків, заснованих на орієнтації – це алгоритм реєстрації обличчя, класифікація виразів, або підгонка алгоритму активної моделі зовнішнього вигляду. Точність орієнтування щодо виконання алгоритмів реєстрації, класифікації виразів, розпізнавання жестів та підгонки ААМ, відповідно, буде цільовою мірою орієнтації.

1.2.5 Попередня обробка для вилучення орієнтирів

Завжди існує певна попередня обробка, перш ніж метод залучиться до виявлення орієнтирів. Типовими з цих кроків є такі: видалення артефактів

освітлення, скромні геометричні корекції, сегментація обличчя, використання інформації про колір.

1.2.6 Методи розмітки обличчя

Безліч методів орієнтування обличчя в літературі можна класифікувати різними способами, наприклад, на основі критеріїв типу або модальності спостережуваних даних (фотозображення, відео послідовність або 3D-дані), джерела інформації, що лежать в основі методології. (інтенсивність, текстура, карта країв, геометрична форма, конфігурація орієнтирів), а також попередня інформація (наприклад антропометричні дані), якщо така використовується.

Метою будь-якої спроби категоризації має бути краще розуміння спільності та відмінностей підходів, а також усвідомлення того, куди рухається сучасний стан. Незважаючи на труднощі пошуку чітких відмінностей, оскільки алгоритми часто використовують методи, спільні для кількох категорій, корисно класифікувати їх на основі типу використовуваної інформації та конкретної методології [53,54]. У попередній такій спробі [53] використовували п'ять категорій алгоритмів орієнтування на основі геометрії, кольору, зовнішнього вигляду, краю та руху.

Існують дві основні категорії методів виявлення орієнтирів на обличчі:

- методи на основі моделі;
- методи на основі текстур.

Методи на основі моделі, також відомі як методи на основі форми, розглядають зображення обличчя та ансамбль орієнтирів обличчя як цілісну форму. Вони вивчають «форми обличчя» з позначених навчальних зображень, а потім на етапі тестування намагаються підібрати правильну форму до невідомого обличчя. Друга категорія, методи на основі текстур, також відомі як методи без моделі, спрямовані на пошук кожного орієнтира обличчя або локальних груп орієнтирів незалежно, без керівництва моделі. У цих методах інформація про форму все ще може бути викликана, але на пізнішому етапі для перевірки.

Кожну з цих двох широких категорій методів орієнтації можна розділити на дві підкатегорії. Методи, засновані на моделі, можуть бути розділені як явні методи, основними прикладами яких є ASM і AAM, і як неявні методи, наприклад, алгоритми, що використовують нейронну мережу, застосовану до всього обличчя. Аналогічно, методи на основі текстур можна обговорювати в підкатегоріях методів на основі перетворення, наприклад, фільтри Габора або функції HOG, і методів на основі шаблонів.

1.2.7 Розмітка 3D-граней

Незважаючи на те, що більшість методологічних досягнень в області орієнтації обличчя було реалізовано на 2D – зображеннях, інтерес до обробки 3D-зображень обличчя швидко зростає через більш широку доступність 3D-камер, наприклад, сенсорного пристрою Kinect, еволюції 3D-телебачення та відео. Нещодавня оглядова стаття про тривимірний опис людського обличчя [86] простежує історію використання орієнтирів від анатомічних досліджень до естетичних проблем, від розпізнавання обличчя до антропометричних заходів для корекції обличчя. Анатомічні орієнтири, що використовуються в 3D, такі ж, як і в 2D-зображеннях, у той час як для орієнтування 2D-зображень використовуються функції рівня сірого, 3D отримує переваги від функцій кривизни поверхні. Звичайно, існують схеми, які одночасно користуються 2D-текстурою та інформацією про тривимірну кривизну, оскільки пристрої 3D-зображення також надають зареєстровані 2D-зображення. Однією з переваг 3D-орієнтування є те, що воно дає можливість використовувати альтернативні методи обробки орієнтирів, оскільки існує кілька способів представлення даних 3D-обличч. Наприклад, хмари точок, карти глибини, численні профілі, вокселі, кривизна та індекс форми [87] використовувалися для розпізнавання обличчя, і вони ще не повністю використані для орієнтування. Більш важливою перевагою є те, що потенційно можна пом'якшити деякі обмеження, які виникають у 2D-орієнтуванні. Треба пам'ятати, що 2D-орієнтування стає дуже чутливим до зміни пози понад 20° нахилу та/або

відхилення, і воно також страждає від ефектів освітлення. У цьому сенсі 3D–дані обличчя обіцяють заповнити прогалини в продуктивності за наявності сильного освітлення та змін у позі. Недоліком є те, що необроблені дані 3D–обробки вимагають значно більшої попередньої обробки в порівнянні з 2D. Наприклад, лицьову поверхню необхідно вирівняти, видалити шипи та розриви та заповнити проміжки.

1.3 Методи реконструкції 3D–моделі обличчя

Протягом багатьох років було зроблено багато спроб оцінити тривимірну поверхню обличчя, що відображається в одному огляді. Перш ніж перераховувати їх, важливо згадати останні методи з кількома зображеннями, які використовують набори зображень для реконструкції. Хоча ці методи проводять точні 3D–реконструкції, вони вимагають багатьох зображень з кількох джерел, щоб створити єдину 3D–форму обличчя, тоді як реконструкція обличчя яка проводиться з окремих зображень.

Статистичні подання фігур, такі як широко популярний 3DMM [1,2,3], використовують багато вирівняних 3D–форм обличчя, щоб дізнатись про розподіл тривимірних граней, представлених у вигляді великого розмірного підпростору. Кожна точка цього підпростору є вектором параметрів, що представляє геометрію обличчя, а іноді вираз і текстуру. Реконструкція виконується шляхом пошуку точки в цьому підпросторі, яка представляє грань, подібну до тієї, що є на вхідному зображенні. Ці методи не намагаються створити дискримінаційні геометрії обличчя, і справді, як вже згадувалося раніше, використовувались лише для розпізнавання обличчя за контрольованих параметрів.

CNN для регресії параметрів 3DMM для фотографій обличчя також визнає відсутність даних про навчання головним занепокоєнням, пропонується синтезувати навчальні грані з відомою геометрією шляхом вибірки з розподілу 3DMM. Цей підхід дає фотографії синтетичного вигляду, які можуть легко спричиняти проблеми з переобладнанням при навчанні великих мереж [9] і їх

оцінені форми не були показані більш надійними чи дискримінаційними, ніж інші методи.

Для того, щоб отримати правильну реконструкцію, деякі роблять тверді припущення щодо сцени та умов перегляду на вхідному зображенні. Наприклад, на основі методів затінення робиться припущення щодо джерел світла, відбивної здатності обличчя тощо. Інші замість цього використовують симетрію обличчя [10]. Припущення, зроблені ними та іншими, часто не виконуються на практиці, обмежуючи застосування цих методів контрольованими налаштуваннями.

Приклади методів, з роботи [11] модифікують тривимірну поверхню прикладів фігур обличчя, підганяючи їх до обличчя, що з'являється на вхідній фотографії. Ці методи надають перевагу стійкості до складних умов перегляду порівняно з детальною реконструкцією. Таким чином, вони використовувались лише для розпізнавання обличчя та для синтезу нових поглядів із невидимих поз.

Деякі методи реконструкції пристосовують 3D–поверхню до виявлених орієнтирів обличчя, а не безпосередньо до інтенсивності обличчя. Сюди входять методи, розроблені для відео, та підходи, засновані на CNN, вони більше зосереджуються на виявленні орієнтирів, ніж на 3D–оцінці фігури, тому не намагаються отримати детальні та чіткі геометрії обличчя. На рисунку 1.6 наведено основні функції при моделюванні обличчя.

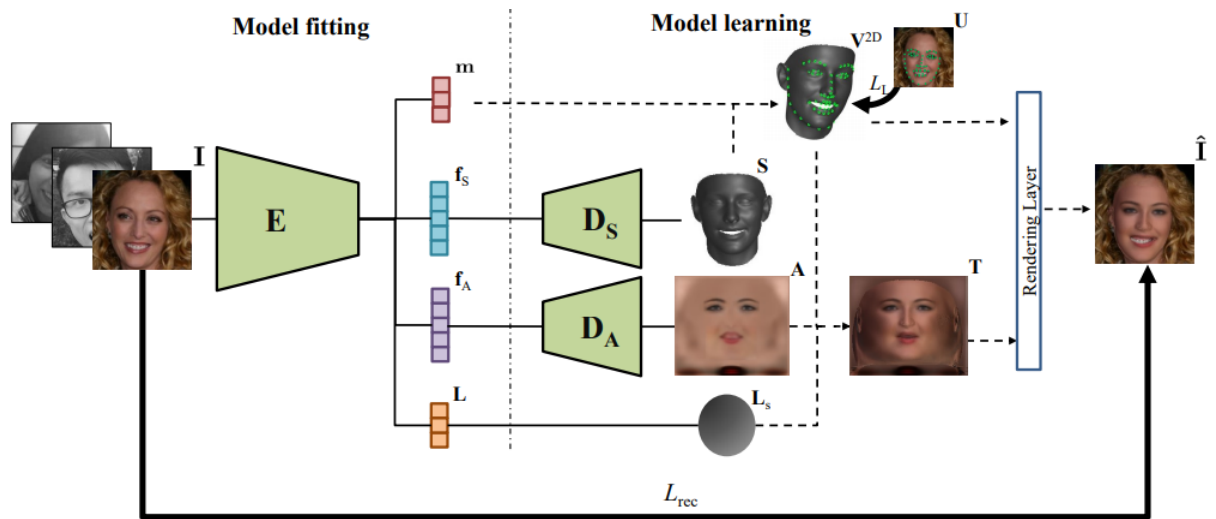


Рисунок 1.6 – Функції моделювання обличчя [18]

З моменту свого дебюту в 1999 році 3DMM стала новим методом досліджень в області аналізу обличчя і стала застосовуватися для вирішення багатьох завдань. Незважаючи на свій вплив, він має недоліки в тому, що потрібні навчальні дані для 3D-сканування, навчання на основі контрольованих 2D-зображень і обмежена потужність уявлення через лінійних основ як для форми, так і для текстури.

Ці недоліки можуть бути величезними при установці 3DMM на обличчя без обмежень або при вивченні 3DMM для звичайних об'єктів, таких як взуття. Було розглянуто альтернативний підхід до навчання 3DMM, при якому нелінійний 3DMM можуть бути вивчені з великого набору зображень обличчя в природних умовах без збору 3D-сканувань обличчя.

1.4 Результат аналізу

Процес генерації обличчя з фото є перекладом 2D-зображення в 3D – модель із збереженням наближених рис обличчя та кольору шкіри у згенерованій моделі.

Генерація обличчя з фото складається з кількох етапів:

- розпізнавання ділянок обличчя на зображенні;
- аналіз характеристик обличчя;

- синтез моделі на основі характеристик;
- текстурування моделі.

1.4.1 Формулювання мети наукового дослідження

Генерація обличчя складається з безлічі етапів, кожен з яких впливає на кінцевий результат і швидкість роботи. Використання існуючих методів або їх комбінація та вивчення різниці методів на кінцеві результати дозволить обрати оптимальну комбінацію методів для використання на кожному етапі для вирішення поставленого завдання.

Тим самим метою даного дослідження можна визначити реалізацію програми для синтезу 3D-моделі обличчя на основі зображення з використанням оптимальних методів для кожного етапу

1.4.2 Перелік основних задач дослідження

Для досягнення мети поставлені наступні завдання:

1. зробити аналіз існуючих методів, моделей та алгоритмів для:
 - ідентифікації обличь на зображенні;
 - вилучення інформації індивідуальних особливостей;
 - створення тривимірної моделі обличчя;
2. на основі аналізу розробити архітектуру та вимоги до системи для ідентифікації обличь на зображенні та створення тривимірної моделі обличчя;
3. створення тривимірної моделі обличчя за допомогою підходу 3D Morphable Model з удосконаленням PCA (метод головних компонент);
4. спроектувати програмні засоби ідентифікації та генерації структурної моделі обличчя.

1.4.3 Очікувані наукові та практичні результати роботи

У ході дослідження буде створено систему для моделювання обличчя з фотографії із використанням сучасних алгоритмів на різних етапах синтезу моделі.

Практична цінність полягає у створенні програми, яка дозволить генерувати 3д обличчя на основі однієї фотографії, без необхідності у спеціальному обладнанні або професійних навичок.

Таким чином, результатом дослідження буде додаток, який зможе заощадити час і ресурси для моделювання обличчя людини з однієї фотографії з подальшим експортом його в необхідні програми, такі як ігрові програми, або програми для моделювання.

1.5 Висновки до розділу

Проаналізовано існуючі методи розпізнавання обличчя, для вибраної задачі може підійти мережа MTCNN так як вона розроблена спеціально для розпізнавання обличчя та орієнтирів, вона дає хороші результати в тестах.

Розглянуто методи виявлення орієнтирів обличчя та типи орієнтирів обличчя, розглянуто недоліки та переваги кожного з них.

Розглянуто методи реконструкції моделі обличчя на основі 3DMM або CNN, розглянуто особливості кожного їх.

Метою даного дослідження можна визначити готову програму для моделювання обличчя на основі фотографії, визначення оптимальних алгоритмів для моделювання та рекомендації щодо використання.

Основним завданням дослідження є розробка системи моделювання обличчя, порівняння обраних методів з альтернативними. У ході дослідження буде розроблено програму, яка дозволить використовувати систему для моделювання обличчя від завантаженого файлу. У цьому розділі також формується головна мета дослідження, визначаються основні завдання виконання, критерії оцінки результатів та описуються очікувані результати та їхня практична цінність.

РОЗДІЛ 2

ПРОЕКТУВАННЯ АРХІТЕКТУРИ

2.1 Вимоги до архітектури програмної системи

Архітектура програмної системи має бути побудована таким чином, щоб кожен модуль був ізольований від іншого. Це дозволить легше супроводжувати та розширювати систему в майбутньому.

Загальні операції мають бути винесені до батьківського класу, це дозволить використовувати різні моделі осіб із винесенням у них специфічної логіки.

Модулі програми не повинні залежати від типу взаємодії користувача з програмою, це може бути графічним або консольним інтерфейсом.

2.2 Методи розпізнавання обличчя

Для розпізнавання обличчя на зображення вибрано метод з використанням багатозадачних каскадних мереж (MTCNN). MTCNN розроблена для розпізнавання та вирівнювання обличчя.

Завдання мережі полягає в тому, щоб вивести три речі: класифікацію обличчя/не обличчя, регресію обмежувальної рамки та локалізацію орієнтирів на обличчі.

1. класифікація граней: це проблема бінарної класифікації, яка використовує перехресні втрати ентропії;
2. регресія обмеженої рамки: метою навчання є проблема регресії. Для кожного вікна–кандидата обчислюється зміщення між кандидатом і найближчою основною істиною. Для цього завдання використовується евклідова втрата;
3. локалізація орієнтирів обличчя: локалізація орієнтирів обличчя формується як задача регресії, в якій функція втрат є евклідовою відстанню;

Процес складається з трьох етапів згорткових мереж, які можуть розпізнавати обличчя та орієнтири, такі як очі, ніс та рот. На першому етапі використовується неглибока CNN для швидкого створення вікон кандидатів. На другому етапі він уточнює пропоновані вікна кандидатів через складнішу CNN. На третьому етапі він використовує третю CNN, складнішу, ніж інші, для подальшого уточнення результату та виведення позицій орієнтирів на обличчі.

Розглянемо детальніше роботу цих трьох етапів.

При першому етапі (“Р-мережа”) зображення розбивається на менші розміри для пошуку осіб різних розмірів на зображенні, щоб отримати вікно–обличчя та їх вектори регресії обмежувальної рамки. Потім використовується оцінені вектори регресії обмеженої рамки для кандидатів. Після цього використовується не максимальне зменшення, щоб об’єднати кандидатів, які сильно перекриваються, на рисунку 2.1 та 2.2. наведено схему та приклад роботи першого етапу.

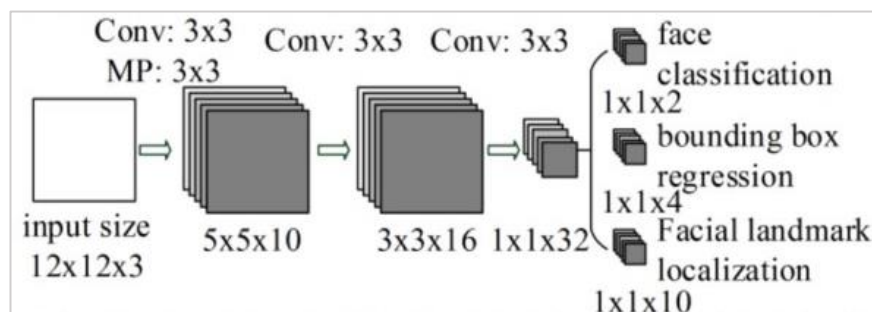


Рисунок 2.1 – Перший етап пошуку обличчя [29]



Рисунок 2.2 – Приклад пошуку обличчя різних розмірів

Підсумковим результатом цього етапу будуть координати прямокутників, що обмежують обличчя.

На другому етапі усі кандидати потрапляють до “R-мережі”. Зауважимо, що ця мережа є CNN, а не FCN, як попередня, оскільки на останньому етапі мережевої архітектури існує щільний рівень. R-Net додатково скорочує кількість кандидатів, виконує калібрування з регресією обмежуючого прямокутника і використовує подавлення без максимуму для злиття кандидатів, що перекриваються. На рисунку 2.3 наведено схему роботи другого етапу.

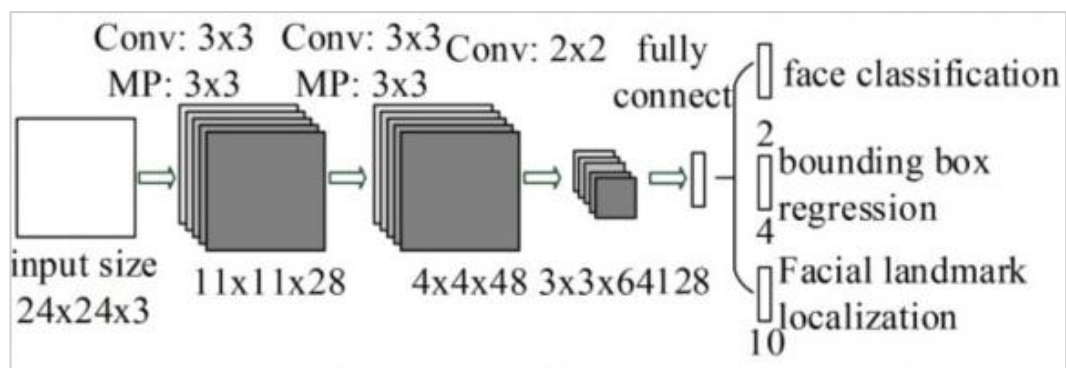


Рисунок 2.3 – Другий етап для зменшення кандидатів [29]

Третій етап аналогічний R-Net, але ця мережа націлена на більш докладний опис обличчя та виведення положень п'яти лицьових орієнтирів для очей, носа та рота. На рисунку 2.4 наведено схему роботи третього етапу.

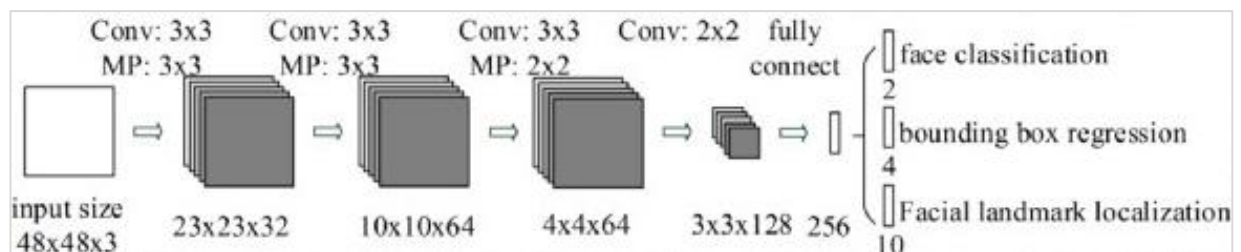


Рисунок 2.4 – Третій етап для зменшення кандидатів [29]

2.3 Методи розмітки обличчя

Для локалізації орієнтирів було обрано мережу 3D-FAN. З можливістю визначення орієнтирів для 2D і 3D обличч. Що має стійкість до пози, роздільної здатності, та кількості використовуваних мережевих параметрів.

Для подолання нестачі наборів даних для тривимірного вирівнювання осіб можливе перетворення двовимірної анотації в тривимірні та подальшого його використання для створення LS3D-W, найбільшого та найбільш складного набору даних тривимірних орієнтирів на сьогоднішній день (~230000 зображень).

2.4 Визначення способу генерації моделі

Для створення моделі обличчя підійде метод на основі 3D морфірованої моделі. 3d морфірованої моделі (3DMM) були запропоновані більше 20 років тому як статистична модель 3D-геометрії та текстури обличчя. Їх можна використовувати як генеративну модель двовимірного зовнішнього вигляду обличчя, з підбором параметрів форми та текстури з параметрами освітлення та камери, які надаються як вхідні дані для засобу візуалізації графіки.

Використання такої моделі у межах аналізу шляхом синтезу дозволяє принципово розділити чинники, що впливають появленню обличчя на зображенні.

3DMM зазвичай будуються із захоплених даних із використанням спеціального обладнання. Це потребує налаштування захоплення обличчя, у якій як тривимірна геометрія, а й власні властивості відображення обличчя, наприклад дифузне альбедо.

BFM було навчено 200 осіб (100 чоловіків, 100 жінок). Середній вік випробуваних - 25 років, сконцентровано у віці студентів, діапазон ваги відносно великий, середня вага 66 кг. На рисунку 2.5 наведено приклад характеристики людей сканування.

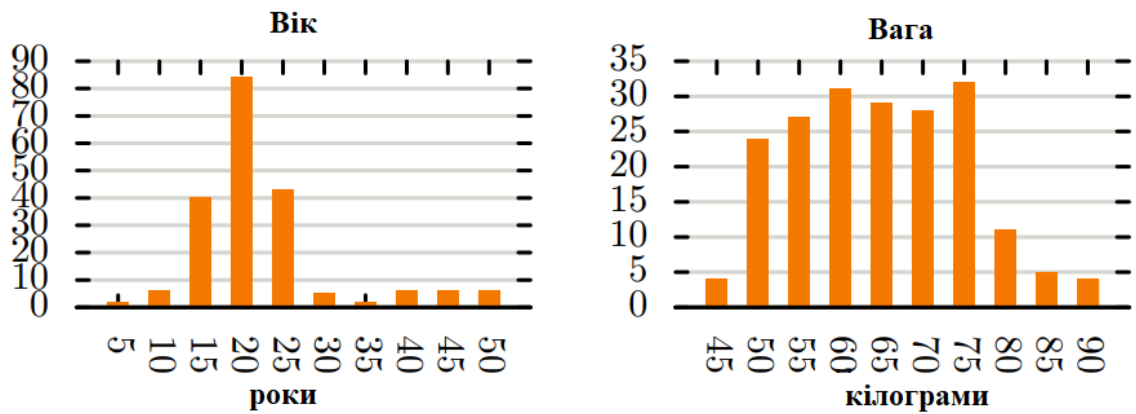


Рисунок 2.5 – Характеристика людей у наборі

Геометрія BFM складається з 53490 тривимірних вершин, з'єднаних 160470 трикутниками. Обличчя різних ідентичностей можуть бути складені як лінійні комбінації з 199 основних компонентів. Модель містить основні компоненти форми, варіації форми, середню текстуру та компоненти текстур.

Середнє значення та перші основні компоненти моделі форми та текстури представлені на рисунку 3.1 та 3.2.

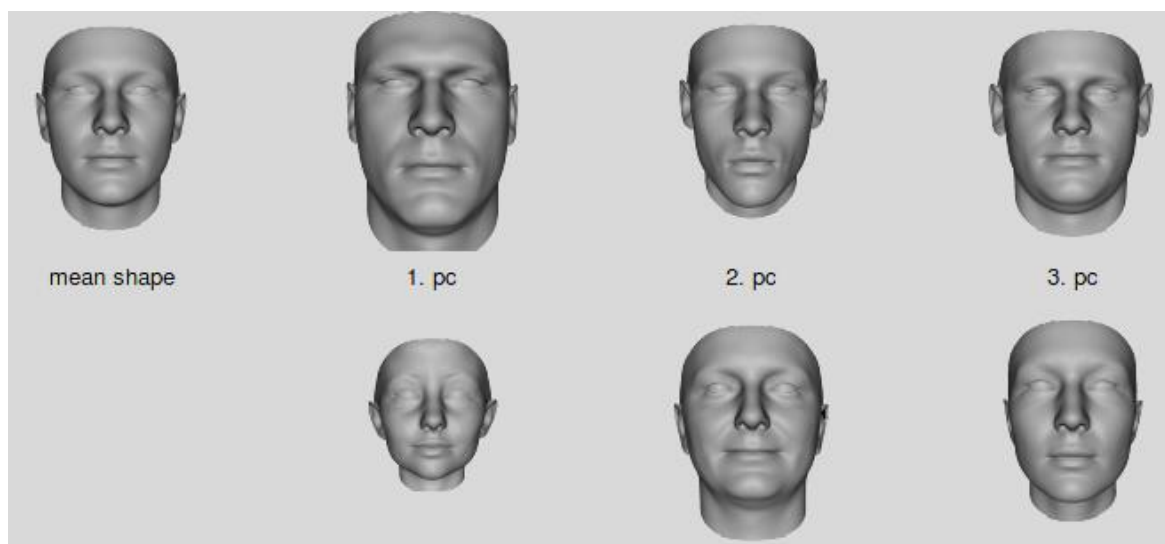


Рисунок 2.6 – Форми обличчя у BFM [47]

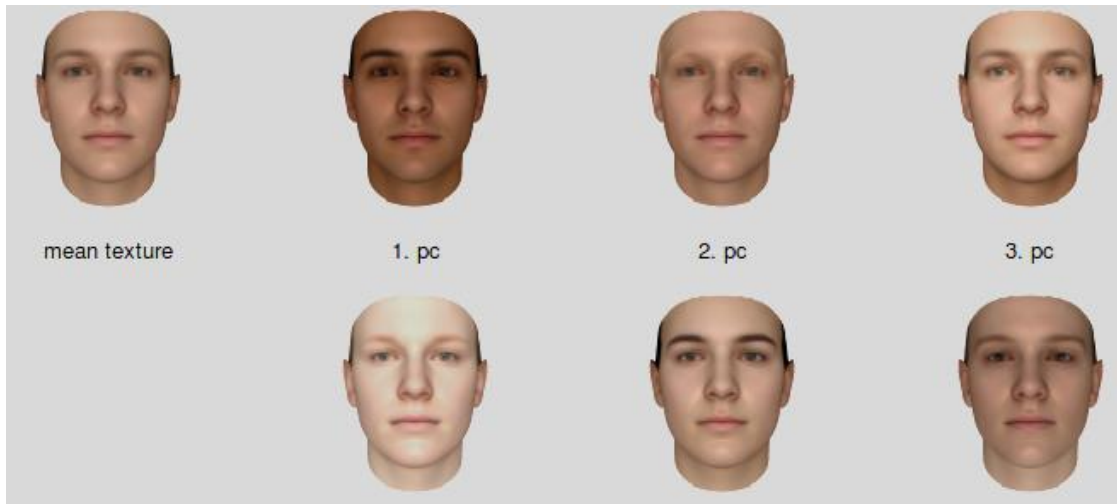


Рисунок 2.7 – Текстури обличчя у BFM [47]

BFM моделі обличчя також містить два набір орієнтирів, визначених відповідно до топології обличчя, приклад наведено на рисунку 2.8.

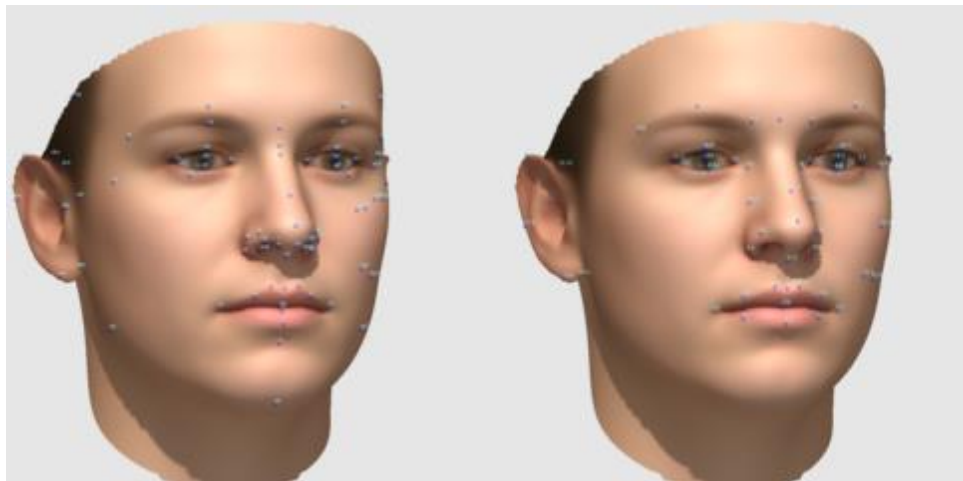


Рисунок 2.8 – Орієнтири обличчя [47]

2.5 Визначення методу зафарбування моделі

З урахуванням вибраного методу генерації моделі на основі морфірованої моделі, крім даних про форму моделі збережено також інформацію про текстуру, знімок якого зроблено в лабораторних умовах з рівномірним освітленням для всіх випробуваних.

Зафарблення обличчя складається з комбінацій вихідної текстури моделі, текстури аналізованої обличчя, та освітлення сцени з урахуванням гама корекції, навколишнього світла та дзеркальності)

2.6 Висновки до розділу

Визначено вимоги до архітектури програмної системи.

Вибрані методи для розпізнавання осіб та орієнтирів обличчя, визначений спосіб генерації моделі, з урахуванням обраної методу генерації моделі, визначені найбільш оптимальний метод розмальовки моделі.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ДОДАТКУ

3.1 Засоби реалізації програмної системи

Для розробки програмної системи було обрано мову програмування версії 3.8. Мова широко використовується для роботи з машинним навчанням. Python пропонує широкий вибір бібліотек для розробки штучного інтелекту. Це пов'язано з такими факторами:

- а) Python має широкий вибір бібліотек для розробки штучного інтелекту, які містять елементи базового рівня, які заощаджують час програмування, ці бібліотеки також спрощують доступ, обробку та перетворення даних;
- б) залежить від неймовірно складних обчислень та багатоетапних робочих процесів, тому чим менше розробнику потрібно турбуватися про складність кодування, тим більше він може зосередитися на пошуку відповідей на проблеми та досягнення цілей;
- в) Python має широкий набір бібліотек, і деякі з них пропонують чудові інструменти візуалізації, це дуже корисно в AI, оскільки включає подання даних у форматі, що легко зчитуються;
- г) підтримка роботи на різних платформах таких як Windows, Linux та macOS, тому потребує невеликих змін або взагалі не потребує їх. Платформи повністю сумісні з мовою програмування Python, а це означає, що фахівцю Python практично не потрібно змінювати код програми під платформу.

Оскільки Python є інтерпретованою скриптовою мовою, швидкість виконання її програм нижче компілюваних мов програмування, однак це не впливає на швидкість машинного навчання, оскільки в обчисленнях використовуються графічні процесори, оскільки більшість бібліотек

безпосередньо використовують архітектуру роботи з графічними процесорами, такі як OpenCL [36] або CUDA [34].

Додатково до стандартних бібліотек Python будуть використані наступні основні бібліотеки:

- а) facenet_pytorch [35] включає ефективну, готову до cuda реалізацію MTCNN;
- б) OpenCV [37] бібліотека з відкритим вихідним кодом, яка містить функції для комп'ютерного зору, такі як аналіз відео, аналіз відеоматеріалів CCTV та аналіз зображень;
- в) face_alignment [38] бібліотека виявлення лицьових орієнтирів, здатну виявляти точки як і 2D, і у 3D координатах, ґрунтується на глибокому навчанні;
- г) numpy [39] підтримка багатовимірних масивів; підтримка високорівневих математичних функцій, призначених до роботи з багатовимірними масивами;
- д) Pytorch [40] фреймворк машинного навчання, створений на основі Torch.

Оскільки основний програмний модуль перекладу створення моделі обличчя утримуватиметься в окремому ізольованому модулі, що не залежить від типу інтерфейсу з клієнтом, прийнято рішення розробити графічний та консольний інтерфейс для роботи з програми.

Графічний інтерфейс дозволить ілюструвати кроки алгоритму, налаштовувати та бачити налаштування генерації моделі за умовчанням. Консольний режим підійде для систем, де графічний інтерфейс не настільки важливий, також можлива розробка нових систем поверх нього, передаючи дозволені параметри скрипту.

3.2 Середовище розробки та тестування

Розробка та тестування програмної системи виконуються на робочій станції, характеристики якої наведено на рисунку 3.1.

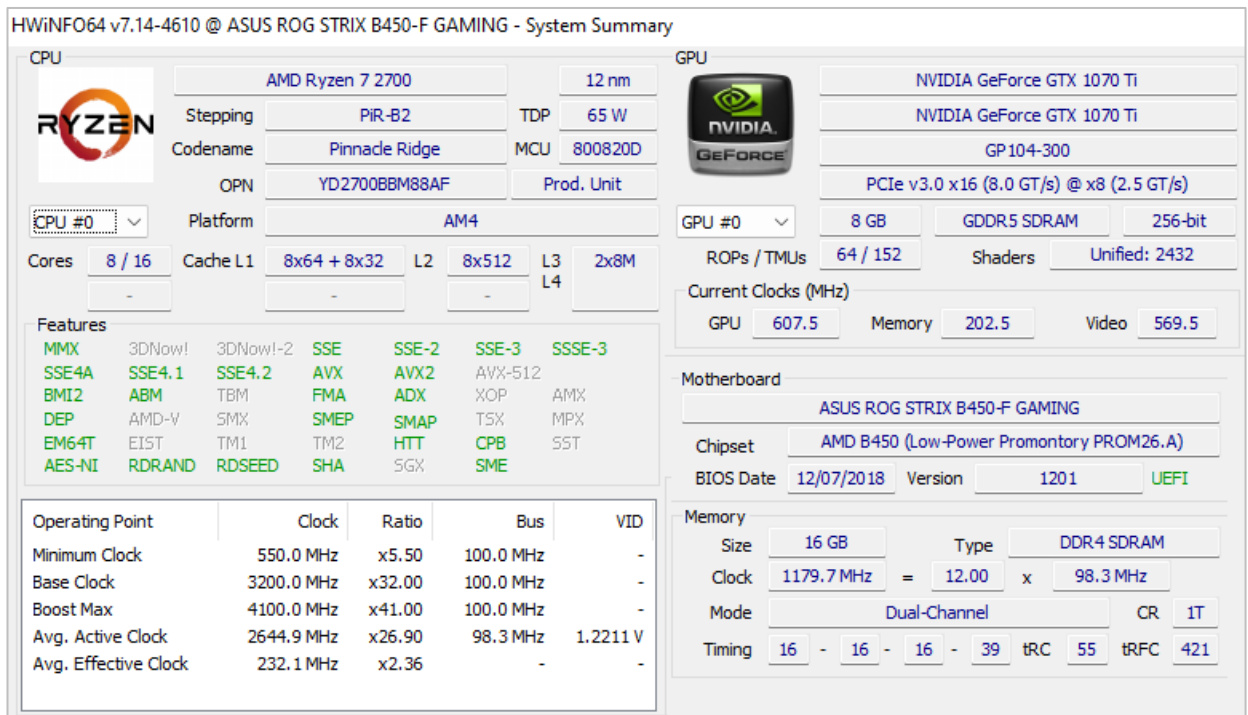


Рисунок 3.1 – Конфігурація робочої станції розробки

Для зручного керування установкою бібліотек, необхідних для роботи програми, використовується система керування середовищем для Python – Anaconda [41]. Вона дозволяє створити окреме середовище Python, яке міститиме всі зазначені бібліотеки. Таким чином, користувач повинен мати встановлений Anaconda і запустити в директорії проекту команду для встановлення всіх бібліотек необхідних проекту.

Для створення графічного інтерфейсу було вибрано PyQt[42]. PyQt - це набір інструментів для віджетів графічного інтерфейсу користувача (GUI). Його витягли з бібліотеки Qt. PyQt – це продукт комбінації мови Python та бібліотеки Qt. Основні переваги обраної бібліотеки:

- програмування GUI за допомогою Qt побудовано навколо ідеї сигналів і слотів для створення контакту між об'єктами, це забезпечує універсальність у роботі з інцидентами GUI, що призводить до більш гладкої бази коду;
- використовується широкий спектр власних API платформи для створення мереж, розробки баз даних тощо, він забезпечує

первинний доступ до них через спеціальний API;

- Qt надає кілька віджетів, таких як кнопки або меню, розроблені з базовим інтерфейсом для всіх сумісних платформ;
- є однією з найбільш часто використовуваних систем інтерфейсу користувача для Python, ви можете легко отримати доступ до різноманітної документації.

Розроблений графічний макет у програмі Qt Designer [43] представлений рисунку 3.2.

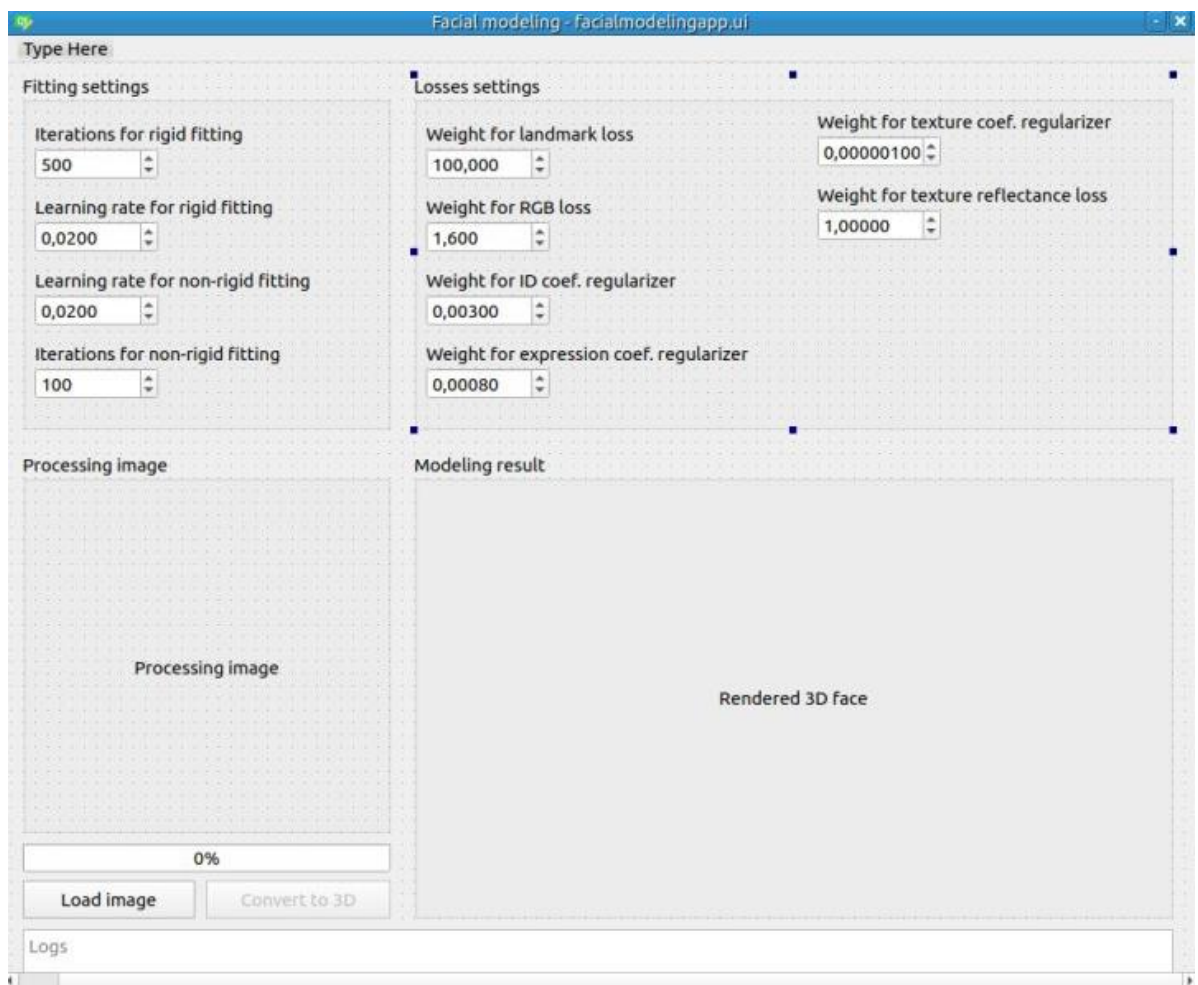


Рисунок 3.2 – Розроблений графічний макет у програмі Qt Designer

3.3 Опис формату збереження моделі

В результаті виконання програмна система створює директорію для кожної моделі, директорія містить такі файли:

- зображення містить згенероване обличчя;
- моделі обличчя у obj форматі;
- передані налаштування програмній системі;
- оригінальне зображення обличчя.

Формат OBJ є одним із найпопулярніших форматів передачі тривимірної комп'ютерної геометрії.

Для збереження зовнішнього вигляду моделі (матеріалу) кожна вершина збереже значення кольору в RGB моделі, перегляд моделі можливий через програми для роботи з 3д моделями, прикладом MeshLab [44].

3.4 Реалізація програмної системи

Модулі програм виконують єдине завдання не порушуючи принцип відповідальності в проектуванні програми, завдяки чому зручне їхнє супроводження або заміна модулів.

Запуск програми можливий у графічному та консольному варіанті для запуску необхідно використовувати відповідні файли.

Для запуску в консольному режимі необхідно запустити файл «image_to_3d.py», для запуску в графічному режимі необхідно запустити файл «arr.py», обидва файли використовують єдиний модуль та логіку для обробки зображення.

Програма має підтримку передачі параметрів моделювання через консольний або графічний режим через поле введення.

Основні модулі програмної системи наведені у таблиці 3.1.

Таблиця 4.1 – Модулі програмної системи

Модуль	Призначення
FaceModel.py	Базовий клас моделі обличчя зберігає основні обчислення для моделі, успадковує клас з бібліотеки

Модуль	Призначення
	torch для роботи з графіком, всі моделі для моделювання обличчя повинні успадкувати цей клас.
BFMFaceModel.py	Клас для роботи з BFM моделлю обличчя, успадковує клас FaceModel і зберігає операції, що стосуються тільки створення моделі типу BFM.
models.py	Модуль для нормування даних із BFM датасету для подальшого використання.
Image_to_3d.py	Основний модуль програми, що містить весь процес моделювання обличчя, модуль не зберігає логіку обчислення моделі, ця частина виділена в окремій модулі для зручнішого супроводу. Модуль можна використовувати для роботи з програмою в консольному режимі.
app.py	Модуль для запуску графічної програми.
ui_controller.py	Клас зберігає логіку роботи з графічним інтерфейсом програми.

Основні функції, що використовуються у програмній системі наведені у таблиці 3.2.

Таблиця 3.2 – Функції програмної системи

Модуль	Функція	Призначення
FaceModel.py	get_renderer	Функція для створення об'єкта для рендерингу, містить налаштування освітлення, камери, та налаштування рендерингу.
FaceModel.py	compute_norm	Функція розрахунку нормалі.

Модуль	Функція	Призначення
FaceModel.py	add_illumination	Функція розрахунку освітлення для однієї вершини
BFMFaceModel.py	split_coefficients	Вилучення даних з BFM моделі.
BFMFaceModel.py	forward	Визначає обчислення, яке виконується при кожному виклику при рендерингу моделі.
BFMFaceModel.py	get_color	Вилучення кольору з текстури обличчя.
Image_to_3d.py	make_model_by_image	Функція створення моделі із переданих налаштувань.
Image_to_3d.py	save_face	Функція для збереження відрендереної моделі та експорту моделі у файл.

Клас FaceModel успадковує клас torch.nn.Module [45], базовий клас для всіх модулів нейронних мереж, створює об'єкт, який поводить себе як функція, але може містити стан (наприклад, ваги шару нейронної мережі), є можливість отримати доступ до вмісту параметрів, обнулити всі їх градієнти, перебирати їх для оновлення ваги і т.д.

Діаграма класів зображена на рисунку 3.3.

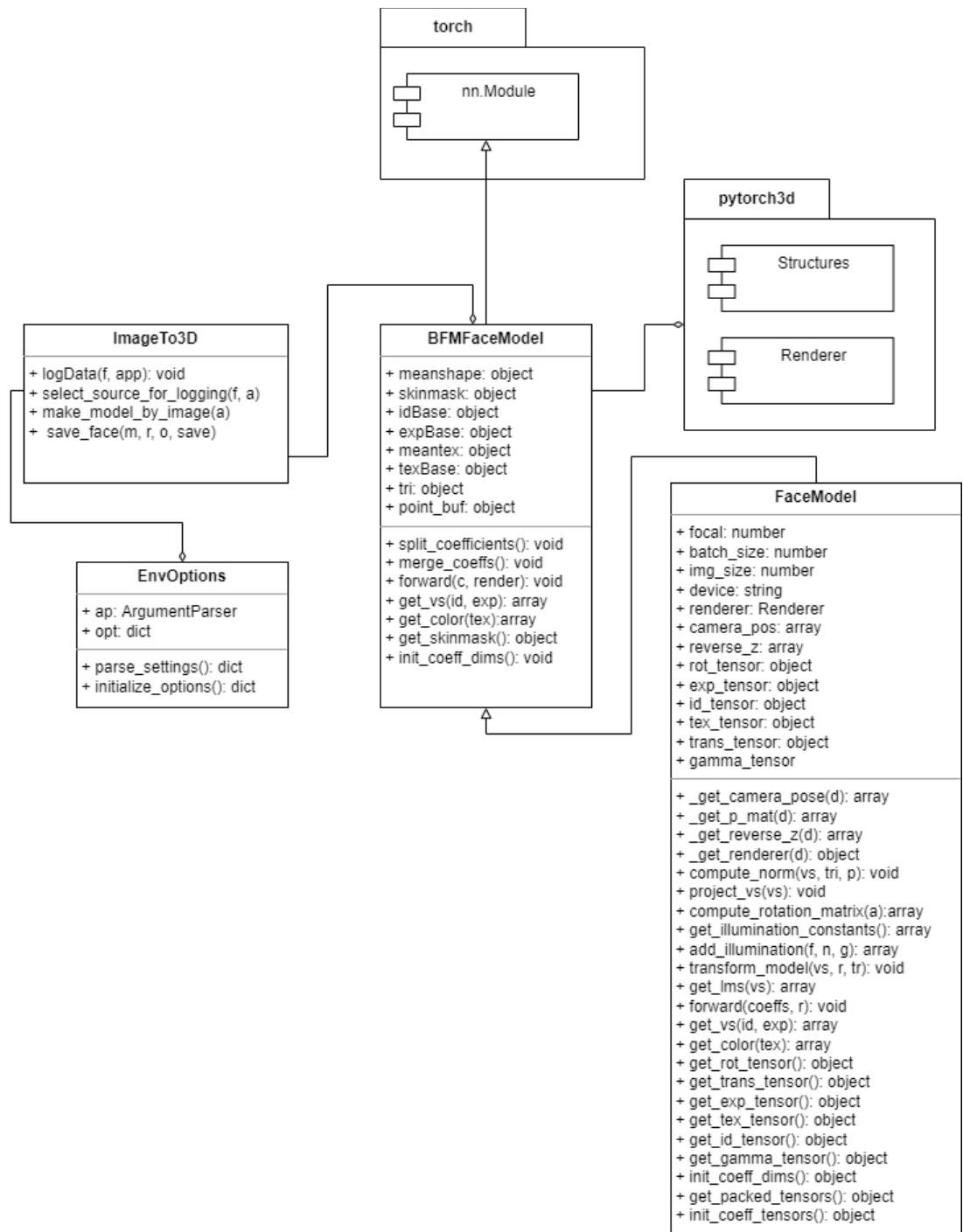


Рисунок 3.3 – Діаграма класів

Для розпізнавання обличчя була обрана бібліотека MTCNN, з використанням глибоко каскадної багатозадачної структури з використанням функцій «підмоделей», кожна з яких посилює їх сильні корелюючі сторони, є можливість запуску обчислення на центральному процесорі або з використанням графічного процесора.

На рисунку 3.4 показаний приклад виявлення обличчя без оклюзії, і рисунку 3.5 область обличчя з оклюзією.



Рисунок 3.4 – Виявлення обличчя без оклюзії

Для знаходження орієнтирів обличчя було обрано бібліотеку face_alignment [38] здатну виявляти орієнтири у 2D та 3D координатах на основі глибокого навчання.

Приклад виявлення орієнтирів наведено рисунку 3.6

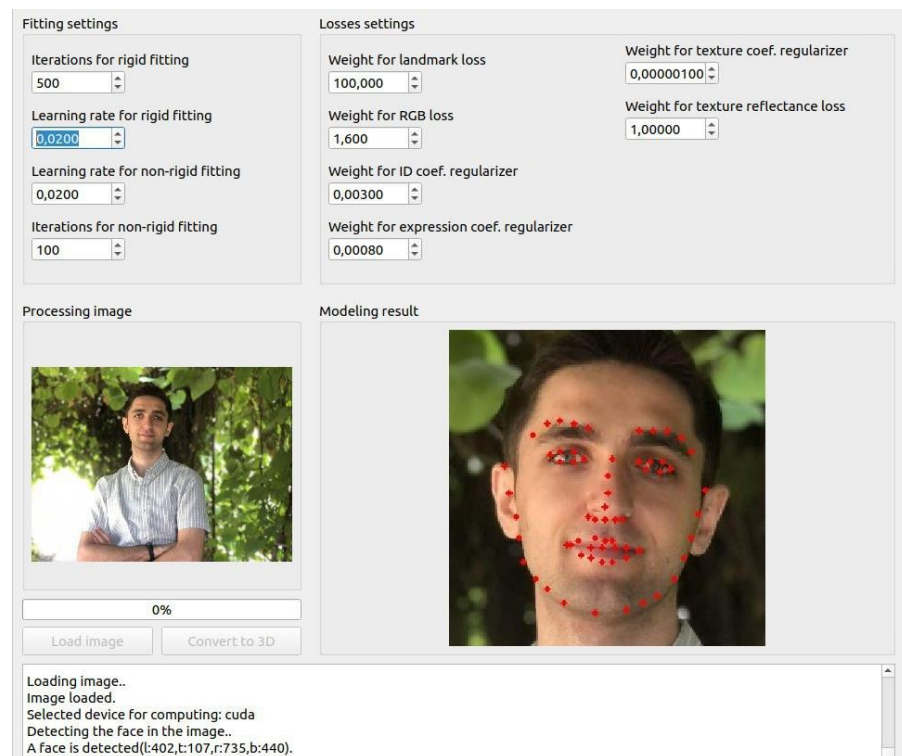


Рисунок 3.6 – Виявлення орієнтирів обличчя

3.5 Висновки до розділу

У цьому розділі була описана програмна система для генерації моделі обличчя з фотографії, структура проекту та бібліотеки, які використовуються. Наведено опис модулів та функцій програми, описано принцип роботи з програмою-інтерфейсом для безпосередньої генерації обличчя.

Як засіб розробки було обрано мову програмування Python та фреймворк для машинного навчання PyTorch. Було наведено приклад розробленого графічного інтерфейсу.

РОЗДІЛ 4

ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ РЕЗУЛЬТАТІВ РОБОТИ НЕЙРОННОЇ
МЕРЕЖІ МАСШТАБУВАННЯ ЗОБРАЖЕНЬ

4.1 Тестування програмної системи

Тестування програмної системи проводилися в зображеннях, що містять різні емоції, нахили, оклюзії. На рисунку 4.1 наведено приклад головного вікна програми після запуску.

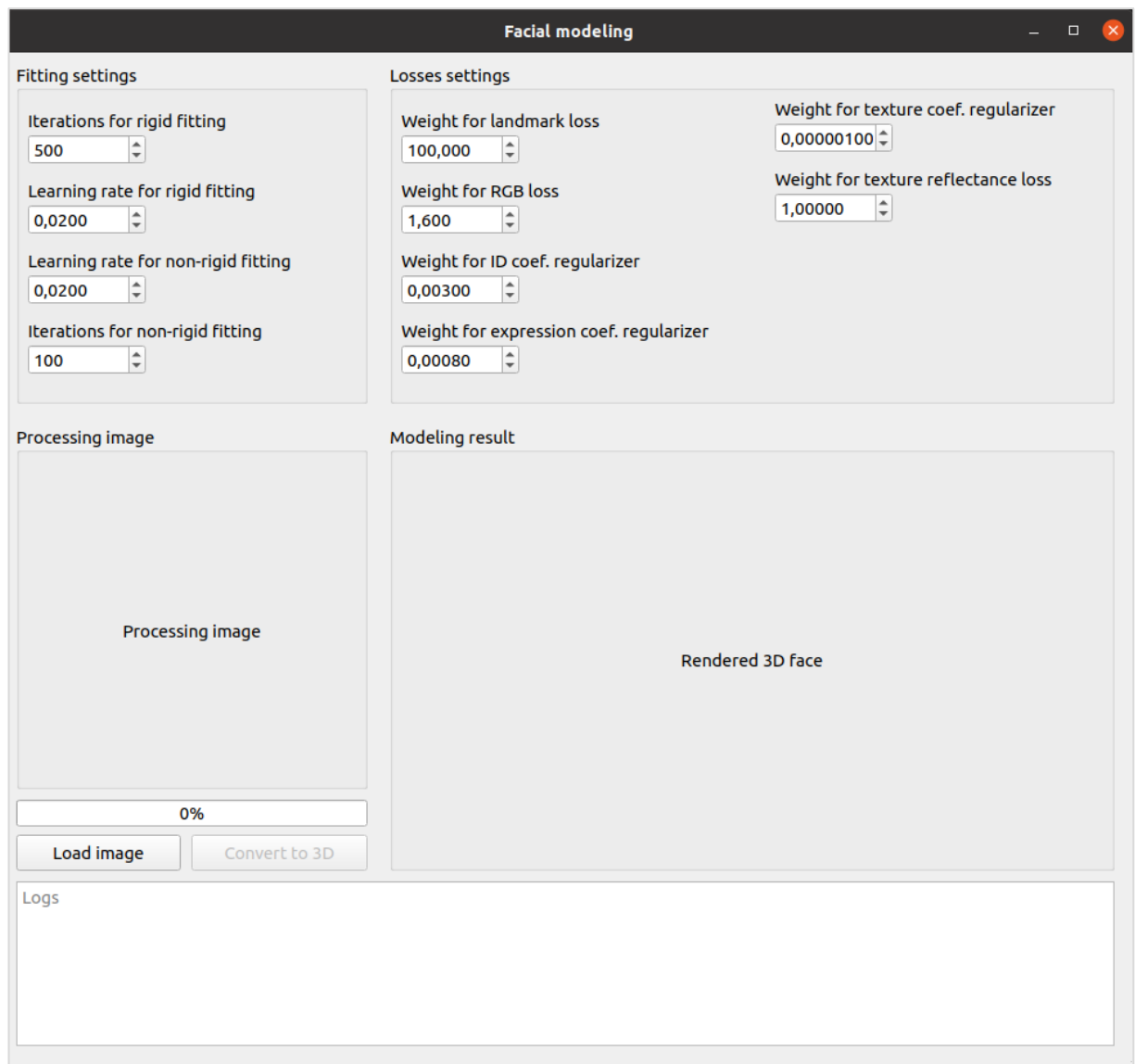


Рисунок 4.1 – Вікно програми

На рисунку 4.2 наведено приклад завантаження у програму портрета з налаштуваннями 100 ітерацій для функції оптимізації.

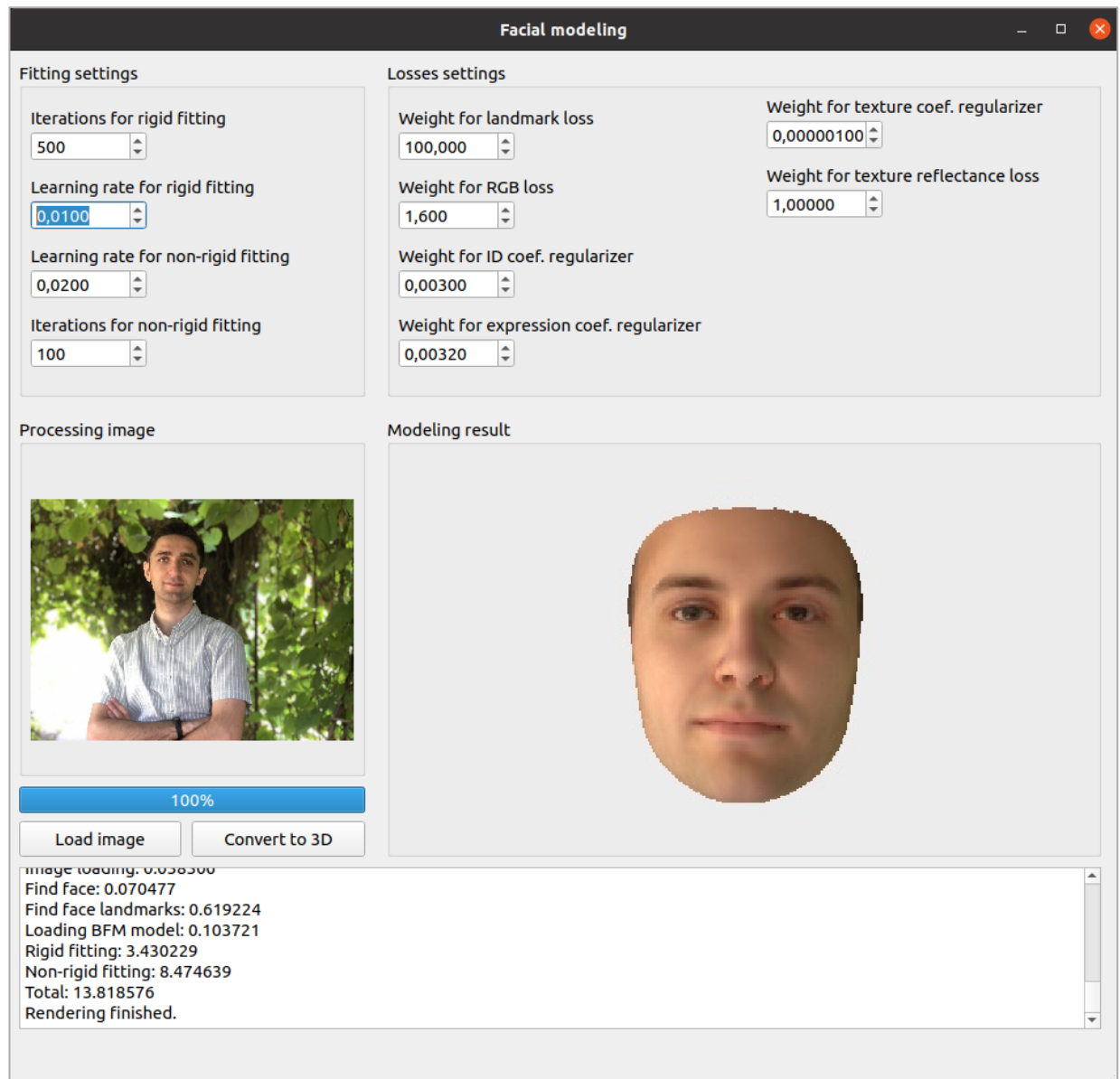


Рисунок 4.2 – Вікно програми

В результаті моделювання програма створює директорію з оригінальним зображенням, відрендереною моделлю, 3D моделлю обличчя у форматі obj та файлом, що містить подробиці про обробку моделі і час витрачений на кожен етап у процесі моделювання. На рисунку 5.3 наведено приклад створеної директорії. При використанні 500 ітерація для жорсткого підгонки та 100 не жорсткої витрачено 13.8 секунд для моделювання обличчя.

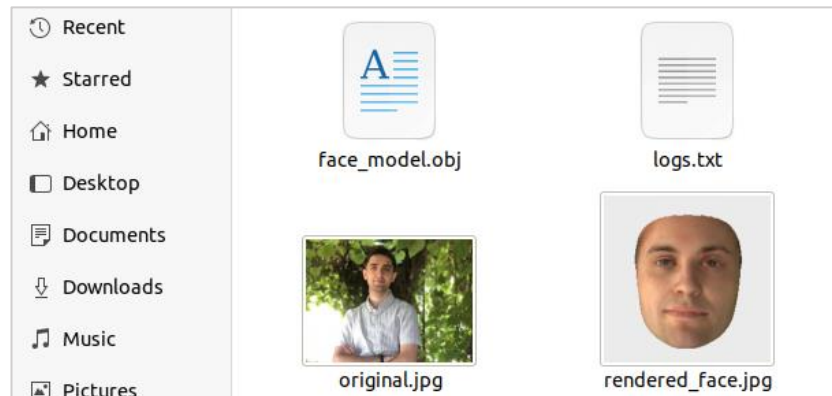


Рисунок 4.3 – Вікно програми

Передача того ж зображення але з передачею 1000 ітерацій у оптимізацію, операція займає довше часу але модель виходить більш наближеною до завантаженої фотографії, на рисунку 4.4 наведено результат програми.

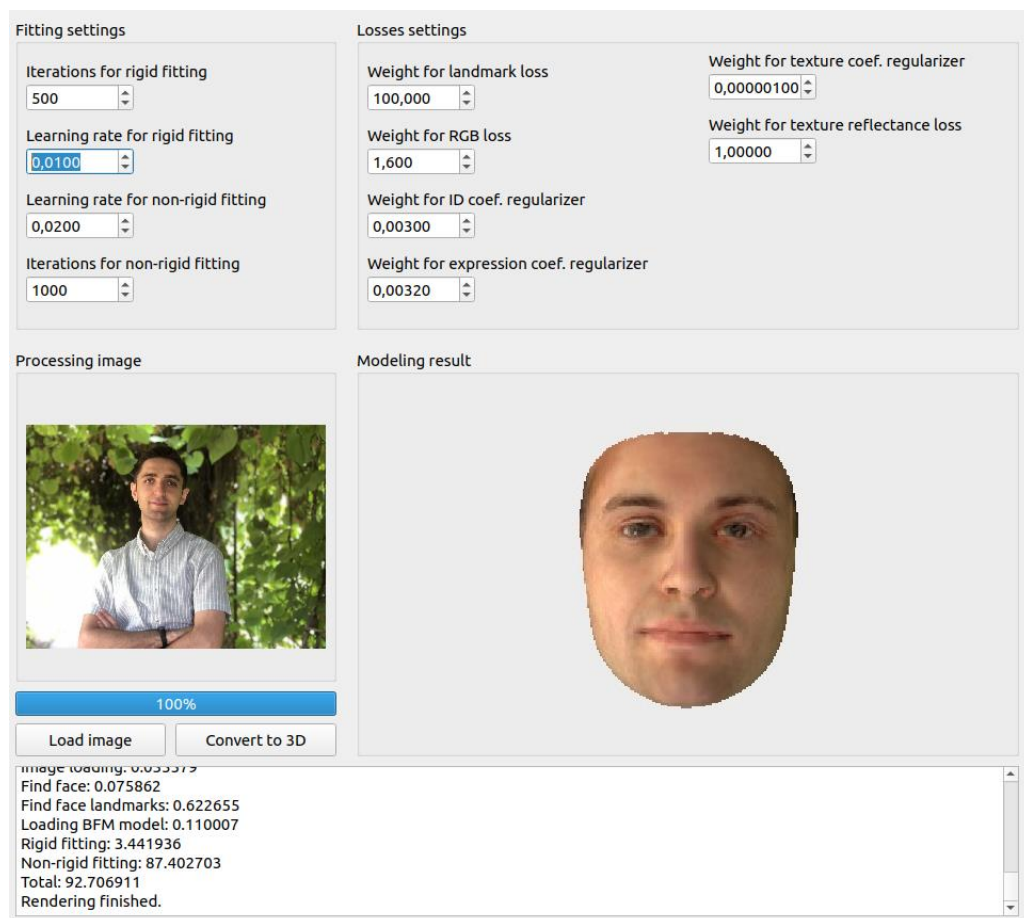
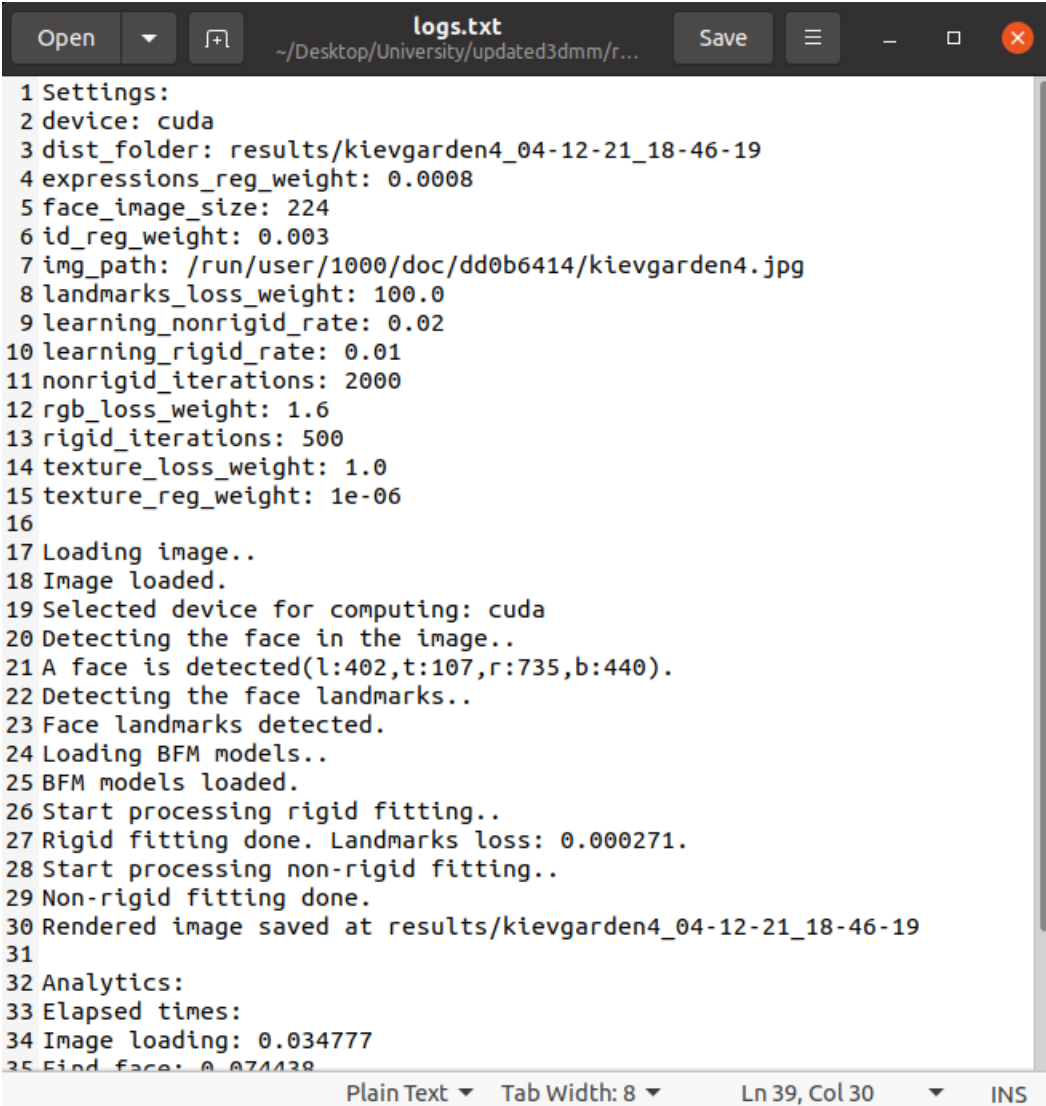


Рисунок 4.4 – Вікно програми

На рисунку 4.5 наведено приклад журнального файлу для моделі.



```

1 Settings:
2 device: cuda
3 dist_folder: results/kiyvgarden4_04-12-21_18-46-19
4 expressions_reg_weight: 0.0008
5 face_image_size: 224
6 id_reg_weight: 0.003
7 img_path: /run/user/1000/doc/dd0b6414/kiyvgarden4.jpg
8 landmarks_loss_weight: 100.0
9 learning_nonrigid_rate: 0.02
10 learning_rigid_rate: 0.01
11 nonrigid_iterations: 2000
12 rgb_loss_weight: 1.6
13 rigid_iterations: 500
14 texture_loss_weight: 1.0
15 texture_reg_weight: 1e-06
16
17 Loading image..
18 Image loaded.
19 Selected device for computing: cuda
20 Detecting the face in the image..
21 A face is detected(l:402,t:107,r:735,b:440).
22 Detecting the face landmarks..
23 Face landmarks detected.
24 Loading BFM models..
25 BFM models loaded.
26 Start processing rigid fitting..
27 Rigid fitting done. Landmarks loss: 0.000271.
28 Start processing non-rigid fitting..
29 Non-rigid fitting done.
30 Rendered image saved at results/kiyvgarden4_04-12-21_18-46-19
31
32 Analytics:
33 Elapsed times:
34 Image loading: 0.034777
35 Find face: 0.071438
  
```

Рисунок 4.5 – Журнальний файл

На рисунку 4.6 приклад обробка зображення з оклюзіями на обличчі.

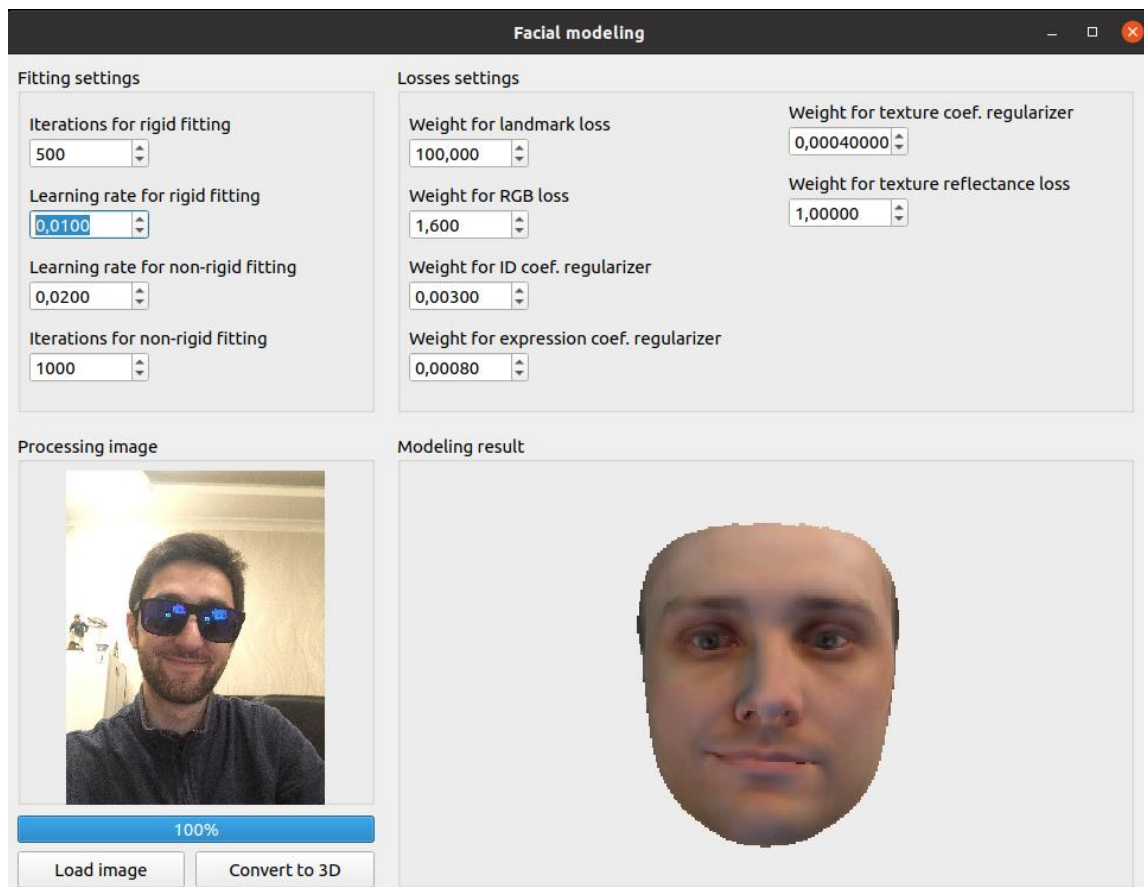


Рисунок 4.5 – Обробка зображення з оклюзіями на обличчі

На рисунку 4.7 наведено приклад обробки зображення з вираженими емоціями обличчя із використанням 1000 ітерацій для функції оптимізації.

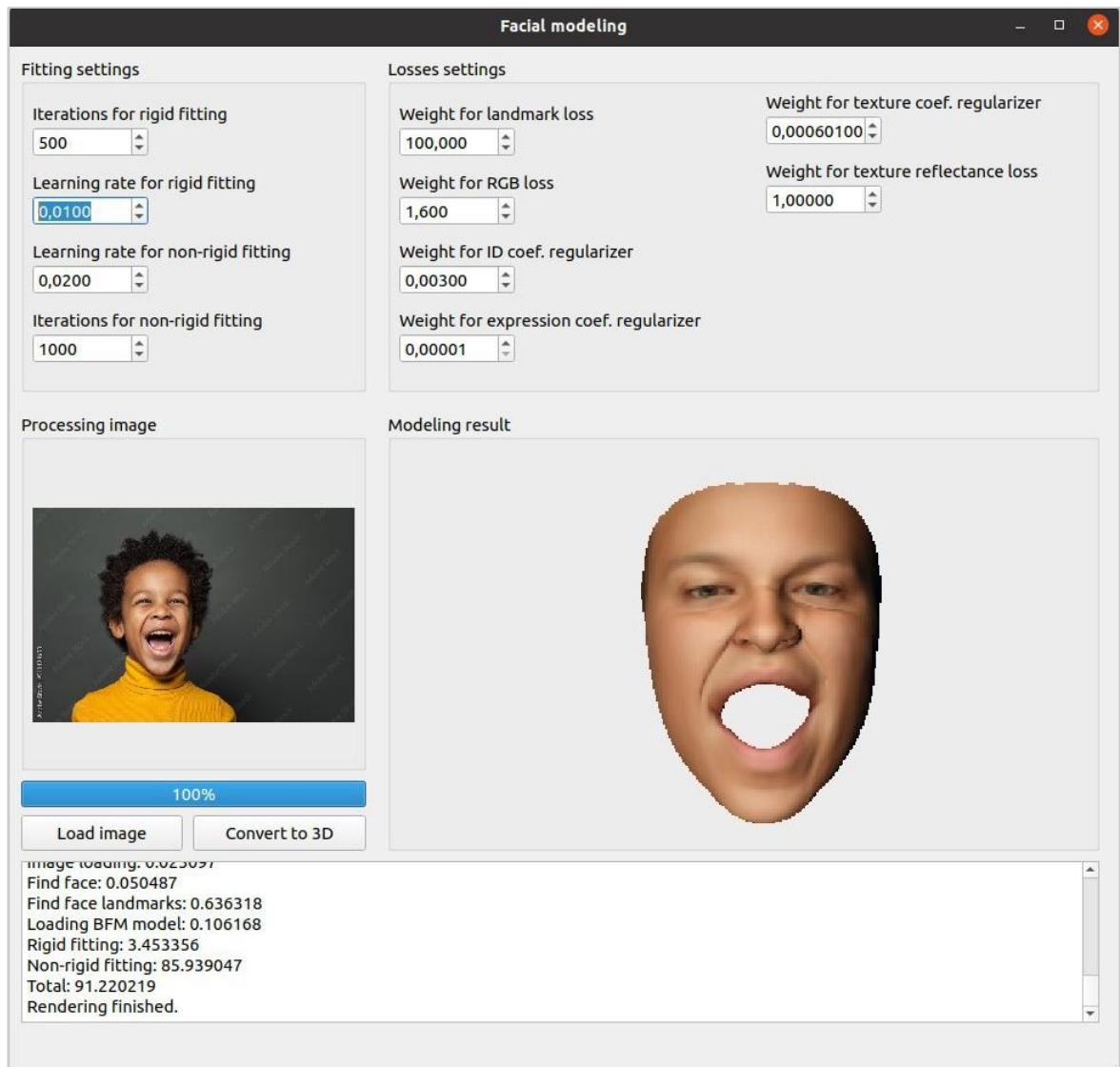


Рисунок 4.7 – Обробка зображення з оклюзіями на обличчі

На рисунку 4.8 наведено приклад перегляду згенерованого obj-файлу моделі в програмі MeshLab.

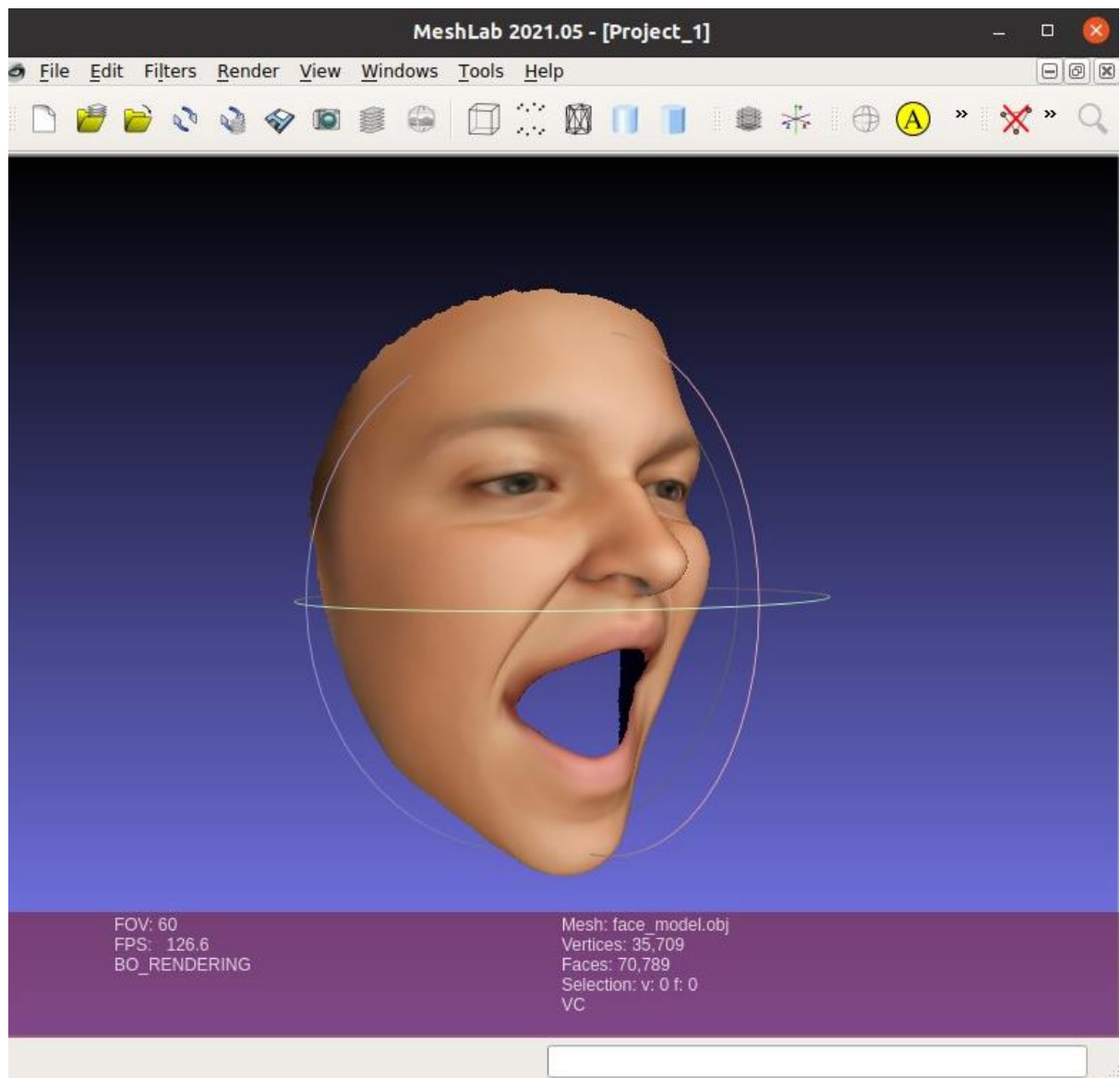


Рисунок 4.8 – Перегляду згенерованого файлу моделі

Приклад роботи програми в консольному режимі наведено на рисунку 4.9, приклад включає передачу параметрів зображення для завантаження та місце для збереження.

На рисунку 4.10 відображено для створення обличчя з використанням 100 ітерацій

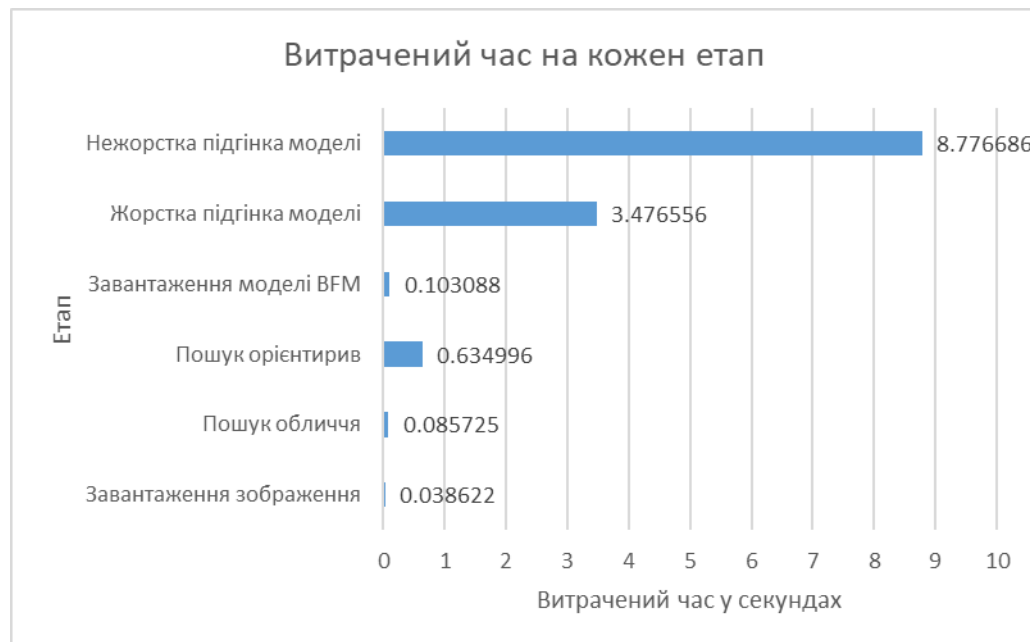


Рисунок 4.10 – Витрачений час на кожен етап

Витрачений час на кожен етап для створення обличчя в секундах наведено в таблиці 5.1 з використанням 500,1000 та 1000 ітерацій.

Таблиця 4.3 – Витрачений час на кожен етап для створення обличчя

Етап \ Ітерації	100	500	1000
Завантаження зображення	0.038622	0.035757	0.034777
Пошук обличчя	0.085725	0.069498	0.074438
Пошук орієнтирів	0.634996	0.620067	0.618354
Завантаження моделі BFM	0.103088	0.10012	0.100897
Жорстка підгінка моделі	3.476556	3.232674	3.56682
Нежорстка підгінка моделі	8.776686	43.837338	87.402703

Графік таблиці 4.3 наведено на рисунку 4.11

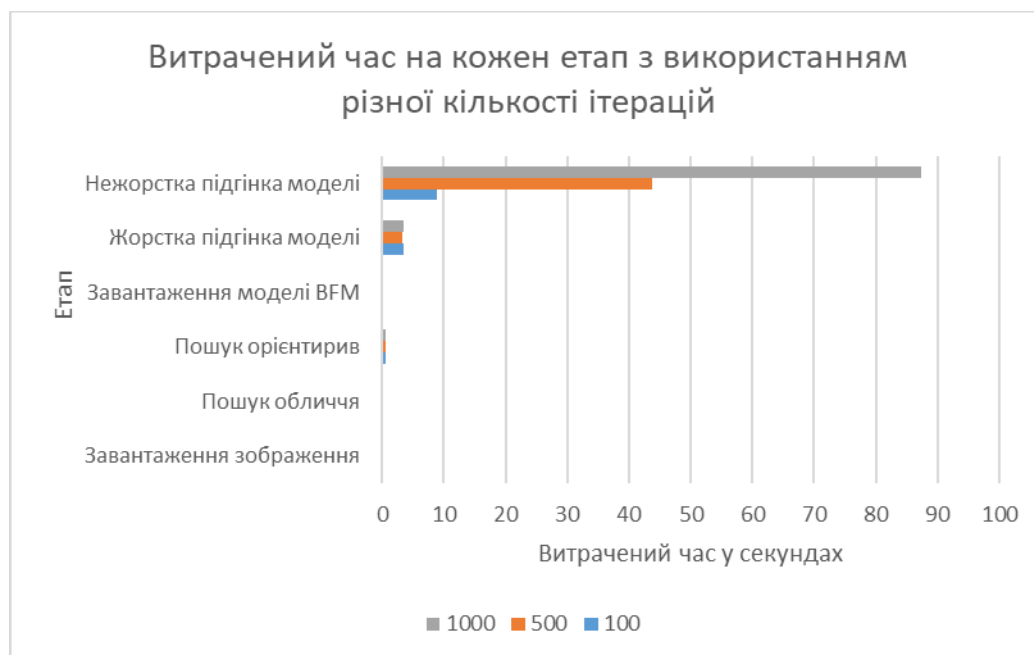


Рисунок 4.11 – Витрачений час на кожен етап

4.3 Висновки до розділу

У даному розділі було проведено аналіз та тестування роботи програмного забезпечення, програмна система зуміла виявити та змодельовати обличчя з оклюзією та без, велика кількість ітерацій для функції оптимізації дає кращі результати, але потребує більше часу.

Завдяки використанню графічного інтерфейсу робота з програмою не потребує особливої підготовки та вміння працювати з терміналом. Проведе тест для моделювання осіб з 100, 500 та 1000.

ВИСНОВКИ

Завдання моделювання 3D обличчя на сьогоднішній день досить актуальне, через велику кількість наукових досліджень, та в комерційних цілях, також програмна система має практичну цінність.

Метою даного дослідження було визначити готову програму для моделювання обличчя на основі зображення, визначено оптимальні алгоритми для моделювання та рекомендації щодо використання. Основне завдання дослідження було розробити систему моделювання обличчя, та порівняти обрані методи з альтернативними. У ході дослідження було розроблено програму, яка дозволить використовувати систему для моделювання обличчя від завантаженого файлу.

Було побудовано архітектуру програмної системи. І обрано найбільш оптимальні методи для розпізнавання обличчя та орієнтирів обличчя, заснованої на глибокому навчанні, а також метод створення моделі з використанням морфної моделі. Розглянуто методи реконструкції моделі обличчя на основі 3DMM або CNN, а також особливості кожного з них.

Проведено аналіз та тестування роботи програмного забезпечення, програмна система зуміла виявити та змоделювати обличчя з оклюзією та без, велика кількість ітерацій для оптимізації дає кращі результати, але потребує більше часу.

Завдяки використанню графічного інтерфейсу робота з програмою не потребує особливої підготовки та вміння працювати з терміналом. Програма показала хороші результати за швидкістю та якістю моделі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 3D Morphable Face Models - Past, Present and Future [Електронний ресурс]. - Режим доступу: <https://arxiv.org/pdf/1909.01815.pdf>
2. A Morphable Model For The Synthesis Of 3D Faces [Електронний ресурс].
— Режим доступу:
<https://gravis.dmi.unibas.ch/publications/Sigg99/morphmod2.pdf>
3. A 3D Face Model for Pose and Illumination Invariant Face Recognition. -
Режим доступу: <https://gravis.dmi.unibas.ch/publications/2009/BFModel09.pdf>
4. Optimal Step Nonrigid ICP Algorithms for Surface Registration [Електронний ресурс]. - Режим доступу:
https://gravis.dmi.unibas.ch/publications/2007/CVPR07_Amberg.pdf
5. 3D Face Reconstruction from A Single Image Assisted by 2D Face Images in the Wild - Documentation [Електронний ресурс]. - Режим доступу:
<https://arxiv.org/pdf/1903.09359.pdf>
6. End-to-end 3D face reconstruction with deep neural networks [Електронний ресурс]. - Режим доступу: <https://arxiv.org/pdf/1704.05020.pdf>
7. Convolutional neural network [Електронний ресурс]. - Режим доступу:
https://en.wikipedia.org/wiki/Convolutional_neural_network
8. Automated 3D Face Reconstruction from Multiple Images using Quality Measures [Електронний ресурс]. - Режим доступу: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Piotraschke_Automated_3D_Face_CVPR_2016_paper.pdf
9. Do We Really Need to Collect Millions of Faces for Effective Face Recognition? [Електронний ресурс]. - Режим доступу:
<https://arxiv.org/abs/1603.07057>
10. Statistical Symmetric Shape from Shading for 3D Structure Recovery of Faces [Електронний ресурс]. - Режим доступу:

- https://www.researchgate.net/publication/227154242_Statistical_Symmetric_Shape_from_Shading_for_3D_Structure_Recovery_of_Faces
11. Example Based 3D Reconstruction from Single 2D Images [Электронный ресурс]. - Режим доступа: <https://ieeexplore.ieee.org/document/1640454>
 12. Adaptive 3D Face Reconstruction from Unconstrained Photo Collections [Электронный ресурс]. - Режим доступа: https://openaccess.thecvf.com/content_cvpr_2016/papers/Roth_Adaptive_3D_Face_CVPR_2016_paper.pdf
 13. Total Moving Face Reconstruction [Электронный ресурс]. - Режим доступа: https://link.springer.com/chapter/10.1007/978-3-319-10593-2_52
 14. Exchanging Faces in Images [Электронный ресурс]. - Режим доступа: <https://gravis.dmi.unibas.ch/publications/EG04.pdf>
 15. D-Aided Face Recognition Robust to Expression and Pose Variations – Краткое руководство [Электронный ресурс]. - Режим доступа: <https://ieeexplore.ieee.org/document/6909642>
 16. Real-time conversion from a single 2D face image to a 3D text-driven emotive audio-visual avatar [Электронный ресурс]. - Режим доступа: <https://ieeexplore.ieee.org/document/4607657>
 17. Efficient, Robust and Accurate Fitting of a 3D Morphable Model [Электронный ресурс]. - Режим доступа: http://lear.inrialpes.fr/people/triggs/events/iccv03/cdrom/iccv03/0059_romdhani.pdf
 18. Learning 3D Face Morphable Model Out of 2D Images [Электронный ресурс]. - Режим доступа: <https://neurohive.io/en/state-of-the-art/learning-3d-face-morphable-model-out-of-2d-images/>
 19. Face Recognition using Principle Component Analysis [Электронный ресурс]. - Режим доступа: <http://staff.ustc.edu.cn/~zwp/teach/MVA/pcaface.pdf>
 20. М.А. Turk and A.P. Pentland, “Face Recognition Using Eigenfaces” [Электронный ресурс]. - Режим доступа: <https://sites.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf>

21. Analysis and Implementation of Optimization Techniques for Facial Recognition [Электронный ресурс]. - Режим доступа: <https://www.hindawi.com/journals/acisc/2021/6672578/>
22. Facial Expression Recognition System Using Extreme Learning Machine [Электронный ресурс]. - Режим доступа: https://www.researchgate.net/publication/329509342_Facial_Expression_Recognition_System_Using_Extreme_Learning_Machine
23. Analysis and Implementation of Optimization Techniques for Facial Recognition [Электронный ресурс]. - Режим доступа: https://pdfs.semanticscholar.org/7cb2/27fb5acdc99a3f8b09fad23073bca487c17d.pdf?_ga=2.87489886.751696467.1638706371-1528164562.1633872940
24. Rich feature hierarchies for accurate object detection and semantic segmentation [Электронный ресурс]. - Режим доступа: <https://arxiv.org/pdf/1311.2524.pdf>
25. Convolutional face finder: A neural architecture for fast and robust face detection [Электронный ресурс]. - Режим доступа: https://www.researchgate.net/publication/3193801_Convolutional_face_finder_A_neural_architecture_for_fast_and_robust_face_detection
26. Multi-view Face Detection Using Deep Convolutional Neural Networks [Электронный ресурс]. - Режим доступа: <https://www.arxiv-vanity.com/papers/1502.02766/>
27. Radial Basis Function Network [Электронный ресурс]. - Режим доступа: <https://www.sciencedirect.com/topics/engineering/radial-basis-function-network>
28. Bilinear CNN Models for Fine-Grained Visual Recognition [Электронный ресурс]. - Режим доступа: <https://ieeexplore.ieee.org/document/7410527>
29. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks [Электронный ресурс]. - Режим доступа: <https://arxiv.org/ftp/arxiv/papers/1604/1604.02878.pdf>

30. The Development and Challenges of Face Alignment Algorithms [Электронный ресурс]. - Режим доступа: <https://iopscience.iop.org/article/10.1088/1742-6596/1335/1/012009/pdf>
31. A comparative study of face landmarking techniques [Электронный ресурс]. - Режим доступа: <https://jivp.eurasipjournals.springeropen.com/track/pdf/10.1186/1687-5281-2013-13.pdf>
32. A comparative study of face landmarking techniques [Электронный ресурс]. - Режим доступа: https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
33. Root-mean-square deviation [Электронный ресурс]. - Режим доступа: https://en.wikipedia.org/wiki/Root-mean-square_deviation
34. CUDA [Электронный ресурс]. - Режим доступа: <https://en.wikipedia.org/wiki/CUDA>
35. Face Recognition Using Pytorch [Электронный ресурс]. - Режим доступа: <https://github.com/timesler/facenet-pytorch>
36. OpenCL [Электронный ресурс]. - Режим доступа: <https://ru.wikipedia.org/wiki/OpenCL>
37. OpenCV [Электронный ресурс]. - Режим доступа: <https://en.wikipedia.org/wiki/OpenCV>
38. Face Recognition [Электронный ресурс]. - Режим доступа: <https://github.com/1adrianb/face-alignment>
39. NumPy [Электронный ресурс]. - Режим доступа: <https://numpy.org/>
40. PyTorch [Электронный ресурс]. - Режим доступа: <https://pytorch.org/>
41. Anaconda [Электронный ресурс]. - Режим доступа: <https://www.anaconda.com/>
42. PyQt [Электронный ресурс]. - Режим доступа: <https://en.wikipedia.org/wiki/PyQt>
43. Qt Creator [Электронный ресурс]. - Режим доступа: https://ru.wikipedia.org/wiki/Qt_Creator
44. MeshLab [Электронный ресурс]. - Режим доступа: <https://www.meshlab.net/>

45. PyTorch Module [Электронный ресурс]. - Режим доступа: <https://pytorch.org/docs/stable/generated/torch.nn.Module.html>
46. Face Recognition Using Neural Network: A Review [Электронный ресурс]. - Режим доступа: http://article.nadiapub.com/IJSIA/vol10_no3/8.pdf
47. Basel Face Model - Details [Электронный ресурс]. - Режим доступа: <https://faces.dmi.unibas.ch/bfm/index.php?nav=1-1-0&id=details>

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

[illegible]

Додаток Б

Лістинг основних програмних модулів

Б.1 Клас BFMFaceModel

```

import torch
from pytorch3d.structures import Meshes
from scipy.io import loadmat
from torch import tensor
from faceModel.FaceModel import FaceModel
from pytorch3d.renderer import (TexturesVertex)
class BFMFaceModel(FaceModel):
    def __init__(self, **kargs):
        super(BFMFaceModel, self).__init__(**kargs)
        bfm_model = loadmat('BFM/BFM09_model_info.mat')
        self.skinmask = torch.tensor(
            bfm_model['skinmask'], requires_grad=False, device=self.device)
        self.kp_inds = torch.tensor(
            bfm_model['keypoints'] - 1).squeeze().long().to(self.device)
        self.meanshape = torch.tensor(bfm_model['meanshape'],
            dtype=torch.float32, requires_grad=False,
            device=self.device)
        self.idBase = torch.tensor(bfm_model['idBase'],
            dtype=torch.float32, requires_grad=False,
            device=self.device)
        self.expBase = torch.tensor(bfm_model['exBase'],
            dtype=torch.float32, requires_grad=False,
            device=self.device)
        self.meantex = torch.tensor(bfm_model['meantex'],
            dtype=torch.float32, requires_grad=False,
            device=self.device)
        self.texBase = torch.tensor(bfm_model['texBase'],
            dtype=torch.float32, requires_grad=False,
            device=self.device)
        self.tri = torch.tensor(bfm_model['tri'] - 1,
            dtype=torch.int64, requires_grad=False,
            device=self.device)
        self.point_buf = torch.tensor(bfm_model['point_buf'] - 1,
            dtype=torch.int64, requires_grad=False,
            device=self.device)
    def get_lms(self, vs):
        lms = vs[:, self.kp_inds, :]
        return lms

    @staticmethod
    def split_coefficients(coeffs):
        # identity(shape) coeff of dim
        id_coeff = coeffs[:, :80]
        exp_coeff = coeffs[:, 80:144]
        # texture albedo coefficient
        tex_coeff = coeffs[:, 144:224]
        angles = coeffs[:, 224:227]

```

```

gamma = coeffs[:, 227:254]
translation = coeffs[:, 254:]
return id_coeff, exp_coeff, tex_coeff, angles, gamma, translation

def merge_coeffs(self, id_coeff, exp_coeff, tex_coeff, angles, gamma, translation):
    coeffs = torch.cat([id_coeff, exp_coeff, tex_coeff,
                        angles, gamma, translation], dim=1)
    return coeffs

def forward(self, coeffs, render=True):
    batch_num = coeffs.shape[0]
    id_coeff, exp_coeff, tex_coeff, angles, gamma, translation = self.split_coefficients(coeffs)
    vs = self.get_vs(id_coeff, exp_coeff)
    angles = tensor([[0.0, 0.0, 0.0]], device=self.device)
    rotation = self.compute_rotation_matrix(angles)
    vs_t = self.transform_model(
        vs, rotation, translation)

    lms_t = self.get_lms(vs_t)
    lms_proj = self.project_vs(lms_t)
    lms_proj = torch.stack(
        [lms_proj[:, :, 0], self.img_size - lms_proj[:, :, 1]], dim=2)
    if render:
        face_texture = self.get_color(tex_coeff)
        face_norm = self.compute_norm(vs, self.tri, self.point_buf)
        face_norm_r = face_norm.bmm(rotation)
        face_color = self.add_illumination(face_texture, face_norm_r, gamma)
        textures = TexturesVertex(face_color)
        mesh = Meshes(
            verts=vs_t,
            faces=self.tri.repeat(batch_num, 1, 1),
            textures=textures
        )
        rendered_img = self.renderer(mesh)
        rendered_img = torch.clamp(rendered_img, 0, 255)
        return {'rendered_img': rendered_img,
                'lms_proj': lms_proj,
                'face_texture': face_texture,
                'vs': vs_t,
                'tri': self.tri,
                'color': face_color,
                'mesh': mesh}
    else:
        return {'lms_proj': lms_proj}

def get_vs(self, id_coeff, exp_coeff):
    n_b = id_coeff.size(0)
    face_shape = torch.einsum('ij,aj->ai', self.idBase, id_coeff) + \
        torch.einsum('ij,aj->ai', self.expBase, exp_coeff) + self.meanshape
    face_shape = face_shape.view(n_b, -1, 3)
    face_shape = face_shape - \
        self.meanshape.view(1, -1, 3).mean(dim=1, keepdim=True)
    return face_shape

def get_color(self, tex_coeff):
    n_b = tex_coeff.size(0)
    face_texture = torch.einsum(
        'ij,aj->ai', self.texBase, tex_coeff) + self.meantex

```



```

        face_texture = face_texture.view(n_b, -1, 3)
        return face_texture

    def get_skinmask(self):
        return self.skinmask

    def init_coeff_dims(self):
        self.id_dims = 80
        self.tex_dims = 80
        self.exp_dims = 64

```

Б.2 Класс FaceModel

```

import torch
import torch.nn as nn
import numpy as np
from pytorch3d.renderer import (
    look_at_view_transform,
    FoVPerspectiveCameras,
    PointLights,
    RasterizationSettings,
    MeshRenderer,
    MeshRasterizer,
    SoftPhongShader,
    blending
)
class FaceModel(nn.Module):
    def __init__(self, batch_size=1,
                  focal=1015, img_size=224, device='cuda:0'):
        super(FaceModel, self).__init__()
        self.focal = focal
        self.batch_size = batch_size
        self.img_size = img_size
        self.device = torch.device(device)
        self.renderer = self._get_renderer(self.device)
        self.p_mat = self._get_p_mat(device)
        self.reverse_z = self._get_reverse_z(device)
        self.camera_pos = self._get_camera_pose(device)
        self.rot_tensor = None
        self.exp_tensor = None
        self.id_tensor = None
        self.tex_tensor = None
        self.trans_tensor = None
        self.gamma_tensor = None
        self.init_coeff_dims()
        self.init_coeff_tensors()

    @staticmethod
    def _get_camera_pose(device):
        camera_pos = torch.tensor(
            [0.0, 0.0, 10.0], device=device).reshape(1, 1, 3)
        return camera_pos

    def _get_p_mat(self, device):
        half_image_width = self.img_size // 2
        p_matrix = np.array([self.focal, 0.0, half_image_width,

```

```

        0.0, self.focal, half_image_width,
        0.0, 0.0, 1.0], dtype=np.float32).reshape(1, 3, 3)
    return torch.tensor(p_matrix, device=device)

    @staticmethod
    def _get_reverse_z(device):
        reverse_z = np.reshape(
            np.array([1.0, 0, 0, 0, 1, 0, 0, 0, -1.0], dtype=np.float32), [1, 3, 3])
        return torch.tensor(reverse_z, device=device)

    def _get_renderer(self, device):
        R, T = look_at_view_transform(10, 0, 0)
        cameras = FoVPerspectiveCameras(device=device, R=R, T=T, znear=0.01,
                                           zfar=50,
                                           fov=2 * np.arctan(self.img_size // 2 / self.focal) * 180. / np.pi)

        lights = PointLights(device=device, location=[[0.0, 0.0, 1e5]],
                              ambient_color=[[1, 1, 1]],
                              specular_color=[[0., 0., 0.]], diffuse_color=[[0., 0., 0.]])
        renderer = MeshRenderer(
            rasterizer=MeshRasterizer(
                cameras=cameras,
                raster_settings=RasterizationSettings(
                    image_size=self.img_size,
                    blur_radius=0,
                    faces_per_pixel=1,
                )
            ),
            shader=SoftPhongShader(
                device=device,
                cameras=cameras,
                lights=lights,
                blend_params=blending.BlendParams(background_color=[236, 236, 236])
            )
        )
        return renderer

    @staticmethod
    def compute_norm(vs, tri, point_buf):

        face_id = tri
        point_id = point_buf
        v1 = vs[:, face_id[:, 0], :]
        v2 = vs[:, face_id[:, 1], :]
        v3 = vs[:, face_id[:, 2], :]
        e1 = v1 - v2
        e2 = v2 - v3
        face_norm = e1.cross(e2)
        empty = torch.zeros((face_norm.size(0), 1, 3),
                             dtype=face_norm.dtype, device=face_norm.device)
        face_norm = torch.cat((face_norm, empty), 1)
        v_norm = face_norm[:, point_id, :].sum(2)
        v_norm = v_norm / v_norm.norm(dim=2).unsqueeze(2)

        return v_norm

    def project_vs(self, vs):

```

```

batch_size = vs.shape[0]

vs = torch.matmul(vs, self.reverse_z.repeat(
    (batch_size, 1, 1))) + self.camera_pos
aug_projection = torch.matmul(
    vs, self.p_mat.repeat((batch_size, 1, 1)).permute((0, 2, 1)))

face_projection = aug_projection[:, :, :2] / \
    torch.reshape(aug_projection[:, :, 2], [batch_size, -1, 1])
return face_projection

@staticmethod
def compute_rotation_matrix(angles):
    n_b = angles.size(0)
    sin_x = torch.sin(angles[:, 0])
    sin_y = torch.sin(angles[:, 1])
    sin_z = torch.sin(angles[:, 2])
    cos_x = torch.cos(angles[:, 0])
    cos_y = torch.cos(angles[:, 1])
    cos_z = torch.cos(angles[:, 2])
    rot_XYZ = torch.eye(3).view(1, 3, 3).repeat(
        n_b * 3, 1, 1).view(3, n_b, 3, 3).to(angles.device)
    rot_XYZ[0, :, 1, 1] = cos_x
    rot_XYZ[0, :, 1, 2] = -sin_x
    rot_XYZ[0, :, 2, 1] = sin_x
    rot_XYZ[0, :, 2, 2] = cos_x
    rot_XYZ[1, :, 0, 0] = cos_y
    rot_XYZ[1, :, 0, 2] = sin_y
    rot_XYZ[1, :, 2, 0] = -sin_y
    rot_XYZ[1, :, 2, 2] = cos_y
    rot_XYZ[2, :, 0, 0] = cos_z
    rot_XYZ[2, :, 0, 1] = -sin_z
    rot_XYZ[2, :, 1, 0] = sin_z
    rot_XYZ[2, :, 1, 1] = cos_z

    rotation = rot_XYZ[2].bmm(rot_XYZ[1]).bmm(rot_XYZ[0])

    return rotation.permute(0, 2, 1)

@staticmethod
def get_illumination_constants():
    a0 = np.pi
    a1 = 2 * np.pi / np.sqrt(3.0)
    a2 = 2 * np.pi / np.sqrt(8.0)
    c0 = 1 / np.sqrt(4 * np.pi)
    c1 = np.sqrt(3.0) / np.sqrt(4 * np.pi)
    c2 = 3 * np.sqrt(5.0) / np.sqrt(12 * np.pi)
    d0 = 0.5 / np.sqrt(3.0)
    return [a0, a1, a2, c0, c1, c2, d0];

@staticmethod
def add_illumination(face_texture, norm, gamma):
    n_b, num_vertex, _ = face_texture.size()
    n_v_full = n_b * num_vertex
    gamma = gamma.view(-1, 3, 9).clone()
    gamma[:, :, 0] += 0.95

```

[illegible]

```

        self.trans_tensor)

def init_coeff_tensors(self, id_coeff=None, tex_coeff=None):
    if id_coeff is None:
        self.id_tensor = torch.zeros(
            (1, self.id_dims), dtype=torch.float32,
            requires_grad=True, device=self.device)
    else:
        assert id_coeff.shape == (1, self.id_dims)
        self.id_tensor = torch.tensor(
            id_coeff, dtype=torch.float32,
            requires_grad=True, device=self.device)
    if tex_coeff is None:
        self.tex_tensor = torch.zeros(
            (1, self.tex_dims), dtype=torch.float32,
            requires_grad=True, device=self.device)
    else:
        assert tex_coeff.shape == (1, self.tex_dims)
        self.tex_tensor = torch.tensor(
            tex_coeff, dtype=torch.float32,
            requires_grad=True, device=self.device)
    self.exp_tensor = torch.zeros(
        (self.batch_size, self.exp_dims), dtype=torch.float32,
        requires_grad=True, device=self.device)
    self.gamma_tensor = torch.zeros(
        (self.batch_size, 27), dtype=torch.float32,
        requires_grad=True, device=self.device)
    self.trans_tensor = torch.zeros(
        (self.batch_size, 3), dtype=torch.float32,
        requires_grad=True, device=self.device)
    self.rot_tensor = torch.zeros(
        (self.batch_size, 3), dtype=torch.float32,
        requires_grad=True, device=self.device)

```

Б.3 Клас EnvOptions

```
import argparse
```

```

class EnvOptions:
    def __init__(self):
        self.ap = argparse.ArgumentParser()
    def parse_settings(self):
        self.opt = self.ap.parse_args()
        return self.opt
    def initialize_options(self):
        self.ap.add_argument('--img_path', type=str, required=True)
        self.ap.add_argument('--dist_folder', type=str, required=True)
        self.ap.add_argument('--face_image_size', type=int, default=256)
        self.ap.add_argument('--rigid_iterations', type=int, default=1000)
        self.ap.add_argument('--nonrigid_iterations', type=int, default=500)
        self.ap.add_argument('--learning_rigid_rate', type=float, default=1e-2)
        self.ap.add_argument('--learning_nonrigid_rate', type=float, default=1e-2)
        self.ap.add_argument('--landmarks_loss_weight', type=float, default=100)
        self.ap.add_argument('--rgb_loss_weight', type=float, default=1.6)
        self.ap.add_argument('--id_reg_weight', type=float, default=1e-3)
        self.ap.add_argument('--expressions_reg_weight', type=float, default=0.8e-3)

```

```
self.ap.add_argument('--texture_reg_weight', type=float, default=1.7e-6)
self.ap.add_argument('--texture_loss_weight', type=float, default=1)
```

Б.4 Допоміжні функції

```
def box_padding(bbox, image_size, padding_ratio=0.2):
    x1, y1, x2, y2 = bbox
    width = x2 - x1
    height = y2 - y1
    size_box = int(max(width, height) * (1 + padding_ratio))
    center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
    x1 = max(int(center_x - size_box // 2), 0)
    y1 = max(int(center_y - size_box // 2), 0)
    size_box = min(image_size[0] - x1, size_box)
    size_box = min(image_size[1] - y1, size_box)
    return [x1, y1, x1 + size_box, y1 + size_box]

def get_landmarks_weights(device):
    w = torch.ones(68).to(device)
    w[28:31] = 10
    w[48:68] = 10
    lm_norm_weights = w / w.sum()
    return lm_norm_weights

def export_to_obj_format(path, v, f, c):
    with open(path, 'w') as file:
        for i in range(len(v)):
            file.write('v %f %f %f %f %f %f\n' %
                      (v[i, 0], v[i, 1], v[i, 2], c[i, 0], c[i, 1], c[i, 2]))

        file.write('\n')
        for i in range(len(f)):
            file.write('f %d %d %d\n' % (f[i, 0], f[i, 1], f[i, 2]))
    file.close()

import torch

def get_photo_loss(pred_img, gt_img, img_mask):
    pred_img = pred_img.float()
    loss = torch.sqrt(torch.sum(torch.square(
        pred_img - gt_img), 3)) * img_mask / 255
    loss = torch.sum(loss, dim=(1, 2)) / torch.sum(img_mask, dim=(1, 2))
    loss = torch.mean(loss)
    return loss

def get_landmarks_loss(pred_lms, gt_lms, weight, img_size=224):
    loss = torch.sum(torch.square(pred_lms / img_size - gt_lms /
                                   img_size), dim=2) * weight.reshape(1, -1)
    loss = torch.mean(loss.sum(1))
    return loss

def get_l2(tensor):
    return torch.square(tensor).sum()

def reflectance_loss(tex, skin_mask):
    skin_mask = skin_mask.unsqueeze(2)
    tex_mean = torch.sum(tex * skin_mask, 1, keepdims=True) / torch.sum(skin_mask)
    loss = torch.sum(torch.square((tex - tex_mean) * skin_mask / 255.)) / \
        (tex.shape[0] * torch.sum(skin_mask))
    return loss
```

Б.5 Код для графічного інтерфейсу

```

import os
import sys
from datetime import datetime

from PyQt6.QtCore import Qt, QObject, pyqtSignal, QThread
from PyQt6.QtGui import QPixmap
from PyQt6.QtWidgets import QMessageBox, QFileDialog
import torch
import image_to_3d
from appui import *
window = None

class Worker(QObject):
    finished = pyqtSignal()
    logSignal = pyqtSignal(str)
    setProgressBarRange = pyqtSignal(int)
    setProgressBarValue = pyqtSignal(int)
    renderedImageUpdated = pyqtSignal()
    def emitLogEvent(self, data):
        self.logSignal.emit(data)
    def emitSetProgressBarRange(self, progress_bar_range):
        self.setProgressBarRange.emit(progress_bar_range)
    def emitSetProgressBarValue(self, progress_bar_value):
        self.setProgressBarValue.emit(progress_bar_value)
    def emitRenderedImageUpdated(self):
        self.renderedImageUpdated.emit()

    def run(self):
        global window
        ui = window.ui
        rendering_options = {
            'img_path': window.original_image_path,
            'face_image_size': 224,
            'dist_folder': window.get_result_path(),
            'device': 'cuda' if torch.cuda.is_available() else 'cpu',
            'rigid_iterations': ui.numberRigidFittingIterationsSpinBox.value(),
            'nonrigid_iterations': ui.numberNonRigidFittingIterationsSpinBox.value(),
            'learning_rigid_rate': ui.rigidLearningRateSpinBox.value(),
            'learning_nonrigid_rate': ui.nonRigidLearningRateSpinBox.value(),
            'landmarks_loss_weight': ui.weightLandmarksLossSpinBox.value(),
            'rgb_loss_weight': ui.weightRgbLossSpinBox.value(),
            'id_reg_weight': ui.weightIdRegularizerSpinBox.value(),
            'expressions_reg_weight': ui.weightExpRegularizerSpinBox.value(),
            'texture_reg_weight': ui.weightTextureRegularizerSpinBox.value(),
            'texture_loss_weight': ui.weightTextureLossSpinBox.value(),
        }
        image_to_3d.make_model_by_image(DotDictionary(rendering_options), self)
        self.finished.emit()

class UiController:
    def __init__(self):
        global window
        app = QtWidgets.QApplication(sys.argv)
        main_window = QtWidgets.QMainWindow()
        ui = Ui_MainWindow()
        ui.setupUi(main_window)
        self.ui = ui
        self.window = main_window
        self.app = app

```

```

self.original_image_path = "
self.renderingThread = QThread
self.renderingWorker = Worker
window = self
self.filename_suffix = "
def initialize(self):
    self.ui.loadImageButton.clicked.connect(self.show_load_image_dialog)
    self.ui.convertTo3dButton.clicked.connect(self.start_rendering)
def generate_new_filename_suffix(self):
    self.filename_suffix = datetime.now().strftime("_%d-%m-%y_%H-%M-%S")

def show_load_image_dialog(self):
    file_names = QFileDialog.getOpenFileName(
        self.window,
        caption='Select File',
        initialFilter='Image files (*.jpg *png)'
    )
    if file_names[0]:
        self.set_processing_image(file_names[0])
def get_result_path(self):
    original_image_basename = os.path.basename(self.original_image_path)[: -4]
    return 'results/' + original_image_basename + self.filename_suffix
def set_processing_image(self, path):
    self.original_image_path = path
    pixmap = QPixmap(path)
    self.ui.imageCaption.setPixmap(pixmap.scaled(self.ui.imageCaption.size(),
Qt.AspectRatioMode.KeepAspectRatio))
    self.ui.convertTo3dButton.setEnabled(True)

def showWindow(self):
    self.window.show()
    sys.exit(self.app.exec())
def start_rendering(self):
    self.generate_new_filename_suffix()
    self.ui.convertTo3dButton.setEnabled(False)
    self.start_rendering_thread()
def start_rendering_thread(self):
    self.renderingThread = QThread()
    self.renderingWorker = Worker()
    self.renderingWorker.moveToThread(self.renderingThread)
    self.renderingThread.started.connect(self.rendering_started)
    self.renderingThread.started.connect(self.renderingWorker.run)
    self.renderingWorker.finished.connect(self.renderingThread.quit)

    self.renderingWorker.finished.connect(self.renderingWorker.deleteLater)
    self.renderingThread.finished.connect(self.renderingThread.deleteLater)
    self.renderingThread.finished.connect(self.rendering_finished)
    self.renderingWorker.logSignal.connect(self.log)
    self.renderingWorker.setProgressRange.connect(self.set_progress_bar_range)
    self.renderingWorker.setProgressValue.connect(self.set_progress_bar_value)
    self.renderingWorker.renderedImageUpdated.connect(self.rendered_image_updated)

    self.renderingThread.start()

    return self.renderingThread

def set_progress_bar_range(self, maximum):

```



```

self.ui.progressBar.setMaximum(maximum)

def set_progress_bar_value(self, value):
    self.ui.progressBar.setValue(value)

def rendered_image_updated(self):
    rendered_face_path = self.get_result_path() + '/rendered_face.jpg'
    pixmap = QPixmap(rendered_face_path)
    self.ui.resultImageCaption.setPixmap(
        pixmap.scaled(self.ui.resultImageCaption.size(), Qt.AspectRatioMode.KeepAspectRatio)
    )
def rendering_started(self):
    self.ui.progressBar.setValue(0)
    self.ui.listLogsTextEdit.clear()
    self.log('Rendering started.')
    self.ui.convertTo3dButton.setEnabled(False)
    self.ui.loadImageButton.setEnabled(False)

def rendering_finished(self):
    self.ui.convertTo3dButton.setEnabled(True)
    self.ui.loadImageButton.setEnabled(True)
    self.log('Rendering finished.')

def log(self, data):
    self.ui.listLogsTextEdit.append(data)
class DotDictionary(dict):
    __getattr__ = dict.get
    __setattr__ = dict.__setitem__
    __delattr__ = dict.__delitem__

#app.py
from ui_controller import *
app_view = UiController()
app_view.initialize()
app_view.showWindow()
from PyQt6 import QtCore, QtGui, QtWidgets
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(968, 900)
        MainWindow.setMinimumSize(QtCore.QSize(968, 755))
        MainWindow.setMaximumSize(QtCore.QSize(968, 900))
        MainWindow.setMouseTracking(False)
        MainWindow.setAutoFillBackground(False)
        MainWindow.setTabShape(QtWidgets.QTabWidget.TabShape.Rounded)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")

        self.convertTo3dButton = QtWidgets.QPushButton(self.centralwidget)
        self.convertTo3dButton.setEnabled(False)
        self.convertTo3dButton.setGeometry(QtCore.QRect(160, 670, 151, 31))
        self.convertTo3dButton.setFlat(False)
        self.convertTo3dButton.setObjectName("convertTo3dButton")
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(10, 10, 301, 291))
        self.groupBox.setObjectName("groupBox")
        self.label = QtWidgets.QLabel(self.groupBox)

```

```

self.label.setGeometry(QRect(10, 40, 251, 17))
self.label.setObjectName("label")
self.rigidLearningRateSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox)
self.rigidLearningRateSpinBox.setGeometry(QRect(10, 120, 101, 26))
self.rigidLearningRateSpinBox.setDecimals(4)
self.rigidLearningRateSpinBox.setProperty("value", 0.02)
self.rigidLearningRateSpinBox.setObjectName("rigidLearningRateSpinBox")
self.numberRigidFittingIterationsSpinBox = QtWidgets.QSpinBox(self.groupBox)
self.numberRigidFittingIterationsSpinBox.setGeometry(QRect(10, 60, 101, 26))
self.numberRigidFittingIterationsSpinBox.setMinimum(1)
self.numberRigidFittingIterationsSpinBox.setMaximum(10000)
self.numberRigidFittingIterationsSpinBox.setProperty("value", 500)
self.numberRigidFittingIterationsSpinBox.setObjectName("numberRigidFittingIterationsSpinBox")
self.label_2 = QtWidgets.QLabel(self.groupBox)
self.label_2.setGeometry(QRect(10, 220, 241, 17))
self.label_2.setObjectName("label_2")
self.numberNonRigidFittingIterationsSpinBox = QtWidgets.QSpinBox(self.groupBox)
self.numberNonRigidFittingIterationsSpinBox.setGeometry(QRect(10, 240, 101, 26))
self.numberNonRigidFittingIterationsSpinBox.setMinimum(1)
self.numberNonRigidFittingIterationsSpinBox.setMaximum(10000)
self.numberNonRigidFittingIterationsSpinBox.setProperty("value", 100)

self.numberNonRigidFittingIterationsSpinBox.setObjectName("numberNonRigidFittingIterationsSpinBo
x")
self.label_3 = QtWidgets.QLabel(self.groupBox)
self.label_3.setGeometry(QRect(10, 100, 251, 17))
self.label_3.setObjectName("label_3")
self.nonRigidLearningRateSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox)
self.nonRigidLearningRateSpinBox.setGeometry(QRect(10, 180, 101, 26))
self.nonRigidLearningRateSpinBox.setSuffix("")
self.nonRigidLearningRateSpinBox.setDecimals(4)
self.nonRigidLearningRateSpinBox.setProperty("value", 0.02)
self.nonRigidLearningRateSpinBox.setObjectName("nonRigidLearningRateSpinBox")
self.label_4 = QtWidgets.QLabel(self.groupBox)
self.label_4.setGeometry(QRect(10, 160, 241, 17))
self.label_4.setObjectName("label_4")
self.groupBox_2 = QtWidgets.QGroupBox(self.centralwidget)
self.groupBox_2.setGeometry(QRect(10, 320, 301, 311))
self.groupBox_2.setObjectName("groupBox_2")
self.imageCaption = QtWidgets.QLabel(self.groupBox_2)
self.imageCaption.setGeometry(QRect(10, 30, 281, 291))

self.imageCaption.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
self.imageCaption.setObjectName("imageCaption")
self.loadImageButton = QtWidgets.QPushButton(self.centralwidget)
self.loadImageButton.setGeometry(QRect(10, 670, 141, 31))
self.loadImageButton.setObjectName("loadImageButton")
self.groupBox_3 = QtWidgets.QGroupBox(self.centralwidget)
self.groupBox_3.setGeometry(QRect(330, 10, 621, 291))
self.groupBox_3.setObjectName("groupBox_3")
self.label_10 = QtWidgets.QLabel(self.groupBox_3)
self.label_10.setGeometry(QRect(10, 40, 251, 17))
self.label_10.setObjectName("label_10")
self.weightLandmarksLossSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox_3)
self.weightLandmarksLossSpinBox.setGeometry(QRect(10, 60, 101, 26))
self.weightLandmarksLossSpinBox.setDecimals(3)
self.weightLandmarksLossSpinBox.setMaximum(10000.0)

```

```

self.weightLandmarksLossSpinBox.setProperty("value", 100.0)
self.weightLandmarksLossSpinBox.setObjectName("weightLandmarksLossSpinBox")
self.label_11 = QtWidgets.QLabel(self.groupBox_3)
self.label_11.setGeometry(QtCore.QRect(10, 100, 251, 17))
self.label_11.setObjectName("label_11")
self.weightRgbLossSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox_3)

self.weightRgbLossSpinBox.setGeometry(QtCore.QRect(10, 120, 101, 26))
self.weightRgbLossSpinBox.setDecimals(3)
self.weightRgbLossSpinBox.setMaximum(255.0)
self.weightRgbLossSpinBox.setProperty("value", 1.6)
self.weightRgbLossSpinBox.setObjectName("weightRgbLossSpinBox")
self.weightIdRegularizerSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox_3)
self.weightIdRegularizerSpinBox.setGeometry(QtCore.QRect(10, 180, 101, 26))

self.weightIdRegularizerSpinBox.setDecimals(5)
self.weightIdRegularizerSpinBox.setMaximum(255.0)
self.weightIdRegularizerSpinBox.setSingleStep(0.001)
self.weightIdRegularizerSpinBox.setProperty("value", 0.003)
self.weightIdRegularizerSpinBox.setObjectName("weightIdRegularizerSpinBox")
self.label_12 = QtWidgets.QLabel(self.groupBox_3)
self.label_12.setGeometry(QtCore.QRect(10, 160, 251, 17))
self.label_12.setObjectName("label_12")
self.weightExpRegularizerSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox_3)
self.weightExpRegularizerSpinBox.setGeometry(QtCore.QRect(10, 240, 101, 26))
self.weightExpRegularizerSpinBox.setDecimals(5)
self.weightExpRegularizerSpinBox.setMinimum(1e-05)
self.weightExpRegularizerSpinBox.setMaximum(255.0)
self.weightExpRegularizerSpinBox.setSingleStep(0.0008)

self.weightExpRegularizerSpinBox.setStepType(QtWidgets.QAbstractSpinBox.StepType.DefaultStepType)

self.weightExpRegularizerSpinBox.setProperty("value", 0.0008)
self.weightExpRegularizerSpinBox.setObjectName("weightExpRegularizerSpinBox")
self.label_13 = QtWidgets.QLabel(self.groupBox_3)
self.label_13.setGeometry(QtCore.QRect(10, 220, 291, 17))
self.label_13.setObjectName("label_13")
self.weightTextureRegularizerSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox_3)
self.weightTextureRegularizerSpinBox.setGeometry(QtCore.QRect(330, 50, 101, 26))
self.weightTextureRegularizerSpinBox.setPrefix("")
self.weightTextureRegularizerSpinBox.setSuffix("")
self.weightTextureRegularizerSpinBox.setDecimals(8)
self.weightTextureRegularizerSpinBox.setMinimum(0.0)
self.weightTextureRegularizerSpinBox.setMaximum(255.0)
self.weightTextureRegularizerSpinBox.setSingleStep(0.0008)

self.weightTextureRegularizerSpinBox.setStepType(QtWidgets.QAbstractSpinBox.StepType.DefaultStepType)

self.weightTextureRegularizerSpinBox.setProperty("value", 1e-06)
self.weightTextureRegularizerSpinBox.setObjectName("weightTextureRegularizerSpinBox")
self.label_14 = QtWidgets.QLabel(self.groupBox_3)
self.label_14.setGeometry(QtCore.QRect(330, 30, 291, 17))
self.label_14.setObjectName("label_14")
self.weightTextureLossSpinBox = QtWidgets.QDoubleSpinBox(self.groupBox_3)
self.weightTextureLossSpinBox.setGeometry(QtCore.QRect(330, 110, 101, 26))
self.weightTextureLossSpinBox.setDecimals(5)
self.weightTextureLossSpinBox.setMinimum(0.0)

```

```

self.weightTextureLossSpinBox.setMaximum(255.0)
self.weightTextureLossSpinBox.setSingleStep(0.0008)

self.weightTextureLossSpinBox.setStepType(QtWidgets.QAbstractSpinBox.StepType.DefaultStepType)
self.weightTextureLossSpinBox.setProperty("value", 1.0)
self.weightTextureLossSpinBox.setObjectName("weightTextureLossSpinBox")
self.label_20 = QtWidgets.QLabel(self.groupBox_3)
self.label_20.setGeometry(QtCore.QRect(330, 90, 291, 17))
self.label_20.setObjectName("label_20")
self.groupBox_4 = QtWidgets.QGroupBox(self.centralwidget)
self.groupBox_4.setGeometry(QtCore.QRect(330, 320, 621, 381))
self.groupBox_4.setAutoFillBackground(False)
self.groupBox_4.setStyleSheet("")
self.groupBox_4.setObjectName("groupBox_4")
self.resultImageCaption = QtWidgets.QLabel(self.groupBox_4)
self.resultImageCaption.setGeometry(QtCore.QRect(10, 30, 601, 341))
self.resultImageCaption.setStyleSheet("background-color: rgb(236, 236, 236);")
self.resultImageCaption.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
self.resultImageCaption.setObjectName("resultImageCaption")
self.progressBar = QtWidgets.QProgressBar(self.centralwidget)
self.progressBar.setGeometry(QtCore.QRect(10, 640, 301, 23))
self.progressBar.setMaximum(100)
self.progressBar.setProperty("value", 0)
self.progressBar.setObjectName("progressBar")

self.listLogsTextEdit = QtWidgets.QTextEdit(self.centralwidget)
self.listLogsTextEdit.setGeometry(QtCore.QRect(10, 710, 941, 141))
self.listLogsTextEdit.setFrameShape(QtWidgets.QFrame.Shape.StyledPanel)
self.listLogsTextEdit.setFrameShadow(QtWidgets.QFrame.Shadow.Sunken)
self.listLogsTextEdit.setDocumentTitle("")
self.listLogsTextEdit.setReadOnly(True)
self.listLogsTextEdit.setObjectName("listLogsTextEdit")
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 968, 22))
self.menubar.setObjectName("menubar")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "Facial modeling"))
    self.convertTo3dButton.setText(_translate("MainWindow", "Convert to 3D"))
    self.groupBox.setTitle(_translate("MainWindow", "Fitting settings"))
    self.label.setText(_translate("MainWindow", "Iterations for rigid fitting"))
    self.label_2.setText(_translate("MainWindow", "Iterations for non-rigid fitting"))
    self.label_3.setText(_translate("MainWindow", "Learning rate for rigid fitting"))
    self.label_4.setText(_translate("MainWindow", "Learning rate for non-rigid fitting"))
    self.groupBox_2.setTitle(_translate("MainWindow", "Processing image"))
    self.imageCaption.setText(_translate("MainWindow", "Processing image"))
    self.loadImageButton.setText(_translate("MainWindow", "Load image"))
    self.groupBox_3.setTitle(_translate("MainWindow", "Losses settings"))

```

```

self.label_10.setText(_translate("MainWindow", "Weight for landmark loss"))
self.label_11.setText(_translate("MainWindow", "Weight for RGB loss"))
self.label_12.setText(_translate("MainWindow", "Weight for ID coef. regularizer"))
self.label_13.setText(_translate("MainWindow", "Weight for expression coef. regularizer"))
self.label_14.setText(_translate("MainWindow", "Weight for texture coef. regularizer"))
self.label_20.setText(_translate("MainWindow", "Weight for texture reflectance loss"))
self.groupBox_4.setTitle(_translate("MainWindow", "Modeling result"))
self.resultImageCaption.setText(_translate("MainWindow", "Rendered 3D face"))
self.listLogsTextEdit.setPlaceholderText(_translate("MainWindow", "Logs"))

```

Б.6 Код для нормування моделі

```

from scipy.io import loadmat, savemat
import numpy as np
from array import array

def LoadExpBasis():
    n_vertex = 53215
    exp_bin = open('BFM/Exp_Pca.bin', 'rb')
    exp_dim = array('i')
    exp_dim.fromfile(exp_bin, 1)
    exp_mu = array('f')
    exp_pc = array('f')
    exp_mu.fromfile(exp_bin, 3*n_vertex)
    exp_pc.fromfile(exp_bin, 3*exp_dim[0]*n_vertex)
    exp_pc = np.array(exp_pc)
    exp_pc = np.reshape(exp_pc, [exp_dim[0], -1])
    exp_pc = np.transpose(exp_pc)
    exp_ev = np.loadtxt('BFM/std_exp.txt')

    return exp_pc, exp_ev

def transferBFMToBaseModel():
    original_bfm = loadmat('BFM/01_MorphableModel.mat')
    shape_pc = original_bfm['shapePC']
    shape_ev = original_bfm['shapeEV']
    shape_mu = original_bfm['shapeMU']
    tex_pc = original_bfm['texPC']
    tex_ev = original_bfm['texEV']
    tex_mu = original_bfm['texMU'] # mean texture
    exp_pc, exp_ev = LoadExpBasis()
    id_base = shape_pc*np.reshape(shape_ev, [-1, 199])
    id_base = id_base/1e5
    id_base = id_base[:, :80]
    ex_base = exp_pc*np.reshape(exp_ev, [-1, 79])
    ex_base = ex_base/1e5
    ex_base = ex_base[:, :64]
    tex_base = tex_pc*np.reshape(tex_ev, [-1, 199])
    tex_base = tex_base[:, :80]
    index_exp = loadmat('BFM/BFM_front_idx.mat')
    index_exp = index_exp['idx'].astype(
        np.int32) - 1
    index_shape = loadmat('BFM/BFM_exp_idx.mat')
    index_shape = index_shape['trimIndex'].astype(
        np.int32) - 1 # starts from 0 (to 53490)
    index_shape = index_shape[index_exp]

```

```

id_base = np.reshape(id_base, [-1, 3, 80])
id_base = id_base[index_shape, :, :]
id_base = np.reshape(id_base, [-1, 80])
tex_base = np.reshape(tex_base, [-1, 3, 80])
tex_base = tex_base[index_shape, :, :]
tex_base = np.reshape(tex_base, [-1, 80])
ex_base = np.reshape(ex_base, [-1, 3, 64])
ex_base = ex_base[index_exp, :, :]
ex_base = np.reshape(ex_base, [-1, 64])
mean_shape = np.reshape(shape_mu, [-1, 3])/1e5
mean_shape = mean_shape[index_shape, :]
mean_shape = np.reshape(mean_shape, [1, -1])
mean_tex = np.reshape(tex_mu, [-1, 3])
mean_tex = mean_tex[index_shape, :]
mean_tex = np.reshape(mean_tex, [1, -1])
other_info = loadmat('BFM/facemodel_info.mat')
frontmask2_idx = other_info['frontmask2_idx']
skinmask = other_info['skinmask']
keypoints = other_info['keypoints']
point_buf = other_info['point_buf']
tri = other_info['tri']
tri_mask2 = other_info['tri_mask2']
savemat('BFM/BFM09_model_info.mat', {'meanshape': mean_shape, 'meantex': mean_tex, 'idBase':
id_base, 'exBase': ex_base, 'texBase': tex_base,
                                     'tri': tri, 'point_buf': point_buf, 'tri_mask2': tri_mask2, 'keypoints': keypoints,
'frontmask2_idx': frontmask2_idx, 'skinmask': skinmask})

if __name__ == '__main__':
    transferBFMToBaseModel()

```

Б.7 Код для нормування моделі

```

from scipy.io import loadmat, savemat
import numpy as np
from array import array

def LoadExpBasis():
    n_vertex = 53215
    exp_bin = open('BFM/Exp_Pca.bin', 'rb')
    exp_dim = array('i')
    exp_dim.fromfile(exp_bin, 1)
    exp_mu = array('f')
    exp_pc = array('f')
    exp_mu.fromfile(exp_bin, 3*n_vertex)
    exp_pc.fromfile(exp_bin, 3*exp_dim[0]*n_vertex)
    exp_pc = np.array(exp_pc)
    exp_pc = np.reshape(exp_pc, [exp_dim[0], -1])
    exp_pc = np.transpose(exp_pc)
    exp_ev = np.loadtxt('BFM/std_exp.txt')
    return exp_pc, exp_ev

def transferBFMToBaseModel():
    original_bfm = loadmat('BFM/01_MorphableModel.mat')
    shape_pc = original_bfm['shapePC']
    shape_ev = original_bfm['shapeEV']
    shape_mu = original_bfm['shapeMU']
    tex_pc = original_bfm['texPC']
    tex_ev = original_bfm['texEV']

```

```

tex_mu = original_bfm['texMU'] # mean texture
exp_pc, exp_ev = LoadExpBasis()
id_base = shape_pc*np.reshape(shape_ev, [-1, 199])
id_base = id_base/1e5
id_base = id_base[:, :80]
ex_base = exp_pc*np.reshape(exp_ev, [-1, 79])
ex_base = ex_base/1e5
ex_base = ex_base[:, :64]
tex_base = tex_pc*np.reshape(tex_ev, [-1, 199])
tex_base = tex_base[:, :80]
index_exp = loadmat('BFM/BFM_front_idx.mat')
index_exp = index_exp['idx'].astype(
    np.int32) - 1

index_shape = loadmat('BFM/BFM_exp_idx.mat')
index_shape = index_shape['trimIndex'].astype(
    np.int32) - 1 # starts from 0 (to 53490)
index_shape = index_shape[index_exp]

id_base = np.reshape(id_base, [-1, 3, 80])
id_base = id_base[index_shape, :, :]
id_base = np.reshape(id_base, [-1, 80])
tex_base = np.reshape(tex_base, [-1, 3, 80])
tex_base = tex_base[index_shape, :, :]
tex_base = np.reshape(tex_base, [-1, 80])
ex_base = np.reshape(ex_base, [-1, 3, 64])
ex_base = ex_base[index_exp, :, :]
ex_base = np.reshape(ex_base, [-1, 64])
mean_shape = np.reshape(shape_mu, [-1, 3])/1e5
mean_shape = mean_shape[index_shape, :]
mean_shape = np.reshape(mean_shape, [1, -1])
mean_tex = np.reshape(tex_mu, [-1, 3])
mean_tex = mean_tex[index_shape, :]
mean_tex = np.reshape(mean_tex, [1, -1])
other_info = loadmat('BFM/facemodel_info.mat')
frontmask2_idx = other_info['frontmask2_idx']
skinmask = other_info['skinmask']
keypoints = other_info['keypoints']
point_buf = other_info['point_buf']
tri = other_info['tri']
tri_mask2 = other_info['tri_mask2']
savemat('BFM/BFM09_model_info.mat', {'meanshape': mean_shape, 'meantex': mean_tex, 'idBase':
id_base, 'exBase': ex_base, 'texBase': tex_base,
                                     'tri': tri, 'point_buf': point_buf, 'tri_mask2': tri_mask2, 'keypoints': keypoints,
'frontmask2_idx': frontmask2_idx, 'skinmask': skinmask})
if __name__ == '__main__':
    transferBFMToBaseModel()

```

Б.8 Головной модуль програми для переведення зображення у 3д модель

```

import pathlib
import time
from shutil import copyfile
from facenet_pytorch import MTCNN
from kernel.EnvOptions import EnvOptions
import cv2
import face_alignment
import numpy as np

```

```

from faceModel.BFMFaceModel import BFMFaceModel
import os
import torch
import kernel.utils as utils
from tqdm import tqdm
import kernel.losses as losses
def logData(data, log_file, ui_app):
    if log_file:
        log_file.write(data + '\n')
    if ui_app:
        ui_app.emitLogEvent(data)
    print(data)
def select_source_for_logging(log_file, ui_app):
    return lambda data: logData(data, log_file, ui_app)
def make_model_by_image(args, viewAppThread=None):
    if not os.path.exists(args.dist_folder):
        os.makedirs(args.dist_folder)
    log_file = open(os.path.join(args.dist_folder, 'logs.txt'), 'w')
    log = select_source_for_logging(log_file, viewAppThread)

    log('Settings:')
    for k, v in sorted(args.items()):
        log('%s: %s' % (str(k), str(v)))
    log("")
    result_rendered_image_path = os.path.join(args.dist_folder, 'rendered_face.jpg')
    copyfile(args.img_path, os.path.join(args.dist_folder, 'original' + pathlib.Path(args.img_path).suffix))
    log('Loading image..')
    start = time.time()
    time_on_start_app = time.time()
    original_image = cv2.imread(args.img_path)[: , : , :-1]
    original_image_height, original_image_width = original_image.shape[:2]
    if viewAppThread:
        cv2.imwrite(result_rendered_image_path, original_image[: , : , :-1])
        viewAppThread.emitRenderedImageUpdated()
    elapsed_image_loading_time = time.time() - start
    log('Image loaded.')
    log('Selected device for computing: %s' % args.device)
    mtcnn = MTCNN(device=args.device, select_largest=False)
    face_alignment_options = face_alignment.FaceAlignment(
        face_alignment.LandmarksType._3D,
        flip_input=False,
        device=args.device,
        face_detector='sfd'
    )
    start = time.time()
    log('Detecting the face in the image..')
    face_boxes, probs = mtcnn.detect(original_image)
    if face_boxes is None:
        log('No face detected.')
        return

    face_box = utils.box_padding(face_boxes[0], (original_image_width, original_image_height))
    face_image = original_image[face_box[1]:face_box[3], face_box[0]:face_box[2], :]
    resized_face_img = cv2.resize(face_image, (args.face_image_size, args.face_image_size))
    if viewAppThread:
        cv2.imwrite(result_rendered_image_path, resized_face_img[: , : , :-1])
        viewAppThread.emitRenderedImageUpdated()

```



```

elapsed_detect_face_time = time.time() - start
log('A face is detected(l:%d,t:%d,r:%d,b:%d).'
    % (face_box[0], face_box[1], face_box[2], face_box[3]))

log('Detecting the face landmarks..')
start = time.time()
face_landmarks = face_alignment_options.get_landmarks_from_image(resized_face_img)[0]
face_landmarks = face_landmarks[:, :2][None, ...]
face_landmarks = torch.tensor(face_landmarks, dtype=torch.float32, device=args.device)
img_tensor = torch.tensor(resized_face_img[None, ...], dtype=torch.float32, device=args.device)

if viewAppThread:
    for pointT in face_landmarks[0]:
        point = pointT.cpu().numpy()
        cv2.circle(resized_face_img, center=(int(point[0]), int(point[1])), radius=2, color=(255, 0, 0),
            thickness=-1)
        cv2.imwrite(result_rendered_image_path, resized_face_img[:, :, :-1])
        viewAppThread.emitRenderedImageUpdated()
elapsed_face_landmarks_time = time.time() - start
log('Face landmarks detected.')

log('Loading BFM models..')
start = time.time()
bfm_face_model = BFMFaceModel(device=args.device, batch_size=1,
img_size=args.face_image_size)
elapsed_load_bfm_model = time.time() - start
log("BFM models loaded.")

log('Start processing rigid fitting..')
start = time.time()
fitting_time = time.time()
landmarks_weights = utils.get_landmarks_weights(args.device)
rigid_optimizer = torch.optim.Adam([bfm_face_model.get_rot_tensor(),
bfm_face_model.get_trans_tensor()],
    lr=args.learning_rigid_rate)

if viewAppThread:
    viewAppThread.emitSetProgressBarRange(args.rigid_iterations)
for i in tqdm(range(args.rigid_iterations), colour='#8bc34a'):
    if viewAppThread:
        viewAppThread.emitSetProgressBarValue(i + 1)
    rigid_optimizer.zero_grad()
    pred_dict = bfm_face_model(bfm_face_model.get_packed_tensors(), render=False)
    lm_loss_val = losses.get_landmarks_loss(
        pred_dict['lms_proj'], face_landmarks, landmarks_weights, img_size=args.face_image_size)
    total_loss = args.landmarks_loss_weight * lm_loss_val
    total_loss.backward()
    rigid_optimizer.step()
    if viewAppThread and i % 10 == 0:
        save_face(bfm_face_model, result_rendered_image_path, args, False)
        viewAppThread.emitRenderedImageUpdated()

if viewAppThread:
    save_face(bfm_face_model, result_rendered_image_path, args, False)
    viewAppThread.emitRenderedImageUpdated()
elapsed_rigid_fitting_time = time.time() - start
log('Rigid fitting done. Landmarks loss: %f.' % (lm_loss_val.detach().cpu().numpy()))

```

```

log('Start processing non-rigid fitting..')
start = time.time()
nonrigid_optimizer = torch.optim.Adam(
    [bfm_face_model.get_id_tensor(), bfm_face_model.get_exp_tensor(),
     bfm_face_model.get_gamma_tensor(), bfm_face_model.get_tex_tensor(),
     bfm_face_model.get_rot_tensor(), bfm_face_model.get_trans_tensor()],
    lr=args.learning_nonrigid_rate
)
if viewAppThread:
    viewAppThread.emitSetProgressBarRange(args.nonrigid_iterations)
for i in tqdm(range(args.nonrigid_iterations), colour='#8bc34a'):
    if viewAppThread:
        viewAppThread.emitSetProgressBarValue(i + 1)
    nonrigid_optimizer.zero_grad()

    pred_dict = bfm_face_model(bfm_face_model.get_packed_tensors(), render=True)
    rendered_img = pred_dict['rendered_img']

    mask = rendered_img[:, :, :, 3].detach()
    photo_loss_val = losses.get_photo_loss(rendered_img[:, :, :, :3], img_tensor, mask > 0)
    lm_loss_val = losses.get_landmarks_loss(pred_dict['lms_proj'], face_landmarks, landmarks_weights,
                                           img_size=args.face_image_size)
    id_reg_loss = losses.get_l2(bfm_face_model.get_id_tensor())
    exp_reg_loss = losses.get_l2(bfm_face_model.get_exp_tensor())
    texture_reg_loss = losses.get_l2(bfm_face_model.get_tex_tensor())
    texture_loss_weight = losses.reflectance_loss(pred_dict['face_texture'],
    bfm_face_model.get_skinmask())

    loss = lm_loss_val * args.landmarks_loss_weight + \
        id_reg_loss * args.id_reg_weight + \
        exp_reg_loss * args.expressions_reg_weight + \
        texture_reg_loss * args.texture_reg_weight + \
        texture_loss_weight * args.texture_loss_weight + \
        photo_loss_val * args.rgb_loss_weight

    loss.backward()
    nonrigid_optimizer.step()
    if viewAppThread and i % 10 == 0:
        save_face(bfm_face_model, result_rendered_image_path, args, False)
        viewAppThread.emitRenderedImageUpdated()
elapsed_nonrigid_fitting_time = time.time() - start
log('Non-rigid fitting done.')

save_face(bfm_face_model, result_rendered_image_path, args, True)
if viewAppThread:
    viewAppThread.emitRenderedImageUpdated()
elapsed_total_time = time.time() - time_on_start_app
log('Rendered image saved at %s' % args.dist_folder)
log('\nAnalytics:')
log('Elapsed times:\n'
    'Image loading: %f\n'
    'Find face: %f\n'
    'Find face landmarks: %f\n'
    'Loading BFM model: %f\n'
    'Rigid fitting: %f\n'
    'Non-rigid fitting: %f\n'
    'Total: %f % (

```

```

        elapsed_image_loading_time,
        elapsed_detect_face_time,
        elapsed_face_landmarks_time,
        elapsed_load_bfm_model,
        elapsed_rigid_fitting_time,
        elapsed_nonrigid_fitting_time,
        elapsed_total_time
    )
)
log_file.close()

def save_face(face_model, result_rendered_image_path, options, save_obj):
    with torch.no_grad():
        coeffs = face_model.get_packed_tensors()
        pred_dict = face_model(coeffs, render=True)
        rendered_img = pred_dict['rendered_img']
        rendered_img = rendered_img.cpu().numpy().squeeze()
        out_img = rendered_img[:, :, :3].astype(np.uint8)
        cv2.imwrite(result_rendered_image_path, out_img[:, :, ::-1])
        if save_obj:
            vs = pred_dict['vs'].cpu().numpy().squeeze()
            tri = pred_dict['tri'].cpu().numpy().squeeze()
            color = pred_dict['color'].cpu().numpy().squeeze()
            utils.export_to_obj_format(os.path.join(options.dist_folder, 'face_model.obj'), vs, tri + 1, color)
if __name__ == '__main__':
    envOptions = EnvOptions()
    envOptions.initialize_options()
    envOptions = envOptions.parse_settings()
    envOptions.device = 'cuda' if torch.cuda.is_available() else 'cpu'
    make_model_by_image(envOptions)

```

Додаток В
Матеріали презентації

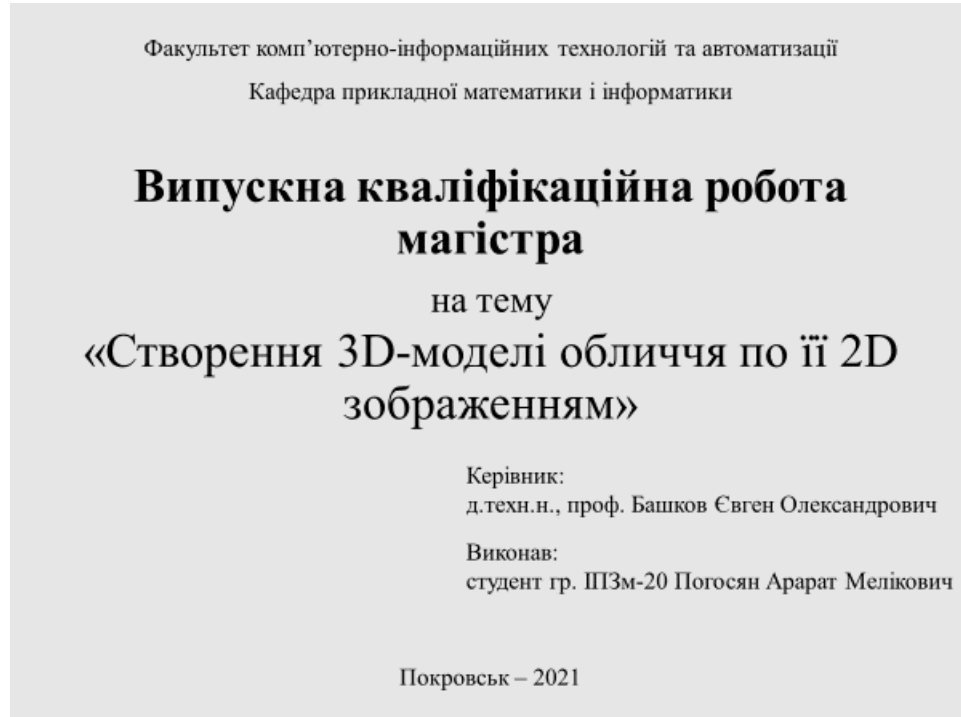


Рисунок В.1 – Слайд 1

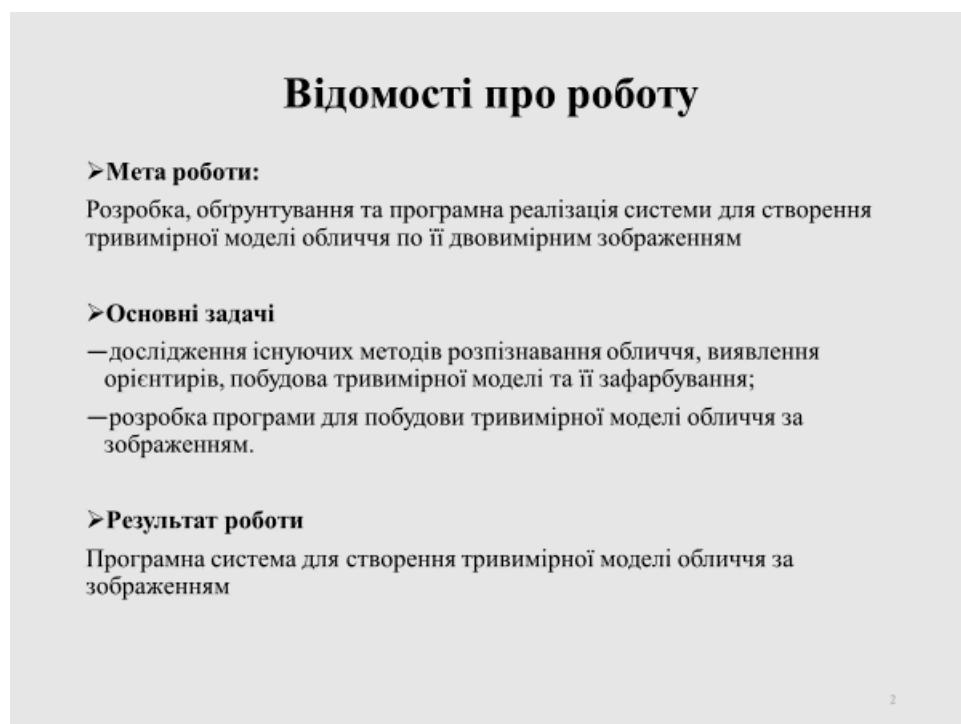


Рисунок В.2 – Слайд 2

Актуальність, практичне значення

- Незважаючи на різноманітність методів побудови та розпізнавання тривимірного обличчя, вони все ще мають такі недоліки, як складність і дороге впровадження. У зв'язку з цим виникає необхідність у розробці нових підходів, які спрощують побудову тривимірних моделей осіб та роблять розпізнавання більш точним.
- за рахунок розвитку ігрової індустрії та доповненої реальності система реконструкції обличчя із зображення є актуальними

3

Рисунок В.3 – Слайд 3

Кроки створення тривимірної моделі обличчя із зображення

- Виявлення обличчя на зображення.
- Вилучення орієнтирів обличчя.
- Грубе наближення 3D морфної моделі до знайдених орієнтирів.
- Точне наближення 3D морфної моделі до знайдених орієнтирів.
- Текстурування(розмальовка) отриманої моделі.



4

Рисунок В.4 – Слайд 4

Кроки створення тривимірної моделі обличчя із зображення

- На основі знань
- PCA
- PCA з використанням штучних нейронних мереж
- Штучна бджолина колонія
- Генетичний алгоритм
- Оптимізація рою частинок
- Deep Dense Face Detector
- Radial Basis Function Neural Networks
- Convolutional Neural Network Cascade
- Bilinear CNNs
- **Multi-task Cascaded Convolutional Networks**

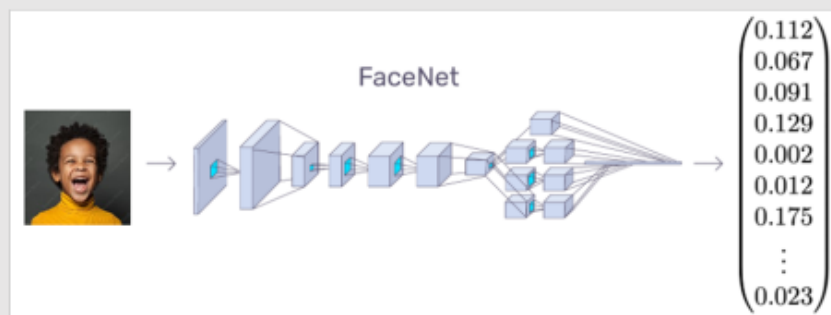
5

Рисунок В.5 – Слайд 5

Multi-task Cascaded Convolutional Networks (MTCNN)

MTCNN – це багатозадачна каскадна згорткова нейронна мережа, із модуля FaceNet від Google. FaceNet – це архітектура глибокого машинного навчання, що складається з згорткових шарів, FaceNet повертає 128-мірний вектор для кожного обличчя.

FaceNet — це кілька нейронних мереж та набір алгоритмів для підготовки та обробки проміжних результатів роботи цих мереж



6

Рисунок В.6 – Слайд 6

Кроки виконання MTCNN

1. Створення безлічі розмірів нашої фотографії, для збільшення шансів на вірне розпізнавання обличчя на фотографії.
2. Мережа P-Net визначає координати осіб та рівень ймовірності для кожного блоку, прибираючи блоки нижче за порогове значення ймовірності.
3. Мережа R-Net виконує відбір найбільш відповідних блоків, які найімовірніше є обличчям.
4. O-Net мережа формує п'ять точок для кожного блоку (очі, ніс, кути губ), якщо ці точки повністю потрапляють до блоку, то найімовірніше цей блок містить обличчя.

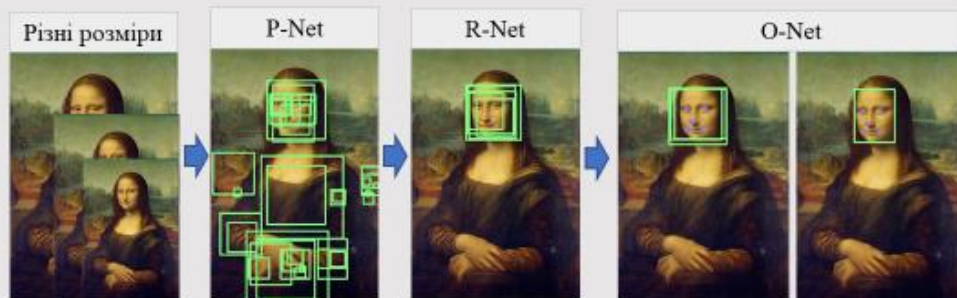


Рисунок В.7 – Слайд 7

Виявлення орієнтирів на обличчі

Орієнтири обличчя – це візуально помітні точки на обличчі, такі як кінець носа, кінці брів і рота.

На такі орієнтири, як куточки очей або кінчик носа, вираз обличчя мало впливає, тому вони більш надійні і насправді називаються опорними точками.

Орієнтири на обличчі можуть спостерігатися лише частково через оклюзію волосся, рухи рук або самооклюзію або через інтенсивні обертання голови.

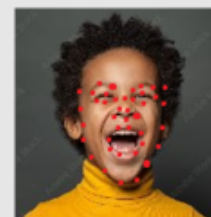


Рисунок В.8 – Слайд 8

Метод виявлення орієнтирів на обличчі

Для знаходження орієнтирів обличчя використовується глибока нейронна мережа FAN яка має хорошу продуктивність для вилучення лицьових орієнтирів, датасет навчання включає ~230000 зображень. Орієнтири можуть бути знайдені для великих обертань голови (60 – 90)



Рисунок В.9 – Слайд 9

Методи створення 3D моделі за допомогою 3DMM

➤ метод на основі фітінга морфированної моделі (3DMM)

За рахунок апроксимації довільні моделі зменшуються до необхідного розміру заздалегідь підготовленої бази, а висока деталізація вихідних моделей дозволяє синтезувати реалістичні обличчя.

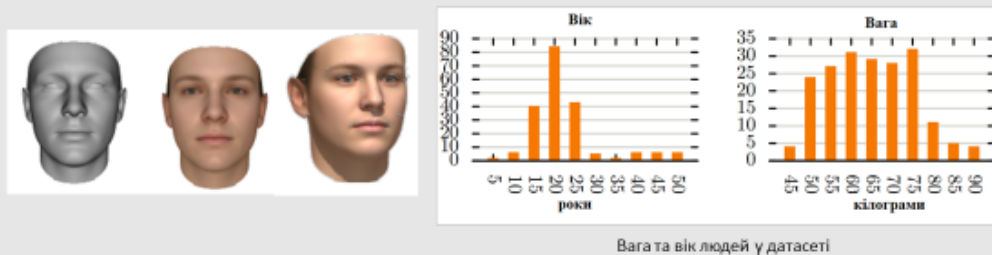
➤ метод на основі згорткової нейронної мережи (CNN)

Алгоритм глибокого навчання, основна ідея якої полягає у чергуванні згорткових нейронних мереж, вимагає великих обсягів даних для навчання та відповідний час на навчання.

Рисунок В.10 – Слайд 10

Тривимірна морфована модель обличчя (3DMM)

- Статистична модель структури та текстури обличчя
- Отримано завдяки скануванням обличчя спеціальним обладнанням
- Обробка
 - Заповнення отворів
 - Згладжування поверхні
 - Оцінка альbedo

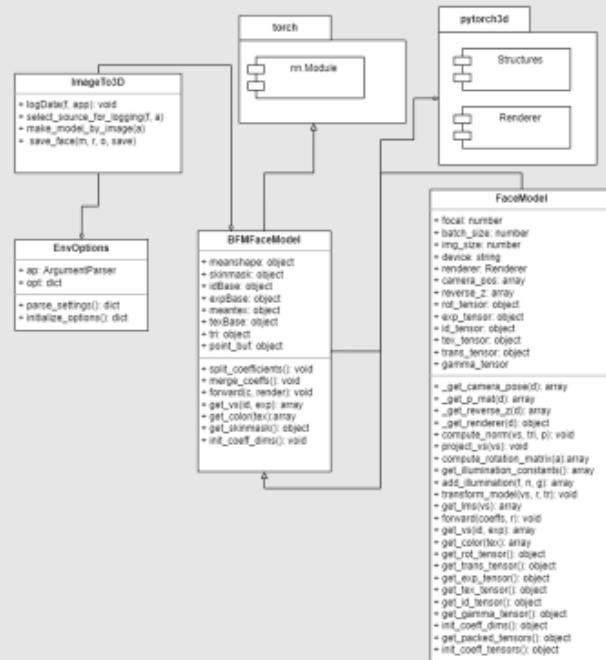


11

Рисунок В.11 – Слайд 11

Проектування системи

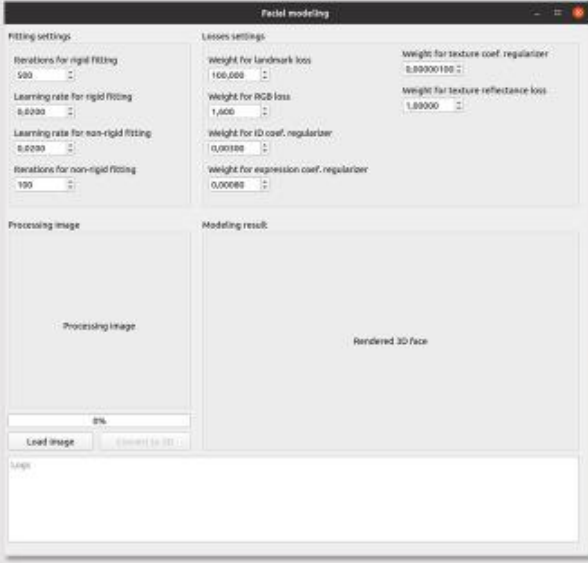
- модульна архітектура;
- замінність модулів;
- контроль версій;
- зручне тестування;
- можливість додавання моделей з різних датасетів шляхом успадкування.



12

Рисунок В.12 – Слайд 12


Розробка системи



Графічний додаток

Засоби розробки:

- Python 3.8
- PyQt
- PyTorch
- numPy
- face_alignment
- OpenCV
- mtcnn
- facenet_pytorch



Консольний додаток

Рисунок В.13 – Слайд 13

Тестування системи

За результатами моделювання обличчя при використанні 100 ітерацій для функції оптимізації Адама було витрачено 13 секунд.

Етап	Ітерації	100
Завантаження зображення		0.038622
Пошук обличчя		0.085725
Пошук орієнтирів		0.634996
Завантаження моделі BFM		0.103088
Жорстка підгінка моделі		3.476556
Нежорстка підгінка моделі		8.776686


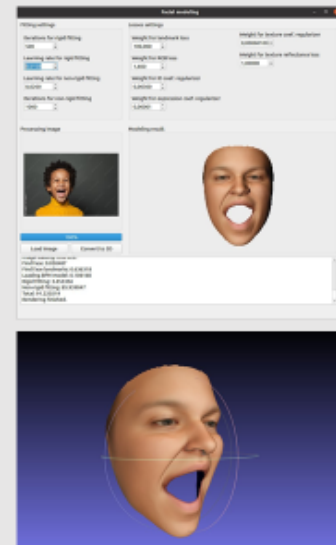


Рисунок В.14 – Слайд 14

Аналіз результатів роботи

- Побудовано архітектуру програмної системи.
- Обрано найбільш оптимальні методи для розпізнавання та вилучення орієнтирів обличчя а також метод створення моделі з використанням морфної моделі.
- Розглянуто методи реконструкції моделі обличчя на основі 3DMM або CNN, а також особливості кожного з них.
- Проведено аналіз та тестування роботи програмного забезпечення.
- Завдяки використанню графічного інтерфейсу робота з програмою не потребує особливої підготовки та вміння працювати з терміналом.



15

Рисунок В.15 – Слайд 15

Приклад створення тривимірної моделі обличчя



16

Рисунок В.16 – Слайд 16

Приклад створення тривимірної моделі обличчя



17

Рисунок В.17 – Слайд 17

Приклад створення тривимірної моделі обличчя



18

Рисунок В.18 – Слайд 18

Дякую за увагу!

16

Рисунок В.19 – Слайд 19