**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ**
**ІГОРЯ СІКОРСЬКОГО»**
Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____2024р.

# Дипломна робота

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»**
**спеціальності 121 «Інженерія програмного забезпечення»**
**на тему: «Розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача»**

Виконав:
Студент IV курсу, групи ТВ-11
Гришай Даніїл Дмитрович _____ (підпис)

Керівник:

Доцент, к.т.н. Залевська О.В _____ (підпис)
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент:

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

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем в енергетиці»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
(підпис)
«____»_____202_р.

# ЗАВДАННЯ
**на дипломну роботу студенту**

Гришай Даніїлу Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи
«Розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача» керівник роботи Доцент, к.т.н. Залевська Ольга Валеріївна
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "__"_____2025 року №_____
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи: мови програмування Python, бібліотеки MediaPipe, OpenCV, JSON, PythonAPI, середовища розробки Visoal Studio Code, Blender.

4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити): розробка програмних модулів для зчитування міміки користувача з відео, обробка та нормалізація параметрів виразів обличчя, формування структурованих даних для передачі. Інтеграція цих даних із параметричними моделями обличчя для динамічного відтворення міміки.
5. Перелік ілюстративного матеріалу: приклади структурованих вихідних даних у форматі JSON; математичні формули нормалізації та обробки виразів обличчя.

6. Дата видачі завдання «__» _____202_ р.

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Строки виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 30.10.2024 | виконано |
| 2 | Дослідження предметної області | 31.10.2024 - 01.11.2024 | виконано |
| 3 | Дослідження існуючих рішень | 02.11.2024 – 01.12.2024 | виконано |
| 4 | Постановка вимог до проєктування системи | 02.12.2024 – 12.01.2025 | виконано |
| 5 | Розробка програмного продукту | 13.01.2025 –11.05.2025 | виконано |
| 6 | Тестування | 12.05.2025 – 15.05.2025 | виконано |
| 7 | Захист програмного продукту | 15.05.2025 | виконано |
| 8 | Оформлення дипломної роботи | 19.05.2025 – 01.06.2025 | виконано |
| 9 | Передзахист | 02.06.2025 | виконано |
| 10 | Захист | 19.06.2025 | виконано |

Студент _____ Даніїл Гришай
( підпис ) ( ім'я, прізвище)

Керівник роботи _____ Ольга Зелевська
( підпис ) (ім'я, прізвище)

# РЕФЕРАТ

**Структура та обсяг дипломної роботи.** Робота містить 53 сторінки, 11 рисунків, 2 додатки,13 формул та 7 посилань.

**Метою роботи** є розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача в реальному часі.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано сучасні підходи до захоплення та передачі міміки обличчя;
- реалізовано модуль для зчитування міміки з відео;
- розроблено алгоритми нормалізації параметрів міміки відповідно до індивідуальних особливостей користувача;
- створено структуру для передачі даних у форматі JSON;
- впроваджено скрипт у середовищі Blender;
- забезпечено згладжування і стабілізацію переданих рухів.

**Практичне значення одержаних результатів** полягає в створенні інструменту, що може бути використаний у системах віртуальної комунікації, інтерактивних додатках, віртуальній реальності, комп'ютерній анімації та дослідженнях взаємодії людини з комп'ютером.

**Ключові слова:** передача міміки, параметрична модель обличчя, MediaPipe, Blender, shape keys, JSON, комп'ютерний зір, анімація в реальному часі.

# ABSTRACT

**Structure and scope of the thesis.** The work contains 53 pages, 11 figures, 2 appendices,13 formulas and 7 references.

**The purpose of the work** is to develop software modules for integrating parametric face models with a real-time facial expression transfer system.

To achieve the set goal, the following tasks were completed:

• analyzed modern approaches to facial expression capture and transfer;

• implemented a module for reading facial expressions from video;

• developed normalization algorithms for expression parameters based on individual user features;

• designed a data transmission structure using the JSON format;

• created a script within the Blender environment;

**The practical value** of the obtained results lies in the creation of a tool that can be used in virtual communication systems, interactive applications, virtual reality, computer animation, and human-computer interaction research.

**Keywords**: facial expression transfer, parametric face model, MediaPipe, Blender, shape keys, JSON, computer vision, real-time animation.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

API (Application Programming Interface) — інтерфейс програмного застосування, який використовується для взаємодії між модулями системи.

CV (Computer Vision) — комп’ютерний зір, галузь, яка охоплює розпізнавання обличчя, міміки тощо.

JSON (JavaScript Object Notation) — формат зберігання структурованих даних між модулями системи.

MediaPipe — бібліотека для зчитування координат ключових точок обличчя з відео

OpenCV — бібліотека для обробки зображень і відеопотоку, що використовується у модулі захоплення міміки.

VSC (Visual Studio Code) — редактор вихідного коду, розроблений Microsoft для Windows, Linux та macOS.

Shape key — параметр морфінгу для зміни форми 3D-моделі

# ВСТУП

У сучасному цифровому просторі дедалі більшого значення набуває інтерактивна взаємодія між користувачем і віртуальним середовищем. Одним з найперспективніших напрямів розвитку цієї взаємодії є відтворення емоцій та міміки людини на віртуальних персонажах. Це має широке застосування — від створення анімованих персонажів у фільмах та іграх до віртуальних асистентів, дистанційної освіти та телемедицини. Емоційна достовірність віртуальних персонажів значною мірою залежить від здатності системи точно передати міміку реальної людини в режимі реального часу.

Передача міміки передбачає складну послідовність операцій: виявлення ключових точок обличчя на відео, обробку параметрів виразів, їхнє згладжування та нормалізацію, а також подальшу трансляцію цих параметрів на модель обличчя. Особливу складність становить адаптація міміки під конкретну модель з урахуванням індивідуальних особливостей обличчя користувача. Поширеним підходом до вирішення цих завдань є використання параметричних моделей обличчя, що дозволяють керувати окремими елементами (губи, очі, брови) через набір керованих параметрів або shape keys.

Сучасні засоби комп’ютерного зору, зокрема бібліотека MediaPipe, відкривають можливості для детального трекінгу міміки без потреби в глибокому навчанні нейромереж. У поєднанні з інструментами середовища Blender, ці технології дозволяють створити систему для анімації міміки на параметричних моделях.

Метою даної роботи є розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача в реальному часі. Поставлена мета охоплює створення системи, що включає модуль захоплення міміки з відеопотоку, обробку й нормалізацію параметрів, структуру збереження даних (JSON), та модуль відтворення міміки у Blender.

В ході цієї роботи було проведено огляд сучасних підходів до захоплення та передачі міміки обличчя, зосереджено увагу на методах зчитування виразів у режимі реального часу. На основі цього аналізу реалізовано модуль, що зчитує координати ключових точок обличчя з відеопотоку з використанням бібліотеки MediaPipe, а також розроблено алгоритми обробки, згладжування та масштабування мімічних параметрів відповідно до індивідуальних особливостей користувача. Отримані дані інтегруються з параметричною моделлю обличчя за допомогою скрипта, реалізованого у середовищі Blender, що дозволяє відтворювати міміку в динаміці.

Практичне значення даної роботи полягає у створенні ефективного й адаптивного рішення, яке може бути використане для керування віртуальними персонажами без використання складних нейронних мереж або дорогого апаратного забезпечення. Розроблена система демонструє можливість точної передачі міміки в реальному часі та має потенціал до подальшого розширення.

# 1 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЕРЕДАЧІ МІМІКИ НА ПАРАМЕТРИЧНУ МОДЕЛЬ ОБЛИЧЧЯ

## 1.1 Постановка задачі

Завданням цієї роботи є розробка програмного рішення, яке дозволить здійснювати передачу міміки користувача на параметричну модель обличчя в режимі реального часу. Така система повинна буде забезпечити зчитування виразів обличчя з відеопотоку, обробку отриманих даних та їхнє застосування до віртуальної моделі з метою реалістичного відтворення міміки.

Для цього необхідно реалізувати модуль захоплення міміки, який отримуватиме координати ключових точок з обличчя користувача за допомогою MediaPipe, також потрібно розробити алгоритми, що будуть забезпечувати згладжування, нормалізацію і масштабування параметрів міміки відповідно до індивідуальних калібрувань. Ці дані зберігатимуться у форматі JSON для подальшого використання іншими компонентами системи.

Окремим завданням є створення програмного модуля у середовищі Blender, який зчитує збережені параметри та керує мімікою параметричної моделі обличчя, використовуючи shape keys та/або кісткову анімацію. Важливо забезпечити синхронність і стабільність відображення міміки без ривків, затримок та артефактів.

Результатом виконання задачі є створення програмної системи, яка забезпечує передачу міміки користувача на параметричну модель обличчя з використанням структурованих даних. Така система дозволяє дослідити можливості інтеграції трекінгу міміки в цифрові моделі та може бути основою для подальших розробок у галузі візуалізації, анімації та взаємодії людини з комп’ютером.

## 1.2 Аналіз існуючих рішень та їх особливості

Завдання розробки системи передачі міміки користувача на параметричну модель обличчя передбачає вибір відповідних технологій, які дозволять реалізувати зчитування міміки в реальному часі, обробку отриманих даних та їх застосування до віртуального обличчя. Для цього доцільно проаналізувати переваги й недоліки популярних мов програмування та бібліотек комп'ютерного зору.

Одним із найбільш поширених варіантів для таких задач є Python. Його популярність зумовлена великою кількістю готових бібліотек, простим синтаксисом, активною спільнотою та підтримкою різних інструментів комп'ютерного зору. Недоліком Python є відносно низька швидкодія, особливо в задачах обробки великих потоків даних у порівнянні з C++.

C++, у свою чергу, відзначається високою продуктивністю та ефективністю роботи з ресурсами, однак має складніший синтаксис і вимагає більше часу на розробку. Також для задач комп'ютерного зору він потребує використання низькорівневих бібліотек, що ускладнює реалізацію.

JavaScript, зокрема у поєднанні з бібліотекою TensorFlow.js, дозволяє виконувати трекінг обличчя у браузері, проте можливості такого підходу є обмеженими: точність нижча, а продуктивність залежить від браузера й не завжди стабільна для задач реального часу.

Щодо вибору бібліотек для трекінгу, серед найбільш популярних варто відзначити MediaPipe — інструмент від Google, що надає попередньо натреновані моделі розпізнавання обличчя та міміки. Альтернативними підходами можуть бути більш складні рішення на основі TensorFlow або OpenFace, але вони потребують навчання моделей та високих обчислювальних ресурсів.

Для обробки відео часто використовуються OpenCV або власні засоби роботи з відеопотоком. OpenCV підтримується на багатьох мовах і надає доступ до базових операцій зображення, однак не містить вбудованих моделей для мімічного аналізу.

У візуалізації міміки можуть застосовуватись як великі рушії, такі як Unity або Unreal Engine, так і 3D-середовища з відкритим кодом на зразок Blender, яке підтримує керування параметрами обличчя через shape keys або кістки. Blender відзначається гнучкістю завдяки підтримці Python API, однак вимагає окремого налаштування під конкретну модель.

Таким чином, для реалізації системи передачі міміки можливо обирати з кількох технічних стеків, кожен з яких має свої сильні сторони залежно від вимог до продуктивності, точності, швидкості розробки та гнучкості візуалізації.

## 1.3 Вибір технологій і засобів реалізації

Найбільш доцільним варіантом реалізації системи передачі міміки користувача був обраний Python, який, попри обмежену швидкодію у порівнянні з мовами нижчого рівня, має потужний набір бібліотек і значні можливості для швидкої реалізації прикладних рішень у сфері комп’ютерного зору. Простий синтаксис, велика спільнота та готові інструменти роблять Python зручним для реалізації як етапу обробки відео, так і формування вихідних даних для подальшої візуалізації.

Для зчитування міміки обличчя було використано бібліотеку MediaPipe, яка надає попередньо натреновану модель Face Mesh. Вона дозволяє отримувати координати ключових точок обличчя з не поганою точністю в режимі реального часу без необхідності навчання моделі. Це рішення значно спрощує процес інтеграції трекінгу міміки в систему.

Захоплення відео з камери та передача зображень у трекінговий модуль здійснюється за допомогою бібліотеки OpenCV, яка надає інтерфейс для роботи з відеопотоком, а також базові функції для попередньої обробки кадрів (зміна кольорової моделі, виведення зображення на екран тощо).

Для передачі результатів обробки міміки між компонентами системи було обрано формат JSON, що дозволяє зберігати структуровані дані у зрозумілому

вигляді та легко передавати їх між Python-модулем і середовищем 3D-анімації.

Для відтворення міміки на віртуальній моделі використовується Blender — потужне середовище 3D-моделювання з відкритим кодом та підтримкою Python API, яке дозволяє автоматизувати керування мімічними параметрами за допомогою скриптів. У проєкті використано як shape keys, так і кісткову анімацію для точного відображення виразів обличчя.

Таким чином, вибір технологічного стеку на основі Python, MediaPipe, OpenCV, JSON і Blender дозволив реалізувати функціональну, доступну до розширення та ресурсоефективну систему передачі міміки в режимі реального часу.

## 1.4 Основні поняття міміки, параметризації та моделювання обличчя

Міміка — це комплекс рухів м’язів обличчя, що відображає емоційний стан або комунікативний намір людини. У цифровому середовищі міміка розглядається як зміна просторового положення окремих ділянок обличчя, яку можна відслідковувати, описувати та моделювати за допомогою математичних або структурних параметрів.

У системах CV (Computer Vision) міміка зазвичай описується через ключові точки обличчя — заздалегідь визначені орієнтири, які відповідають анатомічно важливим ділянкам (наприклад, кути рота, повіки, краї брів). Зміна просторового розташування цих точок дозволяє чисельно описати мімічні вирази.

Для забезпечення керованості та гнучкості при відтворенні виразів використовується поняття параметризації обличчя — подання його стану через набір числових параметрів, які відповідають окремим мімічним характеристикам (наприклад, ступінь відкриття рота або підняття брів). Таке подання дає змогу формалізувати процес передачі міміки та використовувати ці параметри незалежно від конкретного користувача чи моделі.

Параметрична модель обличчя — це цифрове представлення форми обличчя,

яке може змінюватися під впливом керуючих параметрів. Така модель побудована таким чином, щоб відповідати певній нейтральній формі і реагувати на зміну виразів обличчя шляхом геометричних деформацій. До основних способів відображення міміки входять, морфінг (shape keys), технологія що дозволяє зберігати альтернативні геометричні конфігурації віртуальної моделі (таких як усмішка, сум) і керувати їх змішуванням. А також кісткова аніімація, це метод, при якому міміка базується на переміщенні скелетної структури моделі

Усі ці поняття є фундаментальними для побудови системи передачі міміки — від моменту захоплення виразів користувача до їх відтворення на цифровій моделі.

## Висновки до розділу 1

Як висновок можна підсумувати, що дана робота має на меті створення програмної системи, яка дозволить у режимі реального часу передавати міміку користувача на параметричну модель обличчя. Основні функції такої системи включають зчитування координат ключових точок обличчя з відеопотоку, обробку отриманих мімічних параметрів із урахуванням індивідуальних особливостей користувача, а також передачу даних у форматі, зручному для їх подальшого використання у 3D-середовищі.

Аналіз існуючих рішень показав, що кожна з розглянутих технологій має свої переваги та недоліки. Серед мов програмування було розглянуто Python, C++ та JavaScript. Найбільш оптимальним варіантом для реалізації проєкту виявився Python, який, незважаючи на меншу швидкодію, забезпечує високу гнучкість розробки, багатий набір бібліотек і активну підтримку спільноти. Розглянуті альтернативи, такі як C++ або JavaScript, мають свої сильні сторони, однак у контексті даної задачі Python продемонстрував найкраще співвідношення зручності, продуктивності й можливості швидкої інтеграції з іншими інструментами.

У якості засобу трекінгу міміки було обрано MediaPipe Face Mesh, що

дозволяє зчитувати координати до 468 ключових точок обличчя без необхідності самостійного навчання моделі. Для захоплення відео з камери використано OpenCV, який забезпечує просту обробку кадрів у реальному часі. Обмін мімічними параметрами між модулями реалізовано за допомогою формату JSON, що забезпечує зручність та прозорість структури даних. Візуалізація результатів здійснюється в середовищі Blender, яке підтримує використання shape keys і кісткової анімації, що дозволяє точно передавати вирази обличчя моделі.

Також було розглянуто основні поняття, пов’язані з цифровим описом міміки, такі як параметризація обличчя, ключові точки, параметричні 3D-моделі, shape keys та кісткова анімація. Ці поняття є фундаментальними для побудови системи, яка має на меті забезпечити точне, гнучке та стабільне відтворення міміки користувача у цифровому середовищі.

Загалом, створена концепція програмної системи дозволяє реалізувати ефективну та розширювану платформу для передачі міміки, що може знайти застосування в галузях віртуальної анімації, ігрової індустрії, дистанційної комунікації та інтерактивних мультимедійних систем. Обрані технології забезпечують необхідний рівень функціональності, точності та адаптивності, що відкриває можливості для подальшого розвитку й вдосконалення системи.

# 2 АНАЛІЗ ІНСТРУМЕНТІВ ТА СЕРЕДОВИЩ РОЗРОБКИ

## 2.1 Вибір платформи розробки та мови програмування

Як вже сказано для реалізації системи передачі міміки було обрано мову програмування Python. Основними бібліотеками стали MediaPipe для зчитування координат ключових точок обличчя, OpenCV для роботи з відеопотоком, та JSON для обміну структурованими даними між модулями.

Для візуалізації міміки у 3D середовищі використано Blender, який підтримує керування параметричною моделлю обличчя через Python API.

### 2.1.1 Мова програмування Python

Python — це високорівнева мова програмування загального призначення, розроблена Гвідо ван Россумом у 1991 році. Вона отримала широке розповсюдження завдяки простому й зрозумілому синтаксису, потужним стандартним бібліотекам та активній спільноті. Python підтримує кілька парадигм програмування, зокрема об’єктно-орієнтовану, процедурну та функціональну, що робить її гнучким інструментом для вирішення різноманітних задач.

Основні особливості мови Python:

- Інтерпретована мова — не потребує компіляції; програми виконуються безпосередньо з вихідного коду;
- Кросплатформеність — Python підтримується на більшості сучасних операційних систем: Linux, Windows, macOS тощо;
- Відкритий код та вільна ліцензія — Python є безкоштовною мовою програмування з відкритим вихідним кодом;
- Синтаксис, близький до природної мови — читається легко та швидко опановується;
- Автоматичне управління пам’яттю — спрощує розробку та виключає

багато типових помилок;

- Розвинена стандартна бібліотека — включає інструменти для роботи з файлами, мережею, багатопоточністю, форматами даних, регулярними виразами та іншим.

Python активно використовується в наукових обчисленнях, аналізі даних, машинному навчанні, автоматизації, веброзробці та створенні графічних застосунків. У контексті цієї роботи Python був обраний як основна мова реалізації завдяки його здатності поєднувати роботу з відеопотоком, обробку структурованих даних і керування 3D-моделями у єдиному програмному середовищі.

Крім того, доступність розширень і бібліотек, таких як MediaPipe, OpenCV, json, NumPy, а також повноцінна інтеграція з Blender API, дозволили реалізувати всі необхідні модулі системи в рамках одного технологічного стеку. Висока продуктивність при низькому порозі входу робить Python зручним вибором як для прототипування, так і для повноцінного розгортання готового рішення.

### 2.1.2 Бібліотека MediaPipe

MediaPipe —це кросплатформенна бібліотека з відкритим кодом, розроблена в дослідницькому підрозділі Google Research. В перше вона була представлена публіці у 2019 році як відповідь на потребу в створенні ефективних, реального часу рішень у сфері комп’ютерного зору для мобільних пристроїв та веб-платформ. Основна ідея MediaPipe полягає в об'єднанні потужних алгоритмів розпізнавання в єдину гнучку структуру, яка дозволяє розробникам швидко створювати, змінювати а також тестувати складні графи обробки відеоданих.

Ця бібліотека надає зручну архітектуру для обробки мультимедійних потоків, де кожен окремий етап реалізується як незалежний вузол. Така побудова дозволяє ефективно використовувати ресурси системи та досягати стабільної роботи навіть на пристроях з обмеженими обчислювальними можливостями. На відміну від більшості важких моделей deep learning, MediaPipe оптимізована для роботи на

CPU в реальному часі, що робить її ідеальним вибором для проєктів орієнтованих на інтерактивність без прив'язки до серверної інфраструктури або потужних GPU.

Особливу увагу в бібліотеці MediaPipe приділено задачам розпізнавання та трекінгу ключових елементів людського тіла, таких як рук, очей і обличчя. У цьому проєкті було використано модуль MediaPipe Face Mesh, який дозволяє зчитувати до 468 тривимірних координат ключових точок обличчя з відеопотоку в реальному часі. Цей модуль базується на вдосконалених CNN-моделях та механізмах регресії позицій, що забезпечує високу точність навіть при частковому повороті голови чи зміні освітлення.

Історично, подібні технології раніше були доступні лише через складні системи з використанням marker-based трекінгу (наприклад, з використанням систем типу Vicon або Faceware), які вимагали апаратного забезпечення, маркерів на обличчі та спеціального освітлення. MediaPipe зробила подібні можливості доступними широкому колу користувачів, у тому числі на звичайних ноутбуках або смартфонах.

У межах цієї роботи MediaPipe використовується як головний

елемент модуля захоплення міміки. Саме ця бібліотека забезпечує стабільне і швидке зчитування координат ключових точок обличчя користувача з потоку камери. Отримані координати обробляються у Python-скрипті, вони проходять нормалізацію, згладжування, адаптацію до нейтрального стану користувача, і зберігаються у структурованому вигляді у форматі JSON. У подальшому ці дані передаються в Blender для створення реалістичної анімації обличчя персонажа в режимі реального часу.

Таким чином, MediaPipe у цьому проєкті є основною обчислювальною базою, яка дозволяє побудувати систему передачі міміки без глибокого залучення нейромереж чи ресурсомістких методів. Її відкритість і гнучкість, грає на руку невеликим розробникам, або студентам с подібним завданням.

### 2.1.3 OpenCV та допоміжні засоби

OpenCV (Open Source Computer Vision Library) — це потужна бібліотека з відкритим кодом, призначена для комп'ютерного зору та обробки зображень. Вона була створена компанією Intel у 1999 році та з того часу набула величезної популярності серед дослідників, розробників і інженерів. Основною метою OpenCV є надання широкого набору інструментів, які дозволяють розпізнавати та аналізувати зображення й відео, здійснювати трекінг об'єктів, виділення контурів, фільтрацію, геометричні перетворення та багато інших операцій. Середовище активно розвивається, має відкриту спільноту та підтримку на різних мовах програмування, включаючи C++, Python, Java та інші.

OpenCV містить тисячі функцій, які дозволяють працювати з візуальними даними як на низькому, так і на високому рівні. Наприклад, вона надає інструменти для обробки пікселів, виконання чітких математичних операцій над зображеннями, побудови алгоритмів машинного навчання та навіть роботи з тривимірними сценами. Бібліотека підтримує численні формати відео- і графічних файлів, а також може інтегруватися з апаратним забезпеченням, що дозволяє обробляти відеопотік у реальному часі. Її архітектура орієнтована на ефективність, тому багато функцій реалізовані у вигляді оптимізованого C++ коду з підтримкою SIMD-інструкцій і апаратного прискорення.

У сучасних задачах комп'ютерного зору OpenCV використовується як база для розробки систем відеоспостереження, автономного водіння, медичної діагностики, аналізу руху, біометричної ідентифікації та багато чого іншого. Бібліотека також активно застосовується як інфраструктурна складова в навчанні моделей штучного інтелекту та робототехніці.

У межах даного проєкту OpenCV виступає як ключова бібліотека низькорівневого доступу до відеопотоку та обробки зображень, а також як інструмент для побудови цілісного графа обробки, в який інтегруються високорівневі засоби, такі як MediaPipe. Разом із супутніми бібліотеками, зокрема NumPy для обчислень та json для серіалізації даних, OpenCV створює гнучке

середовище, яке дозволяє ефективно працювати з відеоінформацією та забезпечує надійну інтеграцію з іншими модулями системи.

## 2.2 Редактор вихідного коду Visual Studio Code

Visual Studio Code (VSC) — це сучасний багатофункціональний редактор вихідного коду, розроблений компанією Microsoft і вперше представлений публіці у 2015 році. На відміну від класичних інтегрованих середовищ розробки (IDE), таких як Eclipse чи PyCharm, VSC був створений як легкий, кросплатформенний редактор із підтримкою розширень, що дозволяють поступово додавати потрібну функціональність. Завдяки цьому підходу редактор швидко здобув популярність серед розробників програмного забезпечення, вебпрограмістів, інженерів машинного навчання та фахівців з автоматизації.

Visual Studio Code є вільно доступним і відкритим програмним забезпеченням, що підтримується та розвивається як компанією Microsoft, так і світовою спільнотою. Він написаний з використанням таких технологій, як Electron, TypeScript і Node.js, що робить його зручним і гнучким навіть для розширення силами сторонніх розробників.

З моменту свого запуску VSC отримав потужну підтримку основних мов програмування: Python, C++, JavaScript, Go, Java, Rust, PHP та багатьох інших. Однією з ключових особливостей редактора є механізм розширень (Extensions Marketplace), який дозволяє додавати підтримку практично будь-якої технології — від мови до фреймворку чи утиліти розгортання. Саме завдяки цьому Visual Studio Code легко адаптується під будь-який тип проєкту — від невеликого скрипту до складного багатомодульного рішення.

Інтеграція з системами керування версіями, зокрема Git, реалізована на глибокому рівні: користувач може не лише переглядати зміни, а й виконувати коміти, створювати гілки, відновлювати історію прямо в межах редактора. Також підтримується вбудований термінал, що дозволяє працювати з командним рядком,

запускати сценарії, створювати віртуальні середовища тощо, не виходячи з редактора.

У Visual Studio Code реалізована низка інтелектуальних функцій: автодоповнення коду, швидкий пошук по проєкту, миттєвий перехід до визначення функції або змінної, перевірка синтаксису та інтеграція зі статичним аналізом коду. Для Python-розробки існує офіційне розширення, яке підтримує середовища Conda, інтеграцію з Jupyter Notebook, запуск тестів, форматування та налагодження коду.

VSC активно використовується як у навчальних цілях, так і в комерційних проєктах. Завдяки низьким системним вимогам, відкритості та високій кастомізації він став універсальним рішенням для розробників усіх рівнів.

У межах даного дипломного проєкту Visual Studio Code використовувався як основне середовище розробки. У ньому створювались модулі на мові Python для захоплення міміки користувача за допомогою бібліотеки MediaPipe, обробки координат обличчя, формування структурованих даних у форматі JSON, а також для налагодження зв’язку з Blender через Python API. Редактор забезпечував ефективну організацію коду, швидкий запуск скриптів, візуальний контроль за структурою проєкту та підтримку Git.

Таким чином, Visual Studio Code став не лише інструментом написання коду, але й повноцінним середовищем розробки, що забезпечило високу гнучкість, мобільність і зручність реалізації системи передачі міміки.

## 2.3 Використання Git для контролю версій

Git — це розподілена система контролю версій, яка стала стандартом у сфері розробки програмного забезпечення. Вона була створена Лінусом Торвальдсом у 2005 році як інструмент для керування кодовою базою ядра Linux. З того часу Git перетворився на універсальний інструмент, який використовується як в індивідуальних проєктах, так і в масштабних командних розробках.

Основне призначення Git — фіксація змін у програмному коді з можливістю

повернення до будь-якого попереднього стану. Це досягається за допомогою створення комітів — знімків поточного стану проєкту, які зберігаються з коментарем. Такий підхід дозволяє розробникам точно відстежувати історію змін, аналізувати внесені правки та ефективно керувати процесом розробки.

Однією з найважливіших функцій Git є можливість створення гілок (branches). Це дозволяє паралельно розробляти нові функціональні модулі, виправляти помилки або експериментувати з кодом без впливу на основну стабільну версію проєкту. Кожна гілка є ізольованим середовищем, яке згодом можна об’єднати з основною версією після проходження тестування. Цей механізм особливо корисний у командній роботі, де кілька учасників одночасно працюють над різними частинами одного проєкту.

Git також дозволяє ефективно вирішувати конфлікти при об’єднанні змін, що особливо актуально, коли кілька розробників вносять правки в один і той самий файл. Завдяки цьому забезпечується цілісність проєкту та контрольованість усіх змін, навіть у складних сценаріях розробки.

Ще однією важливою перевагою Git є його підтримка віддалених репозиторіїв. Вони дозволяють зберігати резервні копії проєкту на хостингах на зразок GitHub, GitLab або Bitbucket. Це гарантує захист від втрати даних і спрощує доступ до коду з різних пристроїв і місць. Розробники можуть синхронізувати свої локальні копії з віддаленими, що робить командну роботу максимально ефективною.

У рамках цього проєкту система контролю версій Git використовувалася для зберігання та керування всіма версіями коду, включаючи модулі захоплення міміки, обробки даних і скрипти для Blender. Git дозволив зберігати кожен етап розробки, експериментувати з новими підходами без ризику втрати основного функціоналу та швидко повертатися до стабільних версій у разі потреби.

Таким чином, Git виконує не лише роль системи збереження змін, а й виступає як інструмент організації розробки, що забезпечує стабільність, надійність і керованість усього життєвого циклу програмного продукту.

## 2.4 Середовище Blender та робота з параметричними моделями

Blender — це потужне та багатофункціональне середовище для створення тривимірної графіки, яке включає в себе всі основні інструменти для моделювання, анімації, візуалізації, композитингу, скульптингу, рендерингу та відеомонтажу. Вперше Blender був розроблений у 1994 році нідерландською компанією NeoGeo як внутрішній інструмент для створення анімацій. У 1998 році він був випущений як окрема програма компанією Not a Number Technologies (NaN). Після банкрутства компанії у 2002 році його вихідний код було викуплено завдяки краудфандинговій кампанії і відкрито під ліцензією GNU GPL. Відтоді він підтримується фондом Blender Foundation та активно розвивається спільнотою ентузіастів, художників, інженерів та науковців.

Blender має модульну структуру і містить численні компоненти для різних аспектів 3D-виробництва. Серед них: редактор мешів для моделювання, інструменти скульптингу, візуальні ноди для створення шейдерів, вбудований рушій візуалізації Cycles з підтримкою трасування променів, а також реальний рендерер Eevee, який дозволяє працювати з освітленням і матеріалами в режимі реального часу.

Однією з ключових можливостей Blender є підтримка анімації на основі параметричних моделей. Це означає, що поведінку та форму 3D-об’єктів можна змінювати за допомогою набору числових параметрів, що описують деформації, положення або вирази. До інструментів, які реалізують параметризацію, належать:

- Shape keys — технологія морфінгу, яка дозволяє створювати кілька альтернативних форм одного об’єкта (наприклад, усмішка, нахмурене обличчя, моргання). Зміна значення shape key плавно трансформує об'єкт між початковим та зміненим станом;
- Armature та rigging — система кісток і суглобів, яка створює «скелет» моделі. За допомогою скелетної анімації (bone-based animation) можна змінювати

положення частин моделі, зберігаючи при цьому її цілісність. Ріггінг дозволяє прив’язати геометрію до кісток, а також автоматизувати складні рухи та вирази через контрольні об’єкти.

Armature та rigging — система кісток і суглобів, яка створює «скелет» моделі. За допомогою скелетної анімації (bone-based animation) можна змінювати положення частин моделі, зберігаючи при цьому її цілісність. Ріггінг дозволяє прив’язати геометрію до кісток, а також автоматизувати складні рухи та вирази через контрольні об’єкти.

Ще одним важливим аспектом Blender є підтримка скриптування на мові Python. У програмі реалізовано повноцінний API (Blender Python API), що дає можливість керувати практично кожною функцією Blender — створенням об’єктів, зміною матеріалів, запуском рендеру, модифікацією анімації, обробкою shape keys, трансформацією кісток тощо. Це робить Blender не просто графічним редактором, а повноцінним середовищем для програмованої генерації та управління 3D-сценами. Скрипти можуть зберігатись як окремі Python-файли або виконуватись безпосередньо у вбудованому редакторі коду Blender.

Blender підтримує імпорт та експорт великої кількості 3D-форматів, зокрема OBJ, FBX, STL, Collada, glTF. Він також дозволяє створювати симуляції рідини, диму, тканин, м’язів, жорстких тіл і частинок, що робить його універсальним інструментом для фізично коректної візуалізації.

З роками Blender набув широкого застосування в ігровій індустрії, кіновиробництві, науковій візуалізації, архітектурному проєктуванні, AR/VR, віртуальних презентаціях і навіть у біомедичних дослідженнях. Його популярність зростає завдяки комбінації відкритого коду, потужного функціоналу та постійної підтримки нових технологій.

Blender активно використовується не лише окремими митцями, а й студіями — прикладом є відкриті фільми «Elephants Dream», «Big Buck Bunny», «Sintel» і «Spring», повністю створені у Blender та викладені у вільному доступі.

Таким чином, Blender — це не лише програма для моделювання, а повноцінна

екосистема для створення, анімації та управління тривимірним контентом, з широкими можливостями автоматизації та програмної інтеграції.

## 2.5 Формат обміну даними (JSON)

JSON (JavaScript Object Notation) — це текстовий формат обміну даними, створений для зручного зберігання та передачі структурованої інформації між програмними модулями. Формат був розроблений Дугласом Крокфордом у 2001 році й з часом став де-факто стандартом для обміну даними в сучасних інформаційних системах. Його популярність зумовлена простотою, зрозумілістю для людини, легкістю парсингу, а також широкою підтримкою практично всіма мовами програмування, включаючи Python, JavaScript, Java, C++, Go та інші.

Формат JSON базується на структурі, подібній до об’єктів у JavaScript, де дані представлені у вигляді пар "ключ — значення". JSON підтримує вкладені об’єкти, масиви, числові, логічні та текстові значення, що дозволяє описувати складні структури даних, включно з багаторівневими конфігураціями, параметрами та результатами обчислень. Його синтаксис є легким для прочитання і водночас чітко формалізованим, що забезпечує надійність при автоматизованій обробці.

Завдяки своїй універсальності JSON широко використовується в API-запитах, конфігураційних файлах, обміні даними між вебклієнтом і сервером, а також у логах, базах даних NoSQL (наприклад, MongoDB) та машинному навчанні для збереження моделей і параметрів. У багатьох випадках JSON витіснив XML завдяки меншій надмірності та вищій швидкості обробки.

Однією з важливих переваг JSON є його простота інтеграції з мовою Python, яка надає вбудований модуль json для серіалізації (перетворення структури даних у текстовий формат) і десеріалізації (зчитування тексту та перетворення його у внутрішню структуру даних). Це дозволяє зручно працювати з параметрами, що зберігаються у вигляді словників, списків або вкладених об’єктів.

Формат JSON також є зручним для взаємодії між різними середовищами —

наприклад, між Python і JavaScript, або між трекінговою системою й системою візуалізації. Завдяки цій універсальності JSON став одним із найнадійніших і найбільш зручних рішень для передачі даних у багатокомпонентних програмних системах.

На відміну від бінарних форматів обміну, JSON залишається повністю читабельним для людини, що спрощує діагностику, ручне редагування та тестування під час розробки. Це робить його не лише ефективним, але й практичним форматом для повсякденного використання в реальних проєктах.

Таким чином, JSON є важливим компонентом сучасної програмної інженерії, що забезпечує ефективну, стандартизовану та кросплатформенну передачу даних між модулями, процесами та сервісами.

## Висновки до розділу 2

У другому розділі було проведено детальний аналіз інструментів та середовищ, що використовуються для розробки системи передачі міміки користувача на параметричну модель обличчя. Також було описано вибрану мову програмування, Python — гнучку, високорівневу мову з відкритим кодом, яка надає широкий спектр бібліотек та інструментів, необхідних для реалізації обробки відеопотоку, комп’ютерного зору та керування 3D-графікою. Підтримка різних парадигм, простота синтаксису, доступність модулів та активна спільнота розробників зробили Python оптимальним вибором для реалізації всієї логіки проєкту в межах єдиного середовища.

Серед бібліотек ключову роль відіграє MediaPipe, що надає попередньо навчені модулі для зчитування міміки з обличчя, зокрема модуль Face Mesh. У поєднанні з OpenCV, яка відповідає за доступ до відеопотоку, попередню обробку зображень та вивід результатів, ці інструменти забезпечують стабільну та точну роботу в режимі реального часу. Допоміжні бібліотеки, такі як NumPy та json, дозволяють виконувати необхідні числові обчислення та обмін даними у

структурованому вигляді.

Як середовище розробки було обрано Visual Studio Code — сучасний багатофункціональний редактор коду, що забезпечує зручність, продуктивність та високу адаптивність завдяки системі розширень. Його підтримка інтеграції з Git, наявність вбудованого терміналу, засобів налагодження та автодоповнення коду створюють комфортне середовище для розробки та супроводу Python-проєктів будь-якої складності.

Для візуалізації міміки було обрано професійне середовище Blender, що дозволяє створювати та анімувати параметричні 3D-моделі з використанням shape keys та кісткової анімації. Його відкритість, розширення через Python API та багаторічна підтримка спільноти зробили це середовище провідним рішенням у сфері 3D-графіки.

Система Git була використана як засіб контролю версій, що дозволило зберігати історію змін, організовувати роботу над окремими функціональними модулями, уникати конфліктів при об’єднанні коду та забезпечити стабільність у процесі розробки. Завдяки підтримці гілок, комітів та інтеграції з віддаленими репозиторіями Git гарантує відмовостійкість та захист даних у ході еволюції проєкту.

Окремо розглянуто формат JSON, який виступає універсальним засобом обміну даними між модулями. Його читабельність, простота та повна підтримка у Python дозволили ефективно реалізувати зв’язок між частинами системи, що працюють автономно — наприклад, між модулем трекінгу міміки та Blender.

Загалом, обрані інструменти та технології повністю відповідають вимогам поставленого завдання. Вони дозволяють забезпечити гнучкість, масштабованість і функціональність системи, створюючи надійне середовище для реалізації обробки, передачі й візуалізації міміки користувача в реальному часі.

# 3 ОПИС ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЇ СИСТЕМИ ПЕРЕДАЧІ МІМІКИ

## 3.1. Архітектура проекту та опис його компонентів

Розроблена архітектура системи передачі міміки від користувача до віртуального персонажа будується на принципі поділу відповідальностей між незалежними модулями, які взаємодіють між собою через обмін даними у структурованому вигляді. Такий спосіб дозволяє забезпечити гнучкість, масштабованість і адаптивність системи. Головною перевагою вибраного способа є спрощення розгортання системі, її підтримку і можливість використовувати її з різними 3D-моделями.

Сама система складається з двох головних компонентів: модуля зчитування міміки користувача та модуля анімації віртуального персонажа у Blender. Взаємодія між ними відбувається через проміжний файл JSON, який оновлюється у реальному часі.

Модуль зчитування міміки реалізован на мові програмування Python. В його основі лежить бібліотека MediaPipe, яка дозволяє в реальному часі визначати координати 468 ключових точок обличчя з відеопотоку. Для доступу до камери, зчитування кадрів, їх конвертації та виводу використовується OpenCV, що також дозволяє проводити попередню обробку зображення.

Головним етапом у цьому модулі є обчислення параметрів міміки: відкриття рота, положення бров, повік і так далі. Ці параметри не використовуються у сирому вигляді, а проходять нормалізацію відносно розміру обличчя користувача, а також згладжуються з використанням методів фільтрації шуму. Отримані значення формуються у словник і зберігаються у вигляді JSON-файлу.

Другий компонент, що відповідає за відображення міміки, реалізовано у вигляді Python-скрипту всередині середовища Blender. Цей скрипт читає JSON файл, аналізує передані значення та застосовує їх до shape keys чи до анімаційних

кісток відповідної 3D-моделі. Для цього скрипта використовується Blender API, який надає повний контроль над параметрами трансформації об’єктів, положенням кісток, деформаціями геометрії та поведінкою моделі.

Таке розділення модулів дозволяє забезпечити чіткий поділ між технічною частиною, що зчитує міміку, і візуальною частиною, яка відповідає за графічне відтворення. Це також відкриває можливість у майбутньому замінити 3D-сцену, модель або навіть середовище візуалізації, не змінюючи основної логіки зчитування міміки. Наприклад, замість Blender можна було б використати рушій Unity або Unreal Engine, якщо це передбачено в іншій версії реалізації.

Ще однією важливою особливістю архітектури є повна незалежність від серверної інфраструктури. Система функціонує локально, не зберігає дані користувача і не потребує зовнішніх API, що підвищує її приватність, безпеку та стабільність. Такий підхід дозволяє запускати систему на будь-якому комп’ютері з встановленим Python і Blender без складної конфігурації чи налаштування серверів.

На рисунку 3.1 наведено архітектурну схему проєкту, яка демонструє послідовність обробки: зчитування відеопотоку, обробка міміки, формування даних у json та їх застосування до параметричної моделі в Blender.

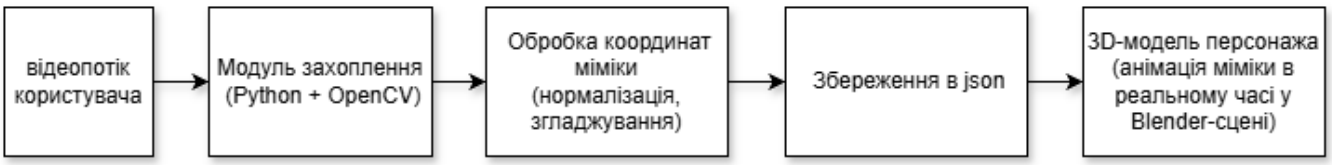



Рис. 3.1. Архітектура системи передачі міміки користувача на 3D-модель

Таким чином, побудована архітектура поєднує в собі модульність, простоту, ефективність і гнучкість, що дозволяє адаптувати систему під конкретні потреби або розширювати її в майбутньому.

## 3.2 Реалізація модуля зчитування міміки

### 3.2.1 Зчитування відеопотоку та визначення ключових точок

Першим етапом у процесі передачі міміки є захоплення відео з камери та виявлення ключових точок обличчя користувача. Для цього в модулі використовується бібліотека OpenCV у поєднанні з MediaPipe Face Mesh — попередньо натренованою моделлю, що забезпечує високу точність в режимі реального часу.

Відеопотік зчитується за допомогою функції cv2.VideoCapture(0), яка активує камеру комп'ютера та дозволяє опрацьовувати кадри по черзі в циклі. Кожен кадр попередньо конвертується з кольорової моделі BGR (типової для OpenCV) у RGB, яка необхідна для обробки у MediaPipe.

Після конвертації кадр передається до face_mesh.process(...), де виконується детектування обличчя та побудова його топології. Використовується налаштування refine_landmarks=True, яке активує покращене позиціонування ключових точок (зокрема — навколо очей та губ) і дозволяє точніше відстежувати міміку.

MediaPipe Face Mesh визначає багато тривимірних орієнтирів (landmarks), які розташовані відповідно до анатомії обличчя. Кожна точка має координати x, y, z, нормалізовані в межах [0, 1] або [-1, 0] відносно ширини та висоти кадру. Це дозволяє системі працювати незалежно від роздільної здатності камери або розміру обличчя.

Серед ключових орієнтирів, які використовуються для подальшого аналізу, виділяються такі:

- 234 і 454 — лівий та правий край обличчя;
- 10 і 152 — верх і низ голови (для визначення висоти);
- 33, 263 — зовнішні кути очей;
- 468, 473 — зіниці (iris) лівого та правого ока;
- 11, 16, 61, 291 — верхня/нижня частина губ та куточки рота;

- 52, 55, 282, 285 — точки бровей.

Додатково реалізована візуалізація важливих точок безпосередньо на зображенні, що транслюється у вікні, з метою візуального контролю правильності трекінгу.

Таким чином, цей етап забезпечує базу для подальших розрахунків — усі мімічні параметри, такі як відкриття рота, положення очей, рух брів чи зміщення губ, формуються саме на основі координат цих ключових точок обличчя. Це дозволяє надалі будувати універсальні та масштабовані алгоритми для передачі міміки.

### 3.2.2 Калібрування нейтрального стану обличчя

Калібрування обличчя — це важливий підготовчий етап, він забезпечує коректну роботу усієї системи. Метою цього етапу є фіксування індивідуальних особливостей обличчя в стані спокою. Отриманні під час калібрування параметри слугують еталоном, відносно якого в подальшому будуть обчислюватися всі змінні.

Процес калібрування виконується в інтерактивному режимі, користувачу потрібно натискати певні клавіші під час роботи модуля захвату міміки, для того щоб зафіксувати поточні значення мімічних параметрів.

Ці нейтральні значення зберігаються у спеціальному файлі калібрування і використовуються для нормалізації вхідних даних у режимі реального часу. Це дозволяє системі не залежати від конкретної форми або пропорцій обличчя, а реагувати лише на зміни. Без цього калібрування у різних користувачів були б проблеми з різними початковими положеннями і це призводило би до некоректного розпізнавання міміки.

### 3.2.3 Обчислення мімічних параметрів

У процесі розрахунку мімічних параметрів із координат ключових точок формуються числові значення, які відповідають виразам обличчя користувача. Це значення відкриття рота, стиснення губ, напрямок голови та очей, а також

положення брів. Формули мають геометричну основу і використовують нормалізовані відстані між анатомічно важливими орієнтирами.

Розрахунок відкриття рота є одним з етапів у створенні реалістичної анімації персонажа, оскільки цей параметр значно впливає на загальний вираз обличчя. Для визначення ступеня відкриття рота використовуються дві точки обличчя — верхня (точка 13) і нижня (точка 14) губи. Ці точки вибрані через їх стабільне положення при рухах щелепи і губ, що дозволяє точно оцінювати зміни відстані між ними.

Спочатку вимірюється відстань між цими точками по вертикалі, яка потім нормалізується відносно висоти обличчя. Це необхідно для того, щоб отримане значення було пропорційним до розміру обличчя, а не залежало від масштабу моделі. Після цього значення відкриття рота порівнюється з попередньо збереженим нейтральним положенням, яке було встановлено під час калібрування. Це дозволяє врахувати природне положення рота, коли він закритий, і відобразити навіть невеликі рухи губ. Формула (3.1) за якою обчислюється ступінь відкритості рота:

$$mouth_{open} = \frac{y_{13} - y_{14}}{h} \quad (3.1)$$

Де $y_{13}$, $y_{14}$— вертикальні координати верхньої та нижньої губ відповідно;

$h$ — висота обличчя (від чола до підборіддя).

Ширина рота (3.2) оцінюється через відстань між лівим та правим куточками губ:

$$mouth_{width} = \frac{x_{291} - x_{61}}{w} \quad (3.2)$$

Де $x_{291}, x_{61}$ — горизонтальні координати правого та лівого куточків рота;

$w$— ширина обличчя.

Рух губ є складним аспектом міміки, оскільки включає в себе як

горизонтальні, так і вертикальні рухи, а також стиснення і розтягування губ. Для точного моделювання цих змін необхідно враховувати як форму, так і розміри обличчя, а також попередньо збережені нейтральні параметри губ.

Першим кроком є визначення горизонтального стиснення губ (pucker). Цей параметр обчислюється на основі відстані між лівим (точка 291) і правим (точка 61) кутами губ. Ця відстань нормалізується відносно загальної ширини обличчя, щоб врахувати різницю в розмірах різних моделей. Для точнішого визначення стиснення використовуються значення, збережені під час калібрування, що дозволяє відстежувати навіть незначні рухи губ. Формула (3.3) за якою обчислюється горизонтальне стиснення губ

$$pucker_{horiz} = \frac{d_x^{neutral} - \frac{x_{291} - x_{61}}{w}}{\Delta_x} \quad (3.3)$$

Де $d_x^{neutral}$ — відстань між губами у нейтральному стані;

$\Delta_x$ — підібране значення нормалізації.

Після цього розраховується вертикальне стиснення губ, яке визначається як відстань між верхньою (точка 11) і нижньою (точка 16) губами. Це значення нормалізується відносно висоти обличчя, щоб зробити параметр більш стабільним при зміні дистанції до камери. Вертикальний стиск розраховується подібно (3.4):

$$pucker_{vert} = \frac{\left(\frac{(y_{11} - y_{16})}{h}\right) - d_y^{neutral}}{\Delta_y} \quad (3.4)$$

Де $y_{11}, y_{16}$ — вертикальні координати верхнього і нижнього центру губ;

$d_y^{neutral}$ — вертикальна відстань у нейтральному стані.

Для покращення точності симуляції руху губ використовується комбінація горизонтального і вертикального стиснення, що дозволяє моделювати більш складні вирази обличчя, такі як поцілунки або свист. Однак, ці значення додатково коригуються залежно від ступеня відкриття рота, щоб уникнути некоректних результатів, коли рот широко відкритий. Для цього використовуються згладжені

значення відкриття рота, що допомагає уникнути різких стрибків в анімації.

Посмішка визначається на основі зміщення кутів рота вгору відносно нижньої губи з урахуванням компенсації відкриття рота (3.5):

$$smile = \max(0, \left(\frac{y_{16} - y_{61}}{h}\right) - comp \quad (3.5)$$

Де $y_{61}, y_{16}$ — вертикальні координати кута рота та нижньої губи;

$comp$ — компенсаційний коефіцієнт залежно від відкриття рота.

Положення голови відображається трьома напрямками: нахил, поворот і підйом/опускання. Всі вони визначаються за відносними відстанями між симетричними точками обличчя (3.6), (3.7), (3.8):

$$head_{tilt} = y_{234} - y_{454} \quad (3.6)$$

$$head_{turn} = x_{234} - x_{454} \quad (3.7)$$

$$head_{updown} = y_{10} - y_{152} \quad (3.8)$$

Де $x, y$ — координати точок: щоки (234, 454), чола (10) та підборіддя (152).

Рух очей є ще більш складним аспектом міміки, оскільки включає в себе як контроль положення зіниць, так і відкриття або закриття повік. Цей процес вимагає високої точності для досягнення реалістичних результатів, оскільки навіть незначне відхилення може значно вплинути на вираз обличчя персонажа.

Першим етапом є визначення горизонтального і вертикального положення зіниць. Для цього використовується набір точок, що окреслюють внутрішні і зовнішні кути очей (наприклад, точки 133 і 33 для лівого ока, а також 362 і 263 для правого). Положення зіниць обчислюється як різниця між відстанями від зіниці до цих контрольних точок. Ці значення потім нормалізуються відносно розміру очної западини, що дозволяє уникнути впливу розміру обличчя або дистанції до камери. Для визначення положення очей використовуються відносні відстані зіниці до внутрішніх і зовнішніх меж ока і обчислюється за формулами (3.9),(3.10):

$$eye_x = \frac{d_{inner} - d_{outer}}{d_{inner} + d_{outer}} \quad (3.9)$$

$$eye_y = \frac{d_{top} - d_{bottom}}{d_{top} + d_{bottom}} \quad (3.10)$$

Де $d_{inner}, d_{outer}$ — відстані зіниці до внутрішнього/зовнішнього краю;

$d_{top}, d_{bottom}$ — відстані зіниці до верхньої/нижньої повіки.

Рух брів ще один з елементом міміки. Для визначення положення брів використовуються точки, що позначають внутрішні та зовнішні кінці кожної брови. Наприклад, точки 52 і 55 для лівої брови, а також 282 і 285 для правої. Ці точки дозволяють визначити як вертикальне положення брів, так і їх нахил.

Положення брів розраховується відносно нейтрального стану, що був зафіксований під час калібрування. Для цього враховуються як внутрішні, так і зовнішні кінці брів, щоб точніше передати такі вирази, як подив, зосередженість або гнів. Висота кожної брови вимірюється як різниця між поточним вертикальним положенням точки брови і її нейтральним значенням.

Оскільки рухи брів можуть залежати від положення очей, застосовується компенсація, яка враховує зміщення зіниць по вертикалі. Це допомагає уникнути спотворень, коли персонаж дивиться вгору або вниз. Наприклад, під час погляду вверх брови можуть здаватися опущеними, якщо не враховувати цю корекцію. Приклад формули (3.11) за якою розраховується положення зовнішніх точок брів:

$$brow_{offset} = \frac{y_{brow} - y_{eye} - bias - y_{neutral}}{range} \quad (3.11)$$

Де $y_{brow}, y_{eye}$ — координати брови та контрольної точки ока;

$bias$ — компенсатор вертикального руху;

$y_{neutral}$ — нейтральне положення брови;

$range$ — нормалізатор чутливост.

Такі математичні моделі дозволяють узагальнено й масштабно незалежно описати мімічний стан обличчя та підготувати параметри до згладжування й анімації.

### 3.2.4 Нормалізація відносно геометрії обличчя

Нормалізація мімічних параметрів виконується з метою уніфікації значень незалежно від відстані до камери, розміру кадру або анатомічних особливостей користувача. Оскільки всі координати, які повертає MediaPipe Face Mesh, задані у відносному (нормалізованому) просторі, виникає потреба перевести їх у локальні метричні відношення для порівняння та передачі на 3D-модель.

Для цього кожен мімічний параметр масштабується відносно ширини обличчя (face_width) або його висоти (face_height), які обчислюються на основі ключових точок — зокрема 234 і 454 для горизонталі, 10 і 152 для вертикалі. Таким чином, усі відстані між частинами обличчя переводяться у відносні величини, незалежні від масштабу.

Крім геометричної нормалізації, використовується локальна адаптивна — тобто віднесення поточних значень до заздалегідь зафіксованих «нейтральних» значень, що зберігаються у файлі eye_calibration.json. Це дозволяє зробити порівняння не з абсолютним нулем, а з природним станом обличчя конкретного користувача. До таких нейтральних величин належать значення відкриття рота, відстані між губами, відкритість очей, положення брів тощо.

Усі ці параметри нормалізуються за універсальною формулою такого вигляду:

$$v_{norm} = \frac{v_{current} - v_{neutral}}{v_{range}} \quad (3.12)$$

- $v_{norm}$ — нормалізоване значення параметра;
- $v_{corrent}$ — поточне значення;
- $v_{neutral}$ — нейтральне значення;
- $v_{range}$ — очікуваний діапазон змін (наприклад, максимальне відкриття рота, відстань між губами тощо).

Нормалізація за такою формулою забезпечує отримання значень у фіксованому діапазоні (зазвичай [0; 1] або [-1; 1]), що надалі дозволяє передавати їх на 3D-модель незалежно від конкретних розмірів обличчя чи положення

користувача в кадрі. Завдяки цьому система зчитування міміки стає стійкою до варіацій масштабу, положення камери та індивідуальних особливостей.

### 3.2.5 Згладжування та фільтрація шуму

Згладжування та фільтрація шуму виконується для того щоб досягти стабільної і плавної анімації, оскільки навіть незначне тремтіння або випадкові сплески вхідних даних можуть призвести до візуальних артефактів на 3D-моделі. Це особливо важливо при роботі з очами.

Для вирішення проблеми з шумом у модулі зчитування міміки реалізовано згладжування всіх ключових параметрів міміки.

Саме згладжування виконується за допомогою експоненційного згортання значень за формулою:

$$v_{smooth} = (1 - a) * v_{prev} + a * v_{current} \quad (3.13)$$

- $v_{smooth}$ — нове згладжене значення;
- $v_{prev}$ — попереднє згладжене значення;
- $v_{current}$ — поточне (сире) значення параметра;
- $a$ — коефіцієнт згладжування.

Завдяки цьому нові значення не замінюють повністю попередні, а лише частково оновлюють їх. Такий підхід дозволяє усунути коливання.

Додатково, у випадку, коли очі закриті, або частково закриті, система автоматично зменшує вплив координат зіниць, для того щоб уникнути скачків при морганні.

### 3.2.6 Формування та збереження mimic.json

Останнім етапом кожного кадру обробки формується структура даних, яка містить усі обчислені параметри міміки. Для цього у коді створюється словник mimic_data, який поступово додаються значення, отримані після проходу описаної

логіки.

Для збереження файлу використовується функція tempfile.NamedTemporaryFile(), що дозволяє для початку створити тимчасовий файл та записати в нього структуру mimic_data і тільки потім безпечно перемістити його на місце основного файлу. Так зроблено для запобігання конфлікту читання-запису, який може виникнути якщо файл буде зчитуватись Blender у момент оновлення.

Завдяки цьому підходу, JSON постійно оновлюється для подальшого використання іншим модулем для зчитування і анімації. Таким чином, цей файл є основним каналом передачі, даних між двома с частинами системи.

## 3.3 Інтеграція з параметричною моделлю обличчя в Blender

Переходячи до другого компонента системи, саме модуля анімації в Blender. Можна розповісти що він реалізується через зчитування параметрів міміки та їх передачі на параметричну 3D-модель. Весь процес побудован навколо буферного файлу mimic.json, який регулярно оновлюється нашим першим компонентом трекінгу. Для того щоб забезпечити плавну і безперервну синхронізаціію, скрипт оновлює міміку з частотою 30 кадрів на секунду — саме таку швидкість формування кадрів забезпечує OpenCV у першому модулі.

На кожному кроці перевіряється, чи файл mimic.json оновлено і чи його запис завершено. Якщо файл занадто свіжий або частково збережений, кадр пропускається. Завдяки такому підходу можна мінімізувати ризик зчитування неповних даних.

Після першого успішного зчитування визначаються нейтральні значення, тобто початкові координати очей та інших параметрів міміки. І усі подальші зміни відбуваються відносно цих еталонних позицій. Так зроблено для відслідковування саме деформацій, а не абсолютного положення. Як приклад можна привести рух зіниці, коли вона зміщується вліво, ця зміна відбувається відносно зафіксованого

нейтрального положення, навіть якщо рухається уся голова.

Усі значення, навіть після згладжування у першому модулі, можуть стрибати. Щоб уникнути цього, у скрипті реалізовано ще одне згладжування, яке застосовується перед тим, як значення передасться до shape keys або кісток.

Це додаткове згладжування виконує роль компенсації витрат чи затримок у потоці даних і підвищує плавність анімації.

## 3.4 Застосування shape keys і кісткової структури

Обрахунок методом прямого перебору є точним методом обчислення параметру функціональної зв’язності [7]. Цей метод визначається наступним чином.

Для подій зв'язності $E_{x,y}$ та незв'язності $\bar{E}_{x,y}$ в РІС, можна бачити, що першому відповідає подія існування хоча б одного простого ланцюга (ПЛ), а другому – хоча б однієї простої січної множини (ПСМ). При цьому елементи графа, які не входять ні до ПЛ, ні до ПСМ, можуть бути в будь-якому з трьох станів: справному, несправному та байдужому [7]. Це положення записується наступними формулами, представленими на рисунку 3.3.

Після підготовки нейтральних координат та перевірки актуальності даних, скрипт переходить саме до анімації, використовуючи кістки і shape keys. Вибраний підхід обгрунтовується розділенню деяких аспектів міміки на ті що потребують більше візуалізації і на ті що потребують менше, а також для демонстрації того що можна використати для анимацій.

Для прикладу використання кісток використаємо відкриття рота. Само відкриття рота реалізовано через обертання кістки щелепи. Аналогічно керування кутами рота, для того щоб змінювати вираз при відкритому роті, тобто скласти губи у виразі «о» чи «а», наприклад для демонстрації крику чи для розтягування кутків при широкому відкритті рота. Така локальна анімація виконується відносно нейтральної позиції кожної кістки, що зберігається при першому кадрі.

Рух очей також реалізовано через маніпуляцію кісток. Визначення погляду базується на координатах зіниць, які надходять окремо для кожного ока. Також додатково система враховує стан моргання і якщо одно око повністю заплющене то для уникнення хаотичних стрибків, починає використовуватись інше око як джерело координат.

Що до shape keys, як і було сказано, вони використовуються для більш емоційних виразів, наприклад, при піднятті брів з'являються зморшки, чи при нахмурені брови трішки висуваються уперед, при посмішці виявляються ямочки. Всі ці параметри керуються відповідними ключами, а ключі активуються із використанням попередньо збережених нейтральних значень.

Нормалізація значень у даному етапі відіграє ключову роль, тому що вона обмежує максимально та мінімально допустимий зсув і запобігає спотворень.

Після обробки всіх параметрів, скрипт оновлює сцену у Blender через bpy.context.view_layer.update(). Завдяки цьому забезпечується актуалізація моделі в режимі реального часу та відображення міміки користувача синхронно з живим відеопотоком.

## Висновки до розділу 3

У третьому розділі було детально розглянуто процес проєктування і реалізації системи передачі міміки користувача на 3D-персонажа в Blender. Розроблена архітектура охоплює два самостійні, але тісно взаємопов'язані програмні модулі: Python-компонент для аналізу міміки в реальному часі та Blender-скрипт для анімації параметричної моделі обличчя.

Перший компонент реалізовано на основі бібліотек MediaPipe та OpenCV. Його завданням є захоплення відеопотоку з камери, виявлення ключових точок обличчя (face landmarks), обчислення параметрів міміки та формування нормалізованого набору значень у форматі JSON. Всі дані записуються у файл mimic.json, який постійно оновлюється з частотою приблизно 30 кадрів на секунду,

що відповідає частоті оновлення зображення при обробці через OpenCV. Так забезпечується відповідність між швидкістю аналізу міміки та темпом оновлення анімації у Blender.

Особливу увагу було приділено процедурі калібрування нейтрального стану обличчя. Саме вона дозволяє фіксувати базові параметри, які надалі використовуються для побудови відносних змін. Це забезпечує правильну інтерпретацію індивідуальної міміки користувача, незалежно від положення голови чи особливостей анатомії. Всі значення, отримані в процесі калібрування, зберігаються у конфігураційному файлі і зчитуються автоматично.

Після збору й обробки даних другий модуль — скрипт, що працює у Blender виконує завдання зчитування файлу mimic.json, інтерпретації його значень та передачі їх у форму анімації. Тут реалізовано логіку синхронізації, де кожне оновлення параметрів міміки проходить перевірку на актуальність. Якщо файл був щойно створений або запис не завершено, кадр пропускається. Це дозволяє уникнути зчитування пошкоджених або неповних даних і гарантує стабільну роботу у режимі реального часу.

Параметри міміки поділяються за способом реалізації: грубі рухи, як-от обертання щелепи, рух куточків рота та очей, реалізуються через кісткову анімацію. Водночас дрібні вирази — моргання, примружування, підняття/опущення брів, посмішка, нахмурення, стискання губ тощо — керуються через shape keys, що дозволяє досягти вищої деталізації. Всі shape keys активуються відносно фіксованих нейтральних значень, які встановлюються при першому зчитуванні даних.

Слід зазначити, що розробка включає як автоматичне визначення нейтральних позицій, так і механізм їхнього ручного калібрування через натискання клавіш у режимі реального часу. Це дозволяє точно адаптувати систему до індивідуальних особливостей користувача навіть без повторного запуску програми.

Окремим технічним аспектом реалізації є подвійне згладжування параметрів

міміки: перше виконується безпосередньо під час обчислення параметрів у Python, друге — перед застосуванням у Blender. Такий підхід був необхідним для досягнення високого рівня стабільності, оскільки навіть після початкового згладжування залишалися незначні стрибки значень. Завдяки додатковій фільтрації в Blender вдалося суттєво покращити сприйняття анімації.

Як підсумок можно скзати, що побудована система є комплексним і рішенням для відтворення міміки в реальному часі. Вона поєднує сучасні методи CV, обробку відеопотоку, математичну нормалізацію параметрів, механізми згладжування та анімацію в Blender. Гнучкість архітектури дозволяє масштабувати систему на нові моделі, додавати підтримку нових мімічних жестів та інтегрувати її у більш складні середовища, включно з ігровими або VR-платформами. Реалізована структура дозволяє не лише передавати вирази обличчя, але й забезпечує точність, реалістичність і синхронність, що робить її придатною для використання для анімаційних та дослідницьких задач.

# 4 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

Система анімації міміки персонажа у Blender працює в реальному часі, використовуючи два основних компоненти: Python-скрипт для зчитування міміки з відеопотоку і Blender-скрипт для застосування цих даних до 3D-моделі. У цьому розділі буде пояснено, як запустити скрипт для захоплення міміки і підключити Blender-скрипт для анімації персонажа.

## 4.1 Запуск скрипта захоплення міміки

Для початку потрібно відкрити термінал і виконати команду: «python main.py» 4.1.

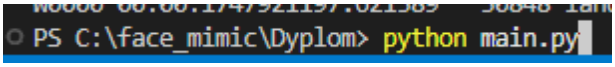



Рис. 4.1. Запуск скрипту

Після запуску відкриється вікно з відеопотоком, у якому система почне в реальному часі зчитувати координати ключових точок обличчя 4.2.

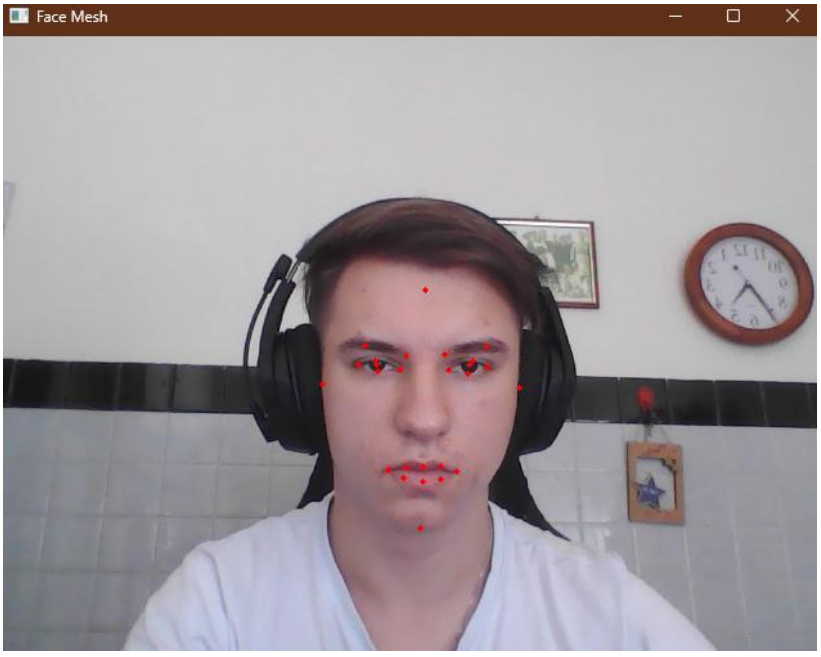



Рис. 4.2. Вікно з відеопотоком та ключовими точками обличчя

Далі користувач повинен виконати калібрування нейтрального стану. Це гарантує правильність подальших розрахунків:

1. Спочатку необхідно прийняти нейтральну позу обличчя (розслаблений вираз, дивитись прямо в камеру) і натиснути клавішу n. Це збереже початкове положення очей, брів та губ, яке буде використовуватись як базове для нормалізації рухів 4.3.

Рис.4.3. Калібрування нейтрального виразу обличчя

2. Далі необхідно відцентрувати погляд, дивлячись прямо в камеру, і натиснути клавішу j. Це зафіксує нейтральне положення зіниць, що забезпечить точніше відстеження напрямку погляду.

3. Після цього потрібно прищулитись, так щоб очі були частково закриті, і натиснути клавішу s. Це допоможе системі розпізнати мінімальний рівень відкритості очей, що дозволить коректно відтворювати емоції, такі як нахмурення або легке заплющування.

4. Також треба заплющити очі та натиснути z. Так система зафіксує поріг

при якому очі вважаються закритими. Приклад 4.4.

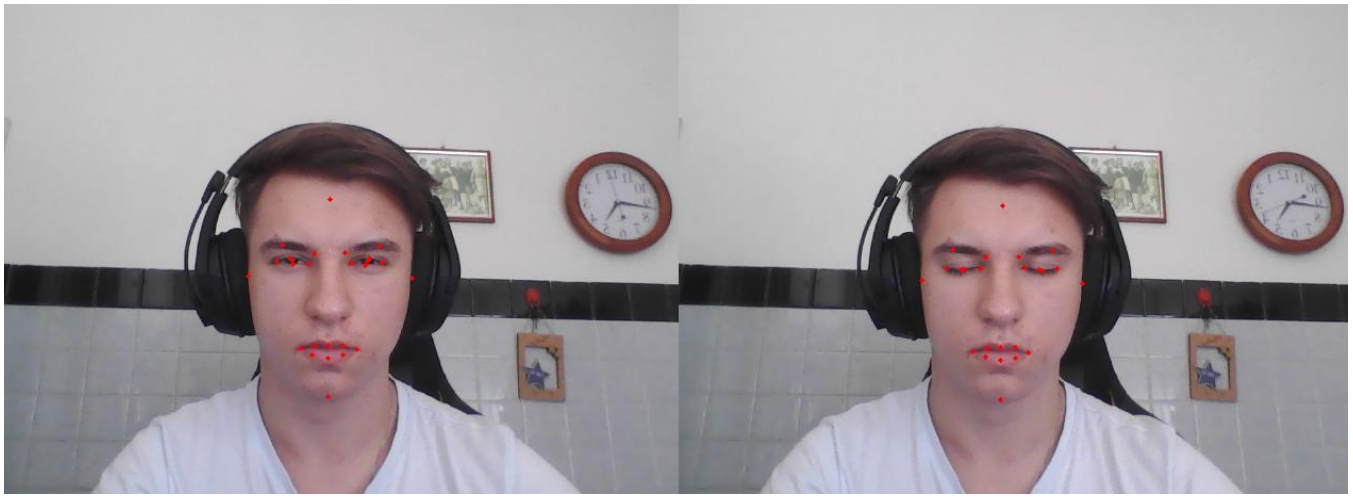


Рис.4.4. Калібрування прижмурювання та моргання

Кожна з цих дій записує значення у файл eye_calibration.json, який буде використано надалі для точного обчислення мімічних параметрів. Завдяки цьому система краще адаптується до обличчя конкретного користувача.

## 4.2 Підключення Blender-скрипта

Після завершення калібрування міміки у Python-скрипті необхідно перейти до другого компонента, анімації у Blender. Для цього спочатку відкрийте Blender і завантажте сцену, яка містить параметричну 3D-модель персонажа.

Завантаження сцени відбувається через меню File а потім Open (рисунок 4.5), після чого потрібно вказати шлях до файлу .blend, який був підготовлений заздалегідь.

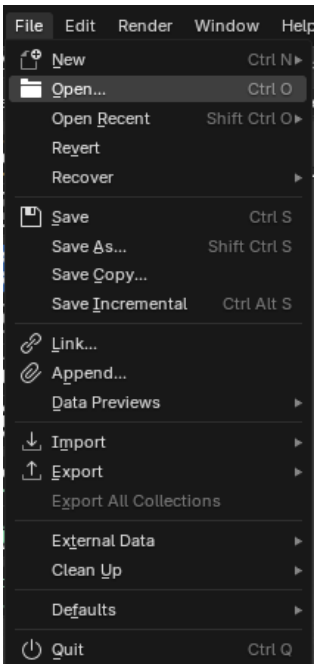



Рис.4.5. Завантаження сцени

У цьому файлі повинна бути арматура з іменем blender file_Rigify, оскільки саме до неї звертається скрипт. Якщо ім’я буде іншим, скрипт не зможе знайти потрібні кістки і не працюватиме (рисунок 4.6).

Рис.4.6. Вигляд завантаженої сцени з 3D-моделлю та арматурою Rigify

Завантаживши сцену перейдіть у редактор тексту scripting, який знаходиться у верхній частині інтерфейсу (рисунок 4.7)

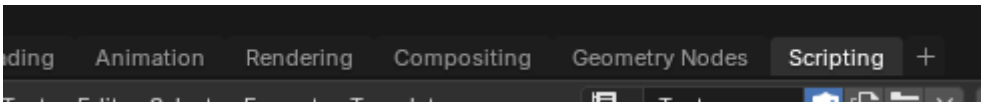



Рис.4.7. Відкриття редактору тексту

У Scripting натисніть кнопку Text, а в ній Open, щоб завантажити файл Text.py. Це основний скрипт, який відповідає за зчитування mimic.json і оновлення міміки в реальному часі (рисунок 4.8).

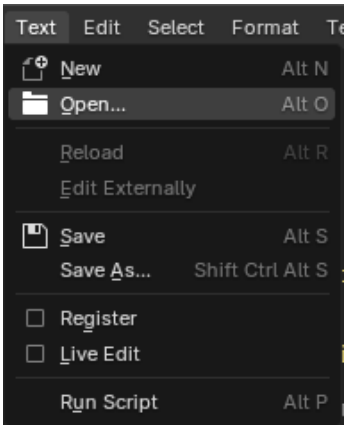



Рис.4.8. Завантаження скрипту

Перед запуском скрипта обов’язково перевірте змінну mimic_path у першому блоці коду. Вона повинна вказувати на повний шлях до файлу mimic.json, який створює main.py. Якщо шляхи не збігаються, Blender не зможе знайти потрібні дані. Приклад: «mimic_path = r"C:\face_mimic\Dyplom\mimic.json"»

Після цього натисніть кнопку для запуску скрипту (рисунок 4.9). Скрипт буде запущено у фоновому режимі, і функція update_from_params() почне викликатись автоматично кожні 1/30 секунди. Така частота оновлення обрана для синхронізації з частотою оновлення mimic.json, що формується у першому модулі через OpenCV.

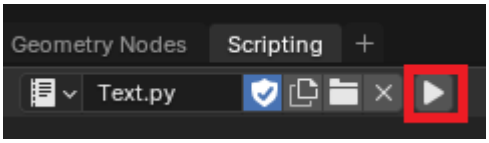



Рис.4.9. Завантаження скрипту

На цьому етапі Blender почне автоматично зчитувати міміку з mimic.json і застосовувати її до 3D-моделі. Рухи щелепи, губ і очей керуються кістками, тоді як більш делікатна міміка — примружування, моргання, посмішка, нахмурення — реалізована через shape keys (рисунок 4.10).

Рис.4.9. Анімація персонажа в реальному часі з використанням параметрів mimic.json

Після запуску обох модулів (Python і Blender) користувач бачить повністю синхронізовану міміку на 3D-моделі. Це дозволяє спостерігати за реалістичними виразами, та використовувати систему в реальному часі для стрімінгу, запису відео або VR-проєктів. Blender-скрипт стабільно працює у фоновому режимі, забезпечуючи постійне оновлення навіть за умов зміни освітлення, що є приємним бонусом.

## Висновки до розділу 4

У цьому розділі ми зосередились на описі взаємодії користувача з системою передачі міміки. Основна увага приділялася послідовності дій за якою користувачу рекомендується діяти для коректної роботи системи, що складається з двох компонентів, скрипта зчитування міміки і скрипта для відтворення цієї міміки на 3D-моделі.

Сам процес починається з запуску main.py, який захоплює зображення с вебкамери і визначає координати ключових точок обличчя, після цього формує набір мімічних параметрів які використовуються у розрахунках. На початку було приділено увагу ручному калібруванню нейтрального стану обличчя, яке користувач виконує за допомогою клавіш, n, j, s , z. Калібрування потрібне для того щоб врахувати індивідуальні особливості користувача, таки як: положення огей, брів, губ, тощо. Калібрування за допомогою клавіш зроблено для того щоб уникнути випадкові помилки при шуму у відео потоці, або випадковостей що можуть трапитись у користувача. Також такий спосіб дозволяє провести калібрування навіть якщо скрипт працює деякий час без потрібності в перезапуску скрипта.

Наступним етапом є підключення скрипта Text.py у середовищі Blender. Було показано, як правильно завантажити сцену, у якій вже присутня арматура з очікуваним іменем, і як через вбудований редактор тексту виконати цей скрипт. Детально описано структуру інтерфейсу Blender та місця, де користувач має знайти вкладку Scripting, а також перевірити шлях до файлу mimic.json у змінній mimic_path.

Реалізована система має простий та інтуїтивно зрозумілий механізм налаштування, який робить її зручною у використанні навіть для новачків. Всі кроки, від запуску до першої анімації, були описані із зазначенням ключових дій, супроводжені наочними прикладами у вигляді скріншотів. Це дозволяє користувачам швидко зорієнтуватися у процесі, уникати помилок та досягати

стабільної роботи системи.

Система передбачає одночасну роботу обох модулів, які взаємодіють через буферний файл mimic.json. Завдяки цьому забезпечується синхронне відображення міміки користувача на параметричному персонажі. Весь процес максимально автоматизовано — користувачеві потрібно лише запустити скрипти та провести одноразове калібрування. Надалі система працює автономно, стабільно оновлюючи анімацію в реальному часі, навіть за умов зміни положення голови або освітлення.

Таким чином, описаний інтерфейс взаємодії з системою охоплює всі технічні аспекти її роботи з точки зору користувача. Водночас він є достатньо гнучким і масштабованим для подальшого розширення. У наступних версіях можливо додати більш гнучке управління всією головою, більш детальну підтримку міміки при розмові або інтеграцію з іншими програмними середовищами наприклад, стрімінговими платформами або ігровими рушіями.

# ВИСНОВКИ

В результаті виконання даної роботи було розроблено систему передачі міміки користувача на 3D-модель персонажа у Blender. Система включає два основних компоненти: скрипт main.py для зчитування міміки з відеопотоку за допомогою MediaPipe та скрипт Text.py для застосування цих даних у Blender.

Процес розробки включав декілька ключових етапів, таких як:

- Визначення координат ключових точок обличчя для відстеження рухів
- Реалізація алгоритмів нормалізації та фільтрації для точного визначення міміки
- Передача даних у Blender для відтворення анімації в реальному часі
- Налаштування shape keys для плавного відтворення складних виразів обличчя.

Архітектура системи була спроектована з урахуванням потреб у точності передачі міміки та мінімізації затримок. Основні компоненти, такі як згладжування сигналу, корекція положення очей, компенсація рухів рота та синхронізація даних, були реалізовані з акцентом на стабільність і продуктивність.

Для забезпечення точності рухів очей були впроваджені методи фільтрації та адаптивної нормалізації, що дозволяють уникнути помилкових тригерів для моргання та руху зіниць. Крім того, було реалізовано механізм калібрування, який дозволяє враховувати індивідуальні особливості обличчя користувача.

Подальші напрямки розвитку системи включають:

- більш гнучке управління всією головою;
- більш детальну підтримку міміки при розмові, напруження щік та рухи язика

Розроблена система передачі міміки є інструментом для створення наближеною до реалістичної анімації в реальному часі, забезпечує достатньо точну передачу емоцій та може бути адаптована для використання в інтерактивних застосунках.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Documentation. [Електронний ресурс]. URL: https://docs.python.org/3/ (дата звернення 07.05.2025р).

2. OpenCV Documentation. [Електронний ресурс]. URL: https://docs.opencv.org/4.x/ (дата звернення 08.05.2025р).

3. MediaPipe Documentation. [Електронний ресурс]. URL: https://developers.google.com/mediapipe (дата звернення 08.05.2025р).

4. JSON Official Website. [Електронний ресурс]. URL: https://www.json.org/json-en.html (дата звернення 09.05.2025р).

5. Python API Documentation. [Електронний ресурс]. URL: https://docs.blender.org/api/current/ (дата звернення 09.05.2025р).

# ДОДАТОК А

Розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ІАТЕ_АПЕПС_ТВ-11

Аркушів 9

Київ – 2025

Text.py

```
import bpy
import json
import os
import time

# КОНФІГУРАЦІЯ

# Шлях до файлу mimic.json
mimic_path = r"C:\face_mimic\Dyplom\mimic.json"

# Назва арматури
armature_name = "blender file_Rigify"

# Зіставлення назв кісток між системою та Rigify
bone_names = {
    "jaw_master": "jaw_master",
    "lips_L": "lips.L",
    "lips_R": "lips.R",
    "eyes": "eyes"
}

neutral = {}  # Зберігає нейтральні положення кісток

# ГЛОБАЛЬНІ ЗМІННІ

smoothed_open = 0.0  # Smoothed mouth open value
smoothed_eye_x_L = 0.0  # Smoothed left eye X position
smoothed_eye_y_L = 0.0  # Smoothed left eye Y position
smoothed_eye_x_R = 0.0  # Smoothed right eye X position
smoothed_eye_y_R = 0.0  # Smoothed right eye Y position

# Зберігання даних очей
eye_data = {
    "L": {"closure": 0.0, "x": 0.5, "y": 0.5},
    "R": {"closure": 0.0, "x": 0.5, "y": 0.5}
}

# Параметри анімації
jaw_rotate_x = 0.3

# ОСНОВНА ФУНКЦІЯ ОНОВЛЕННЯ
```

```
def update_from_params():
    """Main update function called by Blender timer"""
    global smoothed_open, smoothed_eye_x_L, smoothed_eye_y_L,
smoothed_eye_x_R, smoothed_eye_y_R

    # Перевірка на mimic.json
    if not os.path.exists(mimic_path):
        return 1 / 30

    # Пропускати кадр, якщо файл занадто новий
    if time.time() - os.path.getmtime(mimic_path) < 0.01:
        return 1 / 30

    # Завантаження даних імітації
    try:
        with open(mimic_path, "r") as f:
            data = json.load(f)
    except:
        return 1 / 30

    # Отримати об'єкт арматури
    armature = bpy.data.objects.get(armature_name)
    if not armature:
        return 1

    # Оновлення даних про рот та очі
    smoothed_open = data.get("mouth_open_norm", 0.0)
    corner_closer = data.get("mouth_corner_closer", 0.0)

    # Оновити дані про закриття очей
    eye_data["L"]["closure"] = data.get("eye_blink_left", 0.0)
    eye_data["R"]["closure"] = data.get("eye_blink_right", 0.0)

    # Оновлення даних про положення очей
    eye_data["L"]["x"] = data.get("eye_x_left_norm", 0.0)
    eye_data["L"]["y"] = data.get("eye_y_left_norm", 0.0)
    eye_data["R"]["x"] = data.get("eye_x_right_norm", 0.0)
    eye_data["R"]["y"] = data.get("eye_y_right_norm", 0.0)

    # Ініціалізувати нейтральне положення очей на першому кадрі
    if "neutral_eye_x_L" not in neutral:
```

```
        neutral["neutral_eye_x_L"] = eye_data["L"]["x"]
        neutral["neutral_eye_y_L"] = eye_data["L"]["y"]
        neutral["neutral_eye_x_R"] = eye_data["R"]["x"]
        neutral["neutral_eye_y_R"] = eye_data["R"]["y"]

    # Центруйте положення очей, віднімаючи нейтральні значення
    eye_data["L"]["x"] -= neutral["neutral_eye_x_L"]
    eye_data["L"]["y"] -= neutral["neutral_eye_y_L"]
    eye_data["R"]["x"] -= neutral["neutral_eye_x_R"]
    eye_data["R"]["y"] -= neutral["neutral_eye_y_R"]

    # Функція згладжування
    def smooth(prev, current, alpha=0.2):
        return current if prev is None else prev * (1 - alpha) + current *
alpha

    # Згладжування до положень очей
    smoothed_eye_x_L = smooth(smoothed_eye_x_L, eye_data["L"]["x"])
    smoothed_eye_y_L = smooth(smoothed_eye_y_L, eye_data["L"]["y"])
    smoothed_eye_x_R = smooth(smoothed_eye_x_R, eye_data["R"]["x"])
    smoothed_eye_y_R = smooth(smoothed_eye_y_R, eye_data["R"]["y"])

    # Дані про брови та посмішку
    brow_inner_L = data.get("brow_inner_L", 0.0)
    brow_inner_R = data.get("brow_inner_R", 0.0)
    brow_outer_L = data.get("brow_outer_L", 0.0)
    brow_outer_R = data.get("brow_outer_R", 0.0)
    smile_L_raw = data.get("smile_L", 0.0)
    smile_R_raw = data.get("smile_R", 0.0)

    # Обробка кісток
    for key, bone_name in bone_names.items():
        bone = armature.pose.bones.get(bone_name)
        if not bone:
            continue

        # Ініціалізувати нейтральне положення, якщо не встановлено
        if key not in neutral:
            if key == "jaw_master":
                bone.rotation_mode = 'XYZ'
                neutral[key] = 0.0
            else:
```

```
            neutral[key] = bone.location.copy()

    # SHAPE KEYS

    # Знайти сітчастий об'єкт за допомогою ключів форми
    mesh_obj = None
    for child in armature.children:
        if child.type == 'MESH' and child.data.shape_keys:
            mesh_obj = child
            break

    if mesh_obj:
        key_blocks = mesh_obj.data.shape_keys.key_blocks

        # Допоміжна функція для фіксації значень
        def clamp_01(v):
            return max(0.0, min(1.0, v))

        # Get eye open and squint data
        squint_L = clamp_01(data.get("eye_squint_left", 0.0))
        squint_R = clamp_01(data.get("eye_squint_right", 0.0))

        # --- Eye Squint ---
        if "Eye_Squint_L" in key_blocks:
            key_blocks["Eye_Squint_L"].value = squint_L
        if "Eye_Squint_R" in key_blocks:
            key_blocks["Eye_Squint_R"].value = squint_R

        # --- Eye Blink ---
        raw_blink_L = clamp_01(data.get("eye_blink_left", 0.0))
        raw_blink_R = clamp_01(data.get("eye_blink_right", 0.0))

        # Зм моргання під час примруження
        blink_L = clamp_01(raw_blink_L * (1.0 - squint_L * 1.2))
        blink_R = clamp_01(raw_blink_R * (1.0 - squint_R * 1.2))

        # --- Brow Raise/Drop Inner ---
        brow_y_L_norm = brow_inner_L  # [-1..1]
        brow_y_R_norm = brow_inner_R

        # Ініціалізувати значення згладжування брів
        if "brow_L_smooth" not in neutral:
```

```
                neutral["brow_L_smooth"] = brow_y_L_norm
            if "brow_R_smooth" not in neutral:
                neutral["brow_R_smooth"] = brow_y_R_norm

            # Застосувати згладжування
            alpha_brow = 0.15
            neutral["brow_L_smooth"] = (1 - alpha_brow) *
neutral["brow_L_smooth"] + alpha_brow * brow_y_L_norm
            neutral["brow_R_smooth"] = (1 - alpha_brow) *
neutral["brow_R_smooth"] + alpha_brow * brow_y_R_norm

            # Обчислення значення підвищення та зниження
            raise_L = max(0.0, neutral["brow_L_smooth"])
            raise_R = max(0.0, neutral["brow_R_smooth"])
            drop_L = max(0.0, -neutral["brow_L_smooth"])
            drop_R = max(0.0, -neutral["brow_R_smooth"])

            # Застосування brow shape keys
            if "Brow_Raise_Inner_L" in key_blocks:
                key_blocks["Brow_Raise_Inner_L"].value = raise_L
            if "Brow_Raise_Inner_R" in key_blocks:
                key_blocks["Brow_Raise_Inner_R"].value = raise_R
            if "Brow_Drop_L" in key_blocks:
                key_blocks["Brow_Drop_L"].value = drop_L
            if "Brow_Drop_R" in key_blocks:
                key_blocks["Brow_Drop_R"].value = drop_R

            # --- Brow Raise Outer ---
            if "brow_outer_L_smooth" not in neutral:
                neutral["brow_outer_L_smooth"] = brow_outer_L
            if "brow_outer_R_smooth" not in neutral:
                neutral["brow_outer_R_smooth"] = brow_outer_R

            # Застосувати згладжування
            neutral["brow_outer_L_smooth"] = (1 - alpha_brow) *
neutral["brow_outer_L_smooth"] + alpha_brow * brow_outer_L
            neutral["brow_outer_R_smooth"] = (1 - alpha_brow) *
neutral["brow_outer_R_smooth"] + alpha_brow * brow_outer_R

            # Нормалізація значення
            brow_outer_L_norm = max(0.0, min(1.0,
neutral["brow_outer_L_smooth"]))
```

```
            brow_outer_R_norm = max(0.0, min(1.0,
neutral["brow_outer_R_smooth"]))

            # Застосування outer brow shape keys
            if "Brow_Raise_Outer_L" in key_blocks:
                key_blocks["Brow_Raise_Outer_L"].value = brow_outer_L_norm
            if "Brow_Raise_Outer_R" in key_blocks:
                key_blocks["Brow_Raise_Outer_R"].value = brow_outer_R_norm

            # --- Smile Control ---
            if "neutral_smile_L" not in neutral:
                neutral["neutral_smile_L"] = smile_L_raw
            if "neutral_smile_R" not in neutral:
                neutral["neutral_smile_R"] = smile_R_raw

            # Обчислення зміщення посмішки від нейтрального рівня
            delta_L = smile_L_raw - neutral["neutral_smile_L"]
            delta_R = smile_R_raw - neutral["neutral_smile_R"]

            # Ініціалізувати значення плавної посмішки
            if "smile_L_smooth" not in neutral:
                neutral["smile_L_smooth"] = delta_L
            if "smile_R_smooth" not in neutral:
                neutral["smile_R_smooth"] = delta_R

            # Застосувати згладжування
            alpha_smile = 0.3
            neutral["smile_L_smooth"] = (1 - alpha_smile) *
neutral["smile_L_smooth"] + alpha_smile * delta_L
            neutral["smile_R_smooth"] = (1 - alpha_smile) *
neutral["smile_R_smooth"] + alpha_smile * delta_R

            # Нормалізувати значення посмішки
            scale_factor = 0.04
            smile_L_norm = max(0.0, min(1.0, neutral["smile_L_smooth"] /
scale_factor))
            smile_R_norm = max(0.0, min(1.0, neutral["smile_R_smooth"] /
scale_factor))

            # Застосування smile shape keys
            if "Mouth_Dimple_L" in key_blocks:
                key_blocks["Mouth_Dimple_L"].value = smile_L_norm
```

```
            if "Mouth_Dimple_R" in key_blocks:
                key_blocks["Mouth_Dimple_R"].value = smile_R_norm

            # --- Frown Control ---
            if "frown_L_smooth" not in neutral:
                neutral["frown_L_smooth"] = -delta_L
            if "frown_R_smooth" not in neutral:
                neutral["frown_R_smooth"] = -delta_R

            # Застосування згладжування
            neutral["frown_L_smooth"] = (1 - alpha_smile) *
neutral["frown_L_smooth"] - alpha_smile * delta_L
            neutral["frown_R_smooth"] = (1 - alpha_smile) *
neutral["frown_R_smooth"] - alpha_smile * delta_R

            # Нормалізуввання значення нахмурености за допомогою мертвої
зони
            frown_threshold = 0.001  # Ignore
            frown_L_raw = neutral["frown_L_smooth"]
            frown_R_raw = neutral["frown_R_smooth"]

            frown_L_norm = max(0.0, min(1.0, (frown_L_raw -
frown_threshold) / (scale_factor - frown_threshold)))
            frown_R_norm = max(0.0, min(1.0, (frown_R_raw -
frown_threshold) / (scale_factor - frown_threshold)))

            # Застосування frown shape keys
            if "Mouth_Frown_L" in key_blocks:
                key_blocks["Mouth_Frown_L"].value = frown_L_norm
            if "Mouth_Frown_R" in key_blocks:
                key_blocks["Mouth_Frown_R"].value = frown_R_norm

            # --- Mouth Pucker ---
            pucker = data.get("mouth_pucker", 0.0)
            if "Mouth_Pucker_Up_L" in key_blocks:
                key_blocks["Mouth_Pucker_Up_L"].value = pucker
            if "Mouth_Pucker_Up_R" in key_blocks:
                key_blocks["Mouth_Pucker_Up_R"].value = pucker
            if "Mouth_Pucker_Down_L" in key_blocks:
                key_blocks["Mouth_Pucker_Down_L"].value = pucker
            if "Mouth_Pucker_Down_R" in key_blocks:
                key_blocks["Mouth_Pucker_Down_R"].value = pucker
```

```
        # Застосуванняblink shape keys
        if "Eye_Blink_L" in key_blocks:
            key_blocks["Eye_Blink_L"].value = blink_L
        if "Eye_Blink_R" in key_blocks:
            key_blocks["Eye_Blink_R"].value = blink_R

        # BONE

        # Обертання щелепи залежно від відкритості рота
        if key == "jaw_master":
            bone.rotation_euler.x = smoothed_open * jaw_rotate_x

        # Рух куточків губ
        elif key == "lips_L":
            corner_range = 0.009  # На скільки рухати кутові кістки
            bone.location.x = neutral[key].x + corner_closer * corner_range

        elif key == "lips_R":
            corner_range = 0.009
            bone.location.x = neutral[key].x - corner_closer * corner_range

        # Рух очей
        elif key == "eyes":
            # Вик ліве око, якщо праве око закрите
            use_left_eye = blink_R >= 1.0

            # Ініціалізація положень очей
            if "eyes_base_x" not in neutral or "eyes_base_y" not in
neutral:
                neutral["eyes_base_x"] = bone.location.x
                neutral["eyes_base_y"] = bone.location.y
                neutral["eyes_neutral_x"] = eye_data["R"]["x"]
                neutral["eyes_neutral_y"] = smoothed_eye_y_L if
use_left_eye else smoothed_eye_y_R

            # Обчислення рух очей
            if use_left_eye:
                dx = smoothed_eye_x_L * 0.15 * 0.5  # eye_scale_x *
smoothing_compensate
                dy = smoothed_eye_y_L * 0.6 * 0.5   # eye_scale_y *
smoothing_compensate
```

```
            else:
                dx = smoothed_eye_x_R * 0.15 * 0.5  # eye_scale_x *
smoothing_compensate
                dy = smoothed_eye_y_R * 0.6 * 0.5   # eye_scale_y *
smoothing_compensate

            # Застосування руху очей
            bone.location.x = neutral["eyes_base_x"] + dx
            bone.location.y = neutral["eyes_base_y"] + dy

    bpy.context.view_layer.update()
    return 1.0 / 30

# CONTROL FUNCTIONS

def start_mimic():
    """Start the mimic animation"""
    global smoothed_open
    smoothed_open = 0.0
    bpy.app.timers.register(update_from_params)

def stop_mimic():
    """Stop the mimic animation"""
    bpy.app.timers.unregister(update_from_params)

def reset_pose():
    """Reset all bones to their neutral positions"""
    global smoothed_open
    smoothed_open = 0.0
    armature = bpy.data.objects.get(armature_name)
    if not armature:
        return
    for key, bone_name in bone_names.items():
        bone = armature.pose.bones.get(bone_name)
        if bone:
            if key == "jaw_master":
                bone.rotation_mode = 'XYZ'
                bone.rotation_euler.x = 0.0
            elif key in neutral:
                bone.location = neutral[key].copy()
    bpy.context.view_layer.update()
start_mimic()
```

# ДОДАТОК Б

Розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача

Презентація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ІАТЕ_АПЕПС_ТВ-11

Аркушів 25

Київ – 2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

**«Розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача»**


Виконав: Гришай Даніїл Дмитрович, ТВ-11
Дипломний керівник: Залевська Ольга Валеріївна

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

# «Розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача»

Гришай Даніїл Дмитрович, ТВ-11
Залевська Ольга Валеріївна, доцент к.т.н.

Київ - 2025

## Актуальність теми

- Основна проблема - необхідність переносити міміку користувача на віртуального персонажа з точністю і без використання дорогостоючого обладнання. А також покращення логіки зчитування міміки та її трансляції
- Значення для галузі інженерії програмного забезпечення є значним, оскільки створення таких систем потребує розробки алгоритмів обробки відеопотоку, аналізу мімічних параметрів та інтеграції з графічними рушіями. Особливу цінність має можливість побудови універсальної системи, що працює лише з використанням вебкамери та забезпечує реалістичну анімацію без дорогого трекінгового обладнання.

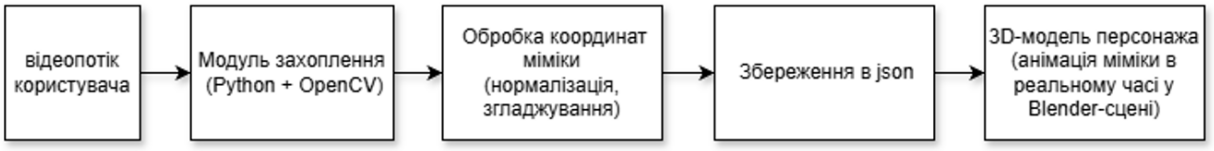


## Мета і завдання

Мета - розробка програмних модулів для інтеграції параметричних моделей обличчя з системою передачі міміки користувача в реальному часі

Основні завдання:
- Обробка відеопотоку та виявлення ключових точок обличчя
- Розрахунок мімічних параметрів
- Калібрування нейтрального стану обличчя
- Передача параметрів у Blender
- Анімація за допомогою shape keys та кісток

## Архітектура системи






## Модуль захоплення міміки

- Застосовується MediaPipe Face Mesh
- Визначаються: положення очей, брів, рота, зіниць
- Обчислюються параметри: mouth_open, blink, brow_y, iris_x/y тощо
- Результати нормалізуються і згладжуютьсяЗберігаються у mimic.json

## Калібрування нейтрального стану

Користувач вручну фіксує:

- Нейтральний вираз обличчя (n)
- Центрування погляду (j)
- Примруження (s)
- Повне закриття очей (z)

Це дозволяє адаптувати систему під індивідуальні особливості обличчя.

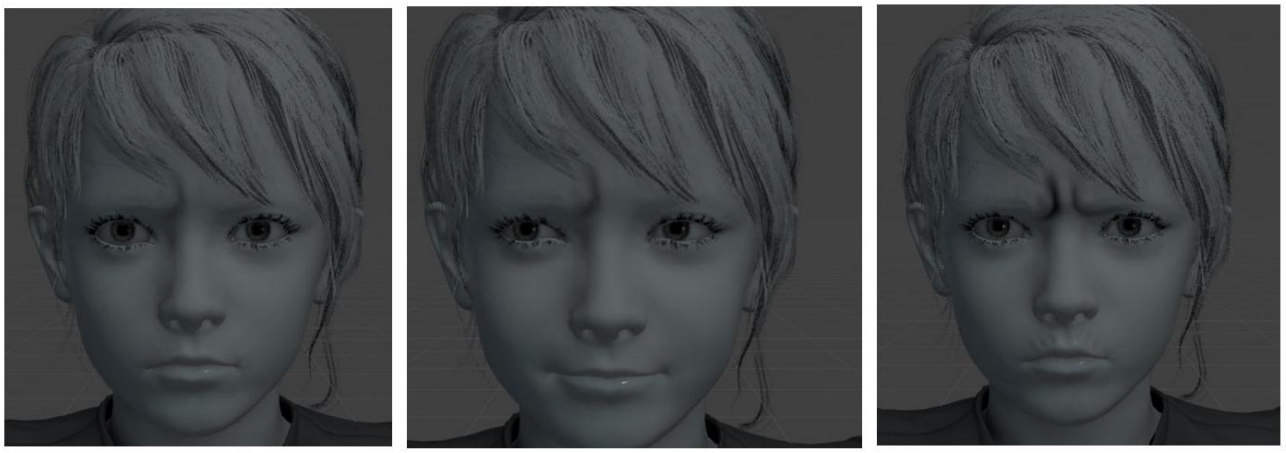


НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

## Модуль анімації у Blender

- Зчитування mimic.json через Text.py
- Кістки (bones) керують ротом, очима, щелепою
- Shape keys керують морганням, посмішками, брівками
- Підтримується подвійне згладжування для плавності
- Оновлення відбувається з частотою 1/30 с

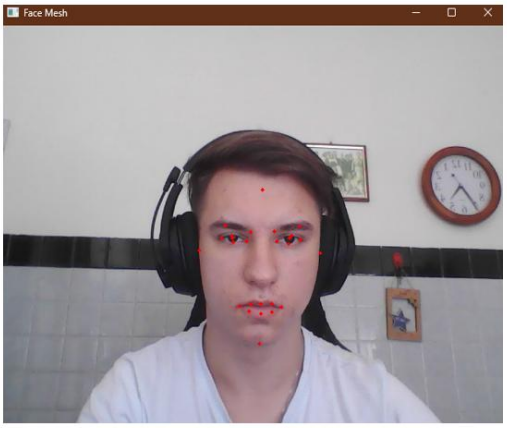
## Приклад анімації





НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

## Інтерфейс користувача



## Використані технології

- Python
- OpenCV
- MediaPipe
- JSON
- Blender
- Shape Keys + Bone Drivers



## Подальший розвиток

- Додавання мовлення
- Підтримка більше shape keys: морщини, щоки, язик
- Оптимізація та інтеграція в ігрові рушії, або окремий проект саме для трансляції міміки з можливістю зміни аватару

## Висновки

В результаті виконання дипломної роботи було розроблено програмний продукт дозволяючий переносити міміку користувача на віртуального персонажа.

Основні аспекти продукту:

- Зчитування mimic.json через Text.py
- Кістки (bones) керують найбільш рухливими частинами міміки
- Shape keys керують дрібними елементами міміки
- Підтримується подвійне згладжування для плавності
- Оновлення відбувається з частотою 1/30 с



# Дякую за увагу!