ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Аддон в Blender на мові Python для зручного переносу моделей MetaHuman»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ілля БЕРЕЖНИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав: здобувач вищої освіти гр. КНД-41
Ілля БЕРЕЖНИЙ

Керівник: Віктор ВИШНІВСЬКИЙ
науковий ступінь, вчене звання д.т.н., професор

Рецензент:
науковий ступінь, вчене звання _____
(Ім'я, ПРІЗВИЩЕ)

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бережному Іллі Валентиновичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Аддон в Blender на мові Python для зручного переносу моделей MetaHuman

керівник кваліфікаційної роботи Віктор ВИШНІВСЬКИЙ д.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. № 56

2. Строк подання кваліфікаційної роботи «30» травня 2025р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація Blender та MetaHuman, API Blender Python, прикладні матеріали з роботи з 3D-скелетами.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Огляд та аналіз сучасних технологій для роботи з 3D-моделями MetaHuman у Blender.

Дослідження особливостей підготовки моделей MetaHuman для подальшого використання у різних середовищах.

Розробка функціонального аддону для Blender: архітектура, принципи роботи, реалізація основних можливостей.

5. Перелік графічного матеріалу: *презентація*

1. Схема етапів підготовки моделі MetaHuman у Blender вручну та за допомогою аддону.

2. Інтерфейс розробленого аддону, приклади використання основних функцій.

4. Приклади сцен до та після автоматизованої підготовки моделей.

5. Фрагменти коду Python з поясненням логіки роботи ключових модулів аддону.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз науково-технічної літератури щодо роботи з MetaHuman та Blender	25.02-09.03.25	Виконано
2	Огляд існуючих інструментів для обробки моделей MetaHuman у Blender	09.03-16.03.25	Виконано
3	Аналіз проблем підготовки моделей MetaHuman та потреби в автоматизації	17.03-23.03.25	Виконано
4	Формування вимог до функціоналу майбутнього аддону	24.03-29.03.25	Виконано
5	Вибір інструментів розробки	01.04-07.04.25	Виконано
6	Розробка аддону: написання коду, реалізація основних функцій	08.04-24.04.25	Виконано
7	Оформлення роботи: вступ, висновки, реферат	25.04-02.05.25	Виконано
8	Розробка демонстраційних матеріалів	03.05-08.05.25	Виконано

Здобувач вищої освіти _____ Ілля Бережний
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Віктор ВИШНІВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 3 табл., 27 рис., 20 джерела.

Мета роботи – дослідження процесу адаптації тривимірних моделей MetaHuman до середовища Blender та розробка програмного засобу (аддону) на мові Python для зручного імпорту, переносу та підготовки моделей MetaHuman до використання в анімаційних проєктах.

Об'єкт дослідження – програмне середовище тривимірного моделювання Blender.

Предмет дослідження – процеси автоматизації імпорту та підготовки моделей MetaHuman.

Короткий зміст роботи:

У роботі проведено аналіз особливостей моделей MetaHuman, створених у Unreal Engine, та труднощів, які виникають при спробі перенесення таких моделей у Blender. Розглянуто структуру FBX-файлів MetaHuman, типову побудову мешів та скелетної структури.

Для спрощення та прискорення процесу імпорту моделей до Blender було розроблено спеціальний аддон на мові Python. Цей аддон забезпечує:

- автоматичне зчитування FBX-файлів моделей MetaHuman;
- налаштування масштабів та координат моделі відповідно до середовища Blender;
- виключення зайвих елементів сцени (камери, лайт-об'єкти тощо);
- зручне групування об'єктів сцени;
- підготовку моделі до подальших взаємодій

КЛЮЧОВІ СЛОВА: BLENDER, METAHUMAN, PYTHON, АДДОН, 3D-МОДЕЛЬ, ІМПОРТ, АВТОМАТИЗАЦІЯ, FBX

ЗМІСТ

ВСТУП

У сучасному світі комп’ютерної графіки та 3D-моделювання значну роль відіграють інструменти, що забезпечують високу якість візуалізації, реалістичність персонажів і ефективність у розробці цифрового контенту. Зі зростанням попиту на інтерактивні технології — у відеоіграх, кінематографі, віртуальній та доповненій реальностях — перед розробниками постають завдання створення якісних цифрових людей із високим рівнем деталізації та реалістичної анімації.

Однією із сучасних платформ для створення реалістичних тривимірних персонажів є MetaHuman Creator, розроблений компанією Epic Games. Цей інструмент дозволяє користувачам без глибоких знань 3D-моделювання створювати високополігональні моделі людей із фотореалістичною шкірою, волоссям, мімікою та скелетною структурою, готових до анімації. Однак, попри можливості MetaHuman, ці моделі спочатку орієнтовані на використання в екосистемі Unreal Engine, що створює низку труднощів для фахівців, які працюють у інших середовищах, таких як Blender — безкоштовний і відкритий інструмент для моделювання.

Імпорт моделей MetaHuman до Blender часто пов’язаний із рядом проблем: некоректне масштабування, неправильне позиціювання об’єктів, наявність зайвих елементів сцени (камер, лайт-об'єктів), складність у злитті мешів та управлінні скелетною структурою. Більшість цих завдань доводиться виконувати вручну, що забирає багато часу, створює ризики помилок і ускладнює подальшу роботу з моделлю. Саме тому виникає потреба в автоматизації процесу перенесення моделей MetaHuman до Blender.

Особливу складність у роботі з моделями MetaHuman становить їхня скелетна структура (риг), яка створена відповідно до вимог Unreal Engine і не є безпосередньо сумісною з більшістю середовищ для анімації, зокрема Blender або Mixamo. Через це виникають труднощі при спробі застосувати до моделі сторонні

анімації або адаптувати її до інших стандартів ригінгу. У межах розробленого аддону реалізовано функціональність, яка створює новий скелет з нуля на основі структури Mixamo. Цей скелет автоматично масштабується відповідно до пропорцій моделі MetaHuman, забезпечуючи точне позиціювання кісток у просторі сцени. Після побудови нового скелета, оригінальний MetaHuman-риг, який є надто складним і непотрібним для подальшої роботи, повністю видаляється.

У таких умовах доцільною є розробка спеціалізованого аддону (розширення) для Blender, написаного мовою Python, який здатен автоматизувати ключові етапи підготовки моделей до роботи: імпорт FBX-файлів, масштабування, очистку сцени, групування об'єктів, а також адаптацію структури моделі до специфіки Blender. Такий інструмент дозволить не лише заощадити час і зусилля користувачів, а й забезпечить стандартизацію та уніфікацію процесу, що особливо важливо в командних або комерційних проєктах.

Розробка подібного інструменту відповідає актуальним потребам індустрії, сприяє оптимізації творчого процесу та підвищенню якості кінцевого результату. У результаті користувачі зможуть зосередитися на творчих аспектах розробки — анімації, рендері, освітленні — замість вирішення технічних труднощів при перенесенні моделей.

Об’єкт дослідження – програмне середовище тривимірного моделювання Blender.

Предмет дослідження – процес автоматизованого імпорту та підготовки моделей MetaHuman у Blender.

Мета роботи – дослідити технічні особливості моделей MetaHuman, проаналізувати труднощі їхнього імпорту до Blender і створити Python-аддон для автоматизації ключових операцій з підготовки моделей до використання в анімаційних та візуалізаційних проєктах.

Наукове завдання:

- вивчити структуру FBX-файлів, які створюються MetaHuman Creator;
- проаналізувати особливості взаємодії таких моделей із Blender;
- ідентифікувати основні труднощі при розробці аддону та налаштуванні

моделей;

- розробити Python-аддон, який автоматизує процес імпорту та адаптації моделей;
- протестувати аддон на прикладах моделей MetaHuman і оцінити його ефективність.

Практична значимість. Створення та впровадження аддону дозволяє спростити та стандартизувати роботу з моделями MetaHuman у Blender. Це відкриває нові можливості для розробників і дизайнерів, які використовують Blender у сфері відеоігор, віртуального виробництва, CGI-анімації та інших галузях, де важлива швидкість і якість роботи з цифровими персонажами.

1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОБОТИ З МОДЕЛЯМИ METAHUMAN У BLENDER

1.1 Сучасні тенденції у сфері 3D-моделювання

У XXI столітті моделювання стало однією з ключових технологій, що формує вигляд сучасної цифрової індустрії. 3D-моделі застосовуються не тільки у створенні відеоігор та кінофільмів, але й у віртуальній реальності, доповненій реальності, архітектурі, медицині, промисловому дизайні, освіті та наукових дослідженнях. Такий великий спектр застосування зумовлює швидкий розвиток як програмного забезпечення, так і апаратних засобів для створення й обробки тривимірного контенту.

Одним із ключових напрямків розвитку стало створення цифрових персонажів, які зовнішньо та поведінково імітують людину. Зросла популярність процедурного моделювання, де значна частина рутинної роботи автоматизується за допомогою параметричних систем. Це дозволяє скоротити час на створення складних моделей та забезпечити гнучкість у зміні їх характеристик.

Особливу роль у сучасному 3D-виробництві відіграють дві потужні екосистеми — Blender та Unreal Engine 5 (UE5). Blender є відкритим програмним середовищем для 3D-моделювання, анімації, симуляцій та рендерингу. Він активно використовується як незалежними художниками, так і професіоналами в індустрії. Однією з головних переваг Blender є його модульність і активна підтримка спільнотою, що забезпечує постійне оновлення функціоналу, зокрема підтримку новітніх стандартів експорту, шейдерів, скриптів і аддонів.

Unreal Engine 5, у свою чергу, представляє нове покоління рушіїв реального часу, орієнтоване не лише на геймдев, але й на віртуальне виробництво, архітектурну візуалізацію та кіновиробництво. Нові функції, як-от Nanite (віртуалізована геометрія) та Lumen (глобальне освітлення в реальному часі), дозволяють працювати з високодеталізованими сценами без суттєвої втрати

продуктивності. UE5 також підтримує безшовну інтеграцію з MetaHuman Creator, що дозволяє створювати реалістичних персонажів зі складними мімічними та анімаційними можливостями.

Сучасна практика передбачає поєднання можливостей Blender та UE5 у межах єдиного виробничого циклу. Часто початкове 3D-моделювання та ригінг виконуються в Blender, після чого сцена або персонаж експортується в форматі FBX або glTF для подальшого використання в UE5. Blender також широко використовується для виправлення моделей MetaHuman, оскільки він дозволяє точково змінювати геометрію, налаштовувати ваги моделі, редагувати матеріали та оптимізувати структуру кісток перед експортом. Серед тенденцій також варто зазначити інтеграцію різних програмних платформ для оптимізації робочого процесу. Для прикладу можна взяти моделі, які створені в одній системі, дедалі частіше адаптуються до використання в інших середовищах (наприклад, імпорт із Unreal Engine до Blender або навпаки). Це потребує підтримки універсальних форматів, а також розробки спеціалізованих плагінів і аддонів. Ця інтеграція стимулює появу нових інструментів — наприклад, аддони для автоматичного ретаргетингу анімацій (Mixamo в MetaHuman), або просто скрипти для збереження орієнтації кісток, пакети з налаштованими шаблонами експорту та інше.

Таким чином, сучасні тенденції у сфері 3D-моделювання вказують на все більшу взаємодію різних програмних платформ. Blender і Unreal Engine 5 виступають не лише як інструменти, а як можливе ядро кросплатформенних робочих процесів. Їх інтеграція забезпечує новий рівень якості, зручності та швидкості виробництва 3D-контенту, що є особливо актуальним в умовах зростання вимог до візуального реалізму, інтерактивності та персоналізації цифрових персонажів.

1.2 Інструменти створення цифрових персонажів

Створення цифрових персонажів є одним із ключових етапів у 3D-виробництві. Він охоплює розробку візуального образу, моделювання, текстурування, ригінг, анімацію та інтеграцію у віртуальне середовище. Для цього використовується широкий набір інструментів, таких як універсальні 3D-редактори, спеціалізовані програми, які дозволяють автоматизувати окремі етапи або працювати з високою точністю.

Одним із найпопулярніших інструментів у сучасному 3D-виробництві є Blender, безкоштовна програма з відкритим кодом, яка забезпечує повний цикл розробки 3D-персонажів. Blender дозволяє створювати складну геометрію, ліпити деталі на моделі, налаштовувати шейдери, створювати UV-мапи, а також застосовувати риггінг та анімацію. Завдяки гнучкості системи модифікаторів та підтримці скриптів Python, цей інструмент дозволяє налаштувати робочий процес відповідно до вимог конкретного проєкту.

Одним з важливим етапів є ригінг — процес створення скелетної структури для управління анімацією. Тут застосовуються як вбудовані інструменти Blender (Auto-Rig Pro, Rigify), так і сторонні рішення, які дозволяють швидко адаптувати персонажа для подальшого використання в анімаційних проєктах або ігровій індурстрії. Ригінг також включає в себе налаштування скіна на моделі (skin weights) — розподіл ваги між кістками й вершинами моделі, що безпосередньо впливає на реалістичність рухів.

Набирає обертів в сучасному процесі створення персонажів MetaHuman Creator — інструмент від Epic Games, який дозволяє створювати фотореалістичних людей з уже налаштованою мімікою, скелетом, текстурами та анімацією. Цифрові персонажі, створені за допомогою MetaHuman, сумісні з UE5, підтримують ліцеві анімації та автоматичну генерацію LOD-рівнів (рівнів деталізації), що значно спрощує інтеграцію у складні сцени.

На рисунку 1.1 наведен приклад сторінки, де створюється власний персонаж.

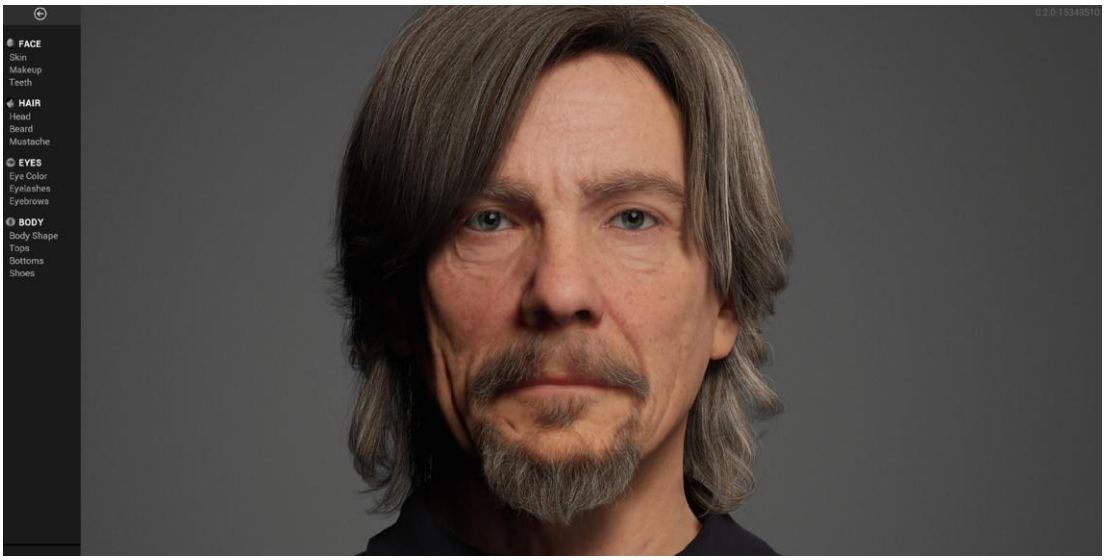


Рисунок 1.1 – Персонаж MetaHuman

Окремо слід відмітити Mixamo, який використовуються для автоматичної обробки ригінгу та застосування анімацій на 3D-персонажів. Mixamo підтримує імпорт моделей у форматі FBX або OBJ, автоматично додає скелет та дозволяє застосовувати та дивитися готові анімації прямо на моделі, що може бути корисно на етапах прототипування.

1.3 MetaHuman Creator: функціональні можливості

MetaHuman Creator — це хмарний сервіс, який дозволяє створювати високоякісних фотореалістичних 3D-персонажів. Ця система є частиною екосистеми Unreal Engine і пропонує унікальний підхід до генерації майже реальних людей, завдяки якому користувачі можуть отримати повноцінну, готову до використання 3D-модель без потреби в ручного моделювання та ригінгу.

Одним з плюсів MetaHuman Creator є інтуїтивно зрозумілий інтерфейс, що працює безпосередньо у браузері. Користувачі можуть швидко налаштовувати риси обличчя, колір шкіри, очей, зачіску, одяг та інші параметри зовнішності. Усі зміни, які користувач робить, відображаються у режимі реального часу з високою якістю візуалізації завдяки серверному рендерингу.

Функціональні можливості MetaHuman Creator охоплюють:

- повністю налаштовану 3D-модель з високою топологією, оптимізовану для реального часу. Моделі підтримують рівні деталізації (LOD), що робить їх придатними для використання в іграх, VR та фільмах;
- автоматизований ригінг: кожна створена модель містить повноцінний скелет, сумісний з Unreal Engine та іншими середовищами, а також з попередньо налаштованою мімікою, що дозволяє реалізувати емоції, мову, синхронізацію з озвученням тощо;
- вбудовану систему шейдерів і матеріалів таких як: шкіра, волосся, очі, зуби й одяг. Налаштовуються на основі процедурних текстур і матеріалів, які коректно відображаються в Unreal Engine.

Моделі MetaHuman можуть бути завантажені у вигляді проєктів для Unreal Engine, або експортовані в форматі FBX, що дозволяє їх подальшу обробку в Blender або інших 3D-редакторах. При цьому можливо зберегти основну структуру ригу, що дає змогу працювати з анімацією, деформацією тіла та мімікою.

1.4 Структура моделі MetaHuman з технічної точки зору

Моделі MetaHuman від Epic Games, є надзвичайно деталізованими цифровими персонажами. Вони мають складну технічну структуру, тому що розроблені з урахуванням реалістичної анатомії, фізіології та міміки, що дозволяє використовувати їх у високоякісних ігрових або кінематографічних проєктах.

Ця модель складається з кількох ключових компонентів:

- геометрія: тіло, голова, очі, ротова порожнина, волосся, одяг та аксесуари;
- система скелетної анімації: стандартизована скелетна система, яка включає розширену структуру обличчя. Це дозволяє керувати більш ніж 100 параметрами міміки та артикуляції;
- матеріали та шейдери: набір налаштувань, які визначають, як виглядає

кожна частина моделі.

На рисунку 1.2 та 1.3 наведені приклади MetaHuman у повний зріст та деталізації шкіри, очей та волосся.

Рисунок 1.2 – Персонаж MetaHuman у повний зріст

Рисунок 1.3 – Деталізації шкіри, очей та волосся

Для повноцінної роботи з нею в Blender необхідно розуміти всі ці технічні аспекти, адаптувати структуру під можливості Blender, а також враховувати різницю між анімаційними системами двох середовищ. Коли модель MetaHuman імпортується в Blender, всі ці матеріали теж передаються, але іноді їх потрібно переналаштувати вручну, бо Unreal Engine та Blender використовують різні методи для візуалізації.

1.5 FBX як ключовий елемент переносу моделей з Unreal Engine до Blender

Формат FBX є одним із поширених та найфункціональніших стандартів для обміну 3D-даними між різними програмними середовищами. У випадку MetaHuman Creator та Unreal Engine 5 можна вважати, що формат FBX відіграє ключову роль у правильному перенесенні моделей до Blender. Це дуже важливо для розробки персональних цифрових персонажів, яких потім потрібно анімувати або налаштовувати. Він підтримує перенесення складної геометрії, текстур, скінінгу та повного скелетного риггінгу, що, у свою чергу, дозволяє зберегти більшість параметрів моделі MetaHuman під час перенесення. Водночас, хоча це дуже гнучкий формат, є певні аспекти, які необхідно враховувати. Наприклад, структура назви кісток, орієнтація скелета та масштаб моделі можуть дещо змінитися під час імпорту в Blender, що в деяких випадках вимагає попереднього налаштування або використання доповнень.

У Unreal Engine експорт FBX передбачає декілька параметрів, що впливають на якість імпорту в Blender. До таких параметрів належать:

- вибір правильного рівня деталізації;
- включення або виключення анімацій;
- збереження матеріалів та текстур;

- масштаб об’єкта відносно систем координат.

Формат FBX є технічно необхідним і водночас критичним інструментом у робочій інтеграції моделей MetaHuman в Blender. Його правильне використання дозволяє досягти високого рівня точності при перенесенні персонажів і забезпечити основу для подальшого редагування, анімації або інтеграції в складні сцени.

1.6 Програмне середовище Blender

Blender — це безкоштовне 3D-програмне забезпечення з відкритим кодом. Воно використовується найбільшою кількістю 3D-художників, аніматорів, розробників ігор та фахівців з візуалізації. Завдяки своїй гнучкості, модульності та великому набору функцій, які можна назвати розширеними можливостями, Blender став універсальною платформою як для нових користувачів, так і для професіоналів. Моделювання, скульптинг, текстурування, рендеринг, анімація, композитинг, візуальні ефекти, відеомонтаж, а також створення інтерактивних додатків. Все це стало можливим без використання стороннього програмного забезпечення.

Однією з вагомих переваг Blender є підтримка повного виробничого циклу 3D-проєкту: моделювання, скульптинг, текстурування, рендеринг, анімація, композитинг, візуальні ефекти, відеомонтаж. Усе це можливо без потреби в сторонньому програм.

У контексті роботи з MetaHuman, Blender відіграє роль універсального редактора, в якому можна не лише переглядати та змінювати структуру моделі, а й створювати повноцінні сцени, анімації, або навіть готувати рендеринг для кінцевого продукту. Крім того, його дієспроможність робить його ідеальним середовищем для подальшого розвитку цифрових персонажів.

Важливою характеристикою Blender є глибока підтримка скриптів на Python, яка дозволяє розробляти власні інструменти, автоматизувати рутинні дії або

створювати повноцінні інтерфейси для взаємодії з користувачем. Завдяки цьому Blender перетворюється не лише на художній інструмент, а й на програмоване середовище, де можна реалізувати аддони будь-якої складності — від простих інструментів до комплексних систем ретаргету, як у випадку з MetaHuman to Mixamo. Такі функції, як створення кастомних ригів, конвертація скелетів, злиття мешів, застосування модифікаторів, автоматична оптимізація сцен — усе це можливо реалізувати безпосередньо в середовищі Blender без необхідності експорту в сторонні програми.

Blender постійно оновлюється, підтримується активною міжнародною спільнотою розробників і користувачів, а всі нововведення інтегруються відкрито через GitHub та Blender Foundation. Це робить платформу відкритою для досліджень і розширень. Інтеграція Blender у сучасні пайплайни дозволяє використовувати його не лише у творчих студіях, а й в освітніх установах, інженерних проєктах, медичних візуалізаціях та наукових симуляціях.

Таким чином, Blender виступає не лише як середовище для моделювання, а як повноцінна, гнучка і відкрита платформа для роботи з цифровими персонажами нового покоління. Його функціональність, підтримка Python та активна спільнота роблять його оптимальним інструментом для інтеграції з MetaHuman та реалізації складних 3D-задач.

1.7 Можливості кастомізації та автоматизації у Blender

Однією з ключових переваг Blender як відкритого 3D-редактора є широка підтримка кастомізації моделей і гнучка система автоматизації, яка дозволяє суттєво підвищити продуктивність роботи з персонажами, зокрема — з високополігональними моделями MetaHuman. Завдяки поєднанню ручного редагування, скриптових рішень та аддонів, Blender забезпечує повноцінну підтримку індивідуалізації персонажів та створення оптимізованого анімаційного середовища.

Blender надає широкі можливості для візуального та технічного налаштування моделей, що дозволяє адаптувати кожного персонажа під конкретні потреби проєкту. Основні можливості кастомізації включають:

1. Модифікацію геометрії: користувачі можуть редагувати форму обличчя, тіла, пальців, створювати унікальні риси або змінювати анатомічні пропорції. Це особливо актуально для створення стилізованих персонажів або адаптації персонажів до інших художніх стилів;
2. Роботу з текстурами і матеріалами: інструменти Blender дозволяють легко змінювати колір шкіри, очей, волосся, застосовувати мапи нормалей, шорсткості. Також можливе додавання декоративних елементів, таких як тату, шрами, макіяж або грим;
3. Налаштування одягу та аксесуарів: користувачі можуть імпортувати готовий одяг, або змоделювати його безпосередньо в Blender. При цьому можливе розміщення аксесуарів (окулярів, сережок, головних уборів) з точною посадкою до форми голови або вух.
4. Індивідуальна стилізація персонажа: шляхом комбінування геометричних змін, нових шейдерів і аксесуарів, можна створити повністю унікального героя, який візуально та концептуально відрізнятиметься від базових шаблонів.

Все це дозволяє зробити персонажа унікальним і повністю відповідним потребам проєкту.

Автоматизація у Blender базується переважно на підтримці мови Python, що дозволяє створювати власні скрипти або використовувати готові рішення для пришвидшення повторюваних дій. Це особливо важливо при роботі з технічно складними моделями, де вручну обробляти всі об’єкти було б неефективно:

1. Імпорт та попереднє налаштування MetaHuman за допомогою одного кліка. Це дозволяє видалити зайві об’єкти, обрати рівень деталізації, підлаштувати масштаб і структуру моделі автоматично, тощо;

2. Прив’язка анімацій до нових скелетів. Спеціальні аддони можуть автоматично адаптувати анімації, перетворювати ключові кадри й переприв’язувати кістки до нових структур;
3. Групові дії над об’єктами, як-от автоматичне перейменування, оптимізація шейдерів, злиття матеріалів, генерація проксі-мешів для попереднього перегляду або збереження анімацій у бібліотеку.

Також Blender має вбудований механізм обробки подій (event-handling), який дозволяє запускати дії на зміну сцени або імпорт нових об’єктів. Це відкриває шлях до створення повноцінних сценаріїв без участі користувача.

Переваги кастомізації і автоматизації:

1. Скоротити час підготовки сцен за рахунок зменшення кількості рутинних дій;
2. Знизити кількість технічних помилок, що виникають при ручному налаштуванні;
3. Легко масштабувати проєкт, коли потрібно підготувати велику кількість персонажів.

1.8 Порівняння скелетних структур MetaHuman та Mixamo

Відмінності скелетів MetaHuman та Mixamo наведені на рисунку 1.4.

Основні відмінності		
Характеристика	MetaHuman	Mixamo
Кількість кісток	Більше 150	Близько 70
Анімація обличчя	Так (кістки + blendshape-и)	Ні
Сумісність з UE5	Повна	Часткова
Точність анатомії	Висока	Середня
Призначення	Кінематографічні проєкти, ігри	Швидка анімація, прототипування

Рисунок 1.4 – Відмінності скелетів MetaHuman та Mixamo

При створенні та анімації 3D-персонажів дуже важливу роль відіграє скелетна структура моделі — набір кісток, який відповідає за рухи персонажа. MetaHuman і Mixamo — це дві популярні системи, які використовують свої унікальні скелети. Їх порівняння є важливим для коректного обміну анімаціями та інтеграції моделей у різні проєкти.

Скелет MetaHuman:

- більше кісток — включає додаткові кістки для обличчя;
- назви кісток відповідають стандартам Unreal Engine (UE5), що дозволяє використовувати готові анімації в UE без переналаштувань;
- анатомічна точність — кістки розташовані дуже близько до реального анатомічного положення.

Скелет Mixamo:

- менше кісток — стандартний скелет має близько 65–70 кісток, без деталізації голови;
- однакова структура для всіх персонажів — це зручно для швидкої анімації, але обмежує точність;
- підходить для базової анімації — ходьба, біг, жести руками тощо

- назви кісток стандартизовані, але не завжди сумісні з MetaHuman без ретаргетингу.

Оскільки скелети різні, анімації з Mixamo не можна напряму використовувати на MetaHuman. Потрібне ручне втручання для переносу саме анімацій.

1.9 Переваги інтеграції MetaHuman та Blender у сучасних проєктах

Інтеграція Metahuman та Blender вкрай актуальна у сфері цифрового виробництва, адже вона доволі зручна. Ця комбінація дозволяє створити високоякісних 3D-персонажів для різноманітних індустрій в набагато швидший час, ніж це все зробити власноруч. Актуальність цієї інтеграції зумовлена не лише технічною сумісністю, а й стратегічною вигодою для творчих індустрій.

MetaHuman дозволяє створювати надзвичайно реалістичних персонажів із детальним моделюванням шкіри, очей, волосся, а також багаторівневою структурою скелета. Blender, у свою чергу, забезпечує свободу кастомізації, потужні засоби анімації, підтримку Python-скриптів та можливість повної інтеграції з ігровими рушіями, такими як Unreal Engine. Разом ці інструменти формують гнучкий, модульний і недорогий процес, який підходить як для невеликих команд, так і для масштабних студій.

Переваги інтеграції:

- доступність і гнучкість: Blender безкоштовна платформа з відкритим кодом, яку можна легально використовувати у комерційних і некомерційних проєктах. Він підтримує імпорт і обробку складних моделей MetaHuman без потреби в дорогому ліцензійному забезпеченні. Завдяки цьому 3D-персонажі найвищого рівня деталізації стали доступними не лише великим студіям, а й фрілансерам та студентам;
- фотореалізм та деталізація: MetaHuman дозволяє створити повноцінного персонажа за лічені хвилини — з унікальними рисами обличчя, якісною

геометрією та текстурами шкіри. У зв’язці з Blender ці персонажі можуть бути швидко адаптовані, анімовані й візуалізовані. Це скорочує цикл виробництва ігрового чи анімаційного контенту в рази;

- інструменти кастомізації: Blender дає змогу редагувати не лише геометрію моделі, а й її матеріали, скелет, одяг, освітлення та навіть взаємодію з іншими об’єктами у сцені. Це робить можливим глибоке налаштування MetaHuman-персонажа під конкретні задачі: від серйозної драматичної ролі до стилізованого мультяшного героя;
- освітні та інді-проєкти: величезна кількість освітніх закладів і незалежних розробників використовує Blender як основний 3D-інструмент. Можливість інтеграції з MetaHuman значно розширює навчальні можливості: студенти можуть вивчати реалістичну анімацію, обробку тканин, міміку, не витрачаючи коштів на дорогі рішення.
- підвищення доступності професійного рівня: інтеграція Blender і MetaHuman ліквідує технологічний бар’єр, який раніше існував між аматорами й професіоналами. Сьогодні будь-який користувач з базовими знаннями Blender може створити кінематографічний персонаж із професійним ригом і експортувати його для використання в Unreal Engine або інших системах.

На рисунку 1.5 можна побачити приблизний час створення персонажу вручну та за допомогою зв’язці MetaHuman та Blender.

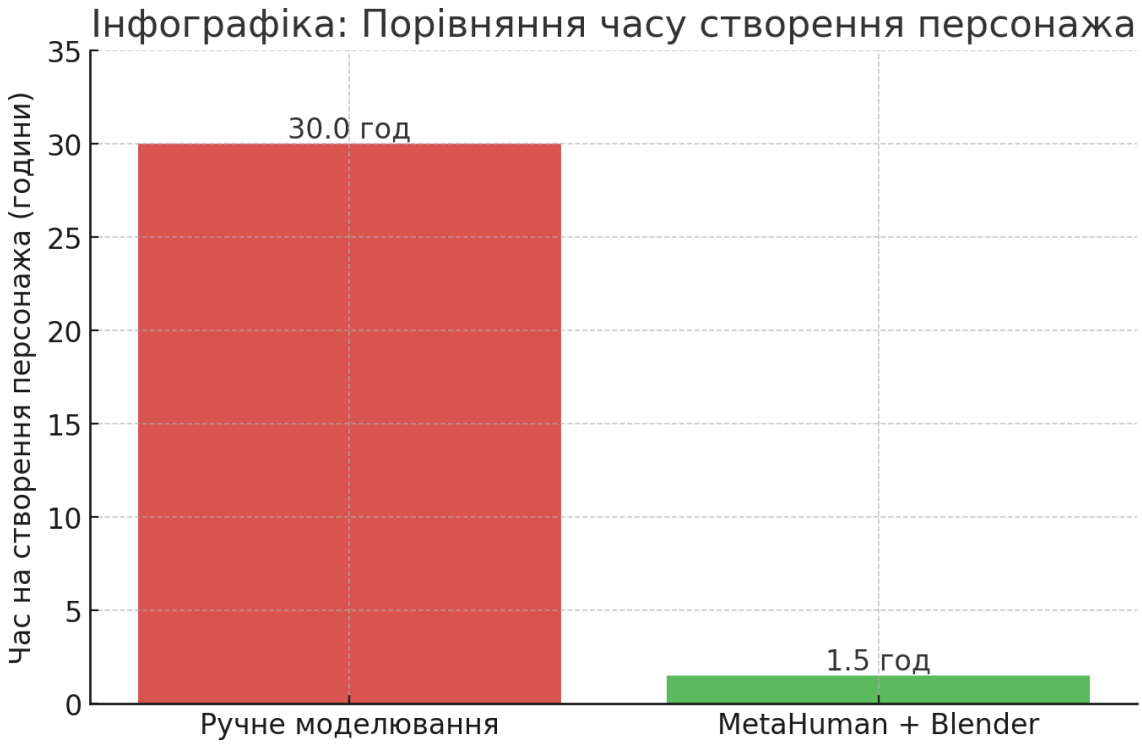


Рисунок 1.5 – Час створення персонажу

2 АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ АДДОНУ ДЛЯ ПІДГОТОВКИ МОДЕЛЕЙ METAHUMAN У BLENDER

2.1 Огляд технологій для підготовки моделей MetaHuman у Blender

2.1.1 Підготовка моделей MetaHuman

Моделі MetaHuman представляють собою високоякісні та детально опрацьовані тривимірні моделі, що використовуються в різноманітних цифрових середовищах. Проте, для забезпечення сумісності цих моделей з іншими програмними засобами, такими як Blender, необхідно здійснити ряд підготовчих етапів. Ці заходи включають модифікацію геометрії, заміну основного скелету, коригування текстур і матеріалів, а також загальну оптимізацію моделей для подальшого їх застосування.

Першим етапом є видалення наявного скелету MetaHuman. Основна проблема, що виникає при перенесенні моделей MetaHuman до інших програм або середовищ анімації, полягає в їхній початковій скелетній структурі. Скелет MetaHuman розроблений спеціально для інтеграції з внутрішніми механізмами Unreal Engine і може не відповідати вимогам інших платформ. На рисунку 2.1 можна побачити його.

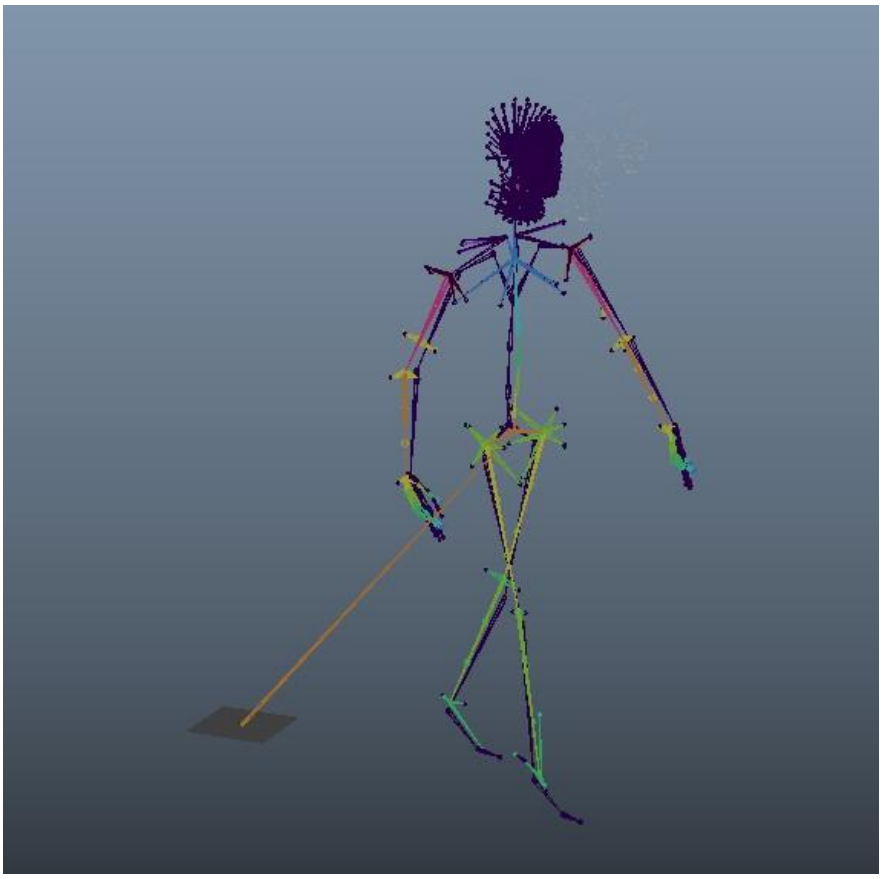


Рисунок 2.1 — Скелет MetaHuman

Наприклад, для роботи в Blender або анімування за допомогою зовнішніх систем, таких як Mixamo, часто потрібно впровадити скелет з іншим форматом і структурою. Таким чином, видалення початкового скелету є важливим кроком у процесі адаптації персонажів до різних інструментів і робочих середовищ. На рисунку 2.2 наведен приклад скелету Mixamo. Після видалення попереднього скелета наступним етапом є формування нового, що забезпечує сумісність з різними програмними інструментами та платформами. Спеціальний аддон для Blender автоматично виконує заміну скелета MetaHuman на альтернативний скелет Mixamo. Такий підхід усуває необхідність у ручному налаштуванні структури скелета, що суттєво прискорює процес підготовки моделі. Це, у свою чергу, підвищує її універсальність та адаптивність для інтеграції в різні 3D-системи.

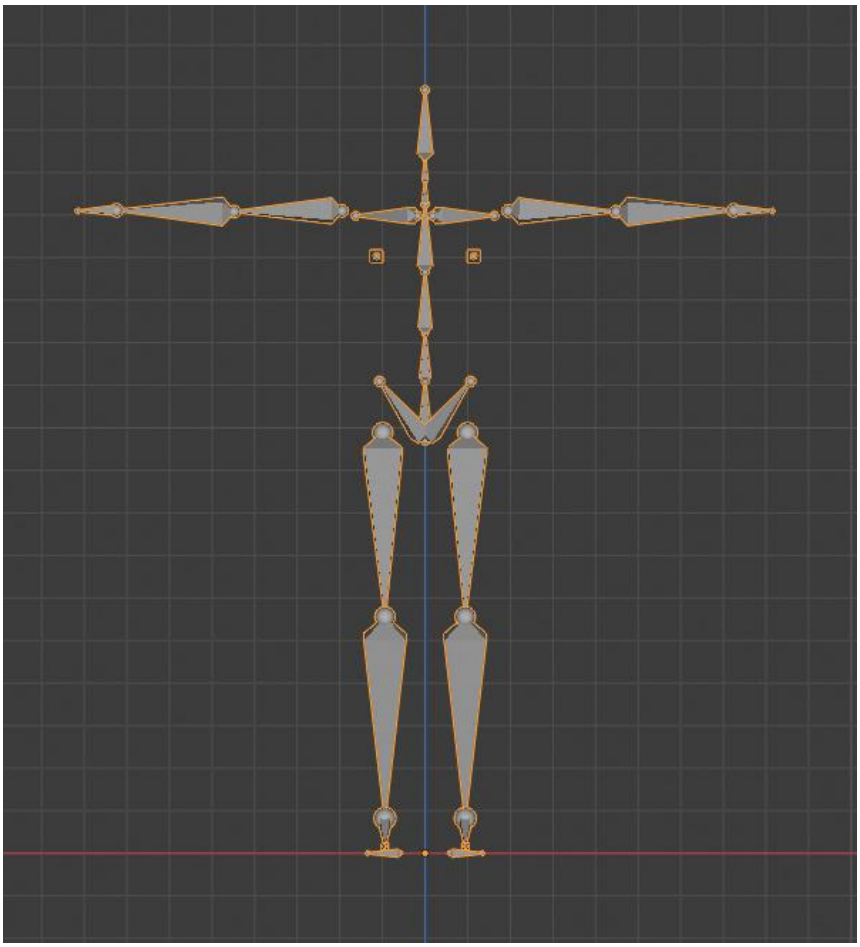


Рисунок 2.2 — Скелет Mixamo

Після створення нового скелета в Mixamo, аддон повинен автоматично вставляти його в модель, точно розташовуючи кістки відповідно до її геометрії. Цей процес є ключовим для збереження анатомічності моделі після зміни скелета. Завдяки цьому модель зберігає правильну структуру, уникаючи деформацій, які можуть виникати через некоректне позиціонування кісток.

2.1.2 Методи та інструменти для автоматизації підготовки моделей

Процес підготовки моделей MetaHuman для роботи в Blender або інших тривимірних системах є доволі складним. Він охоплює не лише заміну скелетних систем, але також передбачає ретельну обробку геометричних структур, текстурних матеріалів і оптимізацію моделі для подальшого редагування чи

інтеграції. З метою спрощення цих процедур застосовуються різноманітні інструменти та автоматизовані методи, які сприяють скороченню часу та зусиль, необхідних для підготовки моделей, а також знижують ризик виникнення помилок у процесі роботи. Суттєву допомогу в цьому аспекті надають спеціалізовані аддони для Blender, які автоматизують більшість ключових етапів адаптації.

Заміну старого скелета на новий, сумісний з різними 3D-системами, можна вважати одним із ключових етапів підготовки MetaHuman, бо він підходить під різні 3D-системами та інші сфери використання. Аддон для Blender дає змогу автоматично видалити попередній скелет і створити новий, оптимізований під стандарти Mixamo. Це необхідно задля інтеграції моделі з іншими платформами, такими як ігрові движки або програми для анімації.

Принцип функціонування:

- аддон самостійно визначає скелет MetaHuman;
- відображає користувачу налаштування для встановлення нового скелета, забезпечуючи збереження пропорцій моделі;
- автоматично виконує процес заміни, узгоджуючи кістки нового скелета з геометрією моделі, що гарантує коректну сумісність.

Також вагомою частиною із ключових завдань автоматизації є підготовка моделі для подальшого використання, що спрямована на забезпечення її адаптивності до різноманітних робочих середовищ. Цей процес може передбачати кілька важливих етапів:

- оптимізація моделі за рівнями деталізації (LOD);
- перевірка та налаштування матеріалів відповідно до особливостей цільових платформ.

Процес оптимізації рівнів деталізації є надзвичайно важливим. Такі моделі, використовуються у режимі реального часу. Скорочення кількості полігонів сприяє значному підвищенню продуктивності, що особливо актуально для великих сцен або ігрових проєктів. Інструменти, такі як аддони для Blender, надають можливість користувачу ефективно управляти рівнями деталізації моделі. Це дозволяє зберігати оптимальну кількість полігонів для різних елементів моделі, досягаючи

при цьому балансу між високою якістю візуалізації та продуктивністю. Приклади LOD різної ступені деталізації можна побачити на рисунку 2.3 та 2.4.

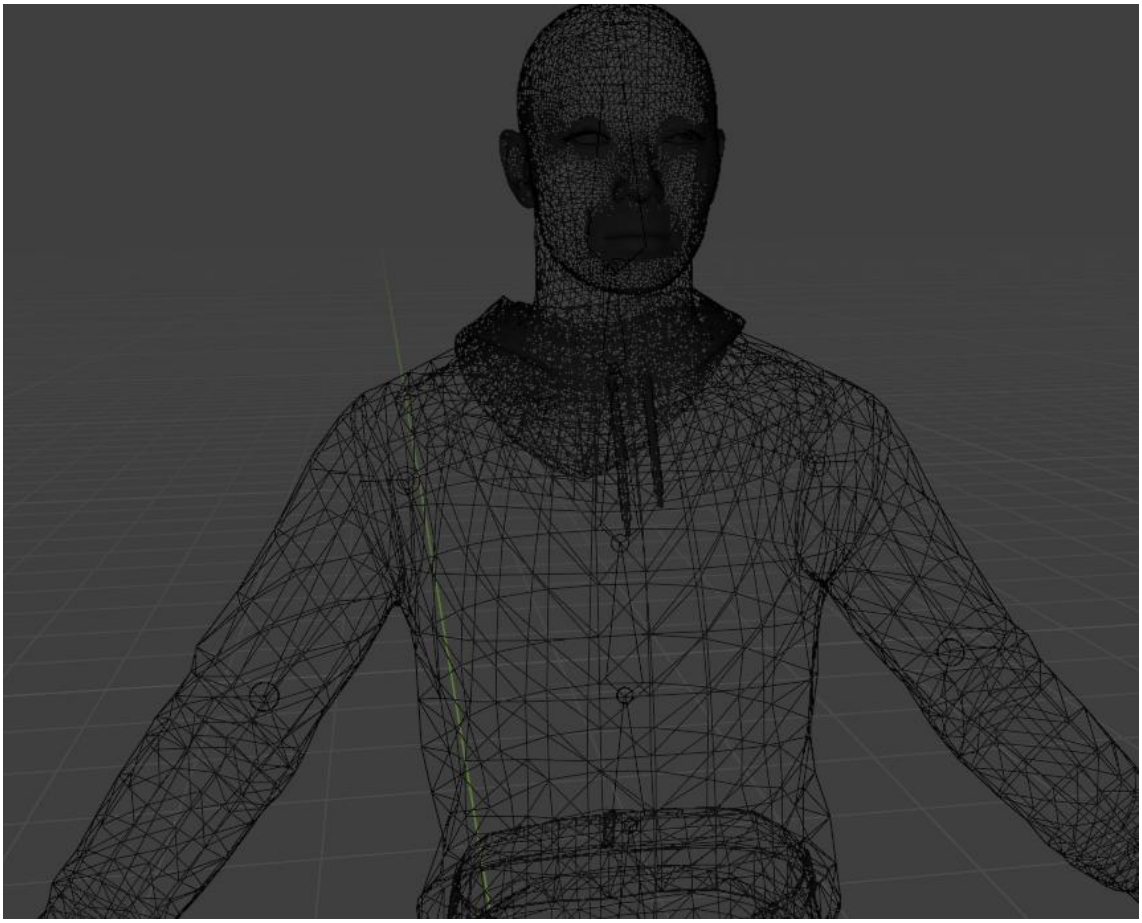


Рисунок 2.3 — LOD середнього ступеня деталізації

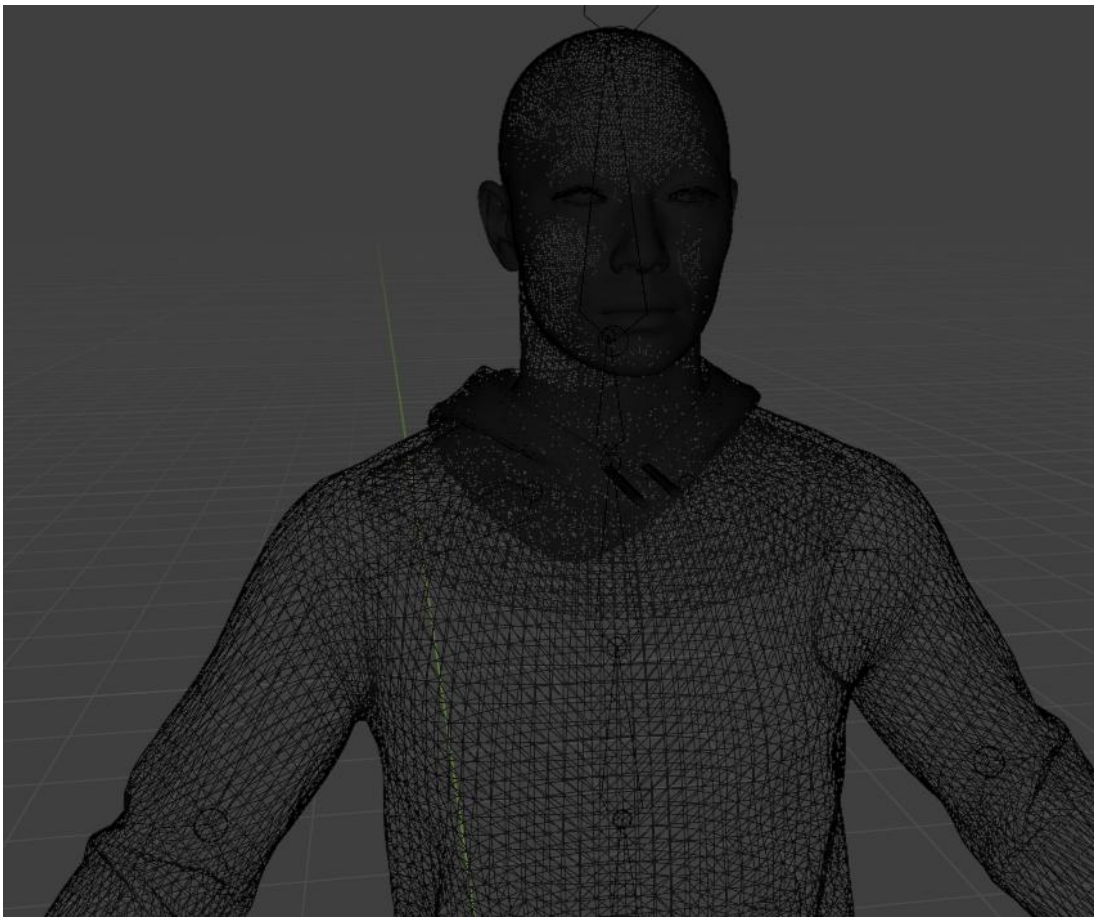


Рисунок 2.4 — LOD найвищого ступеня деталізації

2.2 Технології для розробки аддону

2.2.1 Використання Python та Blender API

У розробці аддону для підготовки моделей MetaHuman у Blender використовується мова програмування Python, яка є основною мовою сценаріїв у Blender. Також завдяки підтримці Blender API, Python дозволяє безпосередньо взаємодіяти з внутрішніми структурами 3D-середовища, що, у свою чергу, можна спостерігати в автоматизації рутинних процесів та створенні зручних інструментів.

Blender API (Application Programming Interface) надає доступ до таких ключових можливостей:

- керування об’єктами сцени: Python дозволяє знаходити, створювати, об’єднувати або видаляти будь-які об’єкти: меші, арматури (риг), групи

вершин, колекції. Наприклад, під час очищення сцени від непотрібних рівнів деталізації (LOD), виконується автоматичне видалення відповідних об’єктів за заданими умовами.;

- зміна трансформацій: скрипти можуть змінювати положення, масштаб і обертання об’єктів, що критично важливо при автоматичному розміщенні нового Mixamo-риг відносно центру моделі MetaHuman. Також підтримується застосування масштабів, обнулення трансформацій і їх застосування;
- редагування геометрії об’єктів: Blender API дає змогу додавати або змінювати кістки у режимі редагування скелета, генерувати скелет з нуля, задавати координати head, tail для кожної кістки та встановлювати батьківські зв’язки. У створеному аддоні саме так відбувається генерація структури Mixamo;
- побудова користувацьких панелей та елементів інтерфейсу: за допомогою модулів, створюється інтерфейс аддона, розміщений у бічній панелі 3D-вікна. У ньому можуть бути реалізовані досить багато елементів. Кожен елемент зв'язаний з функціональністю аддона через оператори.

В випадку створення визначеного адону будуть використані майже всі пункти, які вказані вище. А саме створення та видалення об’єктів, трансформації, розробка панелей інтерфейсу для користування. Також повинні оброблятися події користувача в інтерфейсі аддону, дозволяючи реагувати на вибрані опції, виконувати дії в реальному часі та оновлювати дані сцени без необхідності втручання вручну.

Перевагами використання Python у цьому контексті є:

- кросплатформеність (аддон працює однаково на Windows, macOS, Linux);
- гнучкість та можливість швидкої модифікації логіки;
- активна спільнота розробників Blender, що полегшує підтримку та вдосконалення рішень.

2.2.2 Створення інтерфейсу для користувача

Зручний та інтуїтивно зрозумілий інтерфейс користувача є важливою складовою будь-якого аддону, особливо тоді, коли мова йде про складні задачі підготовки 3D-моделей. У аддоні реалізовано просту та логічну структуру інтерфейсу за допомогою засобів Blender API, що дозволяє користувачеві здійснювати ключові дії без потреби в глибоких технічних знаннях.

Інтерфейс аддону включає наступні основні елементи:

- кнопки запуску основних операцій, таких як заміна скелету або застосування змін;
- галочки для активації або деактивації додаткових опцій;
- випадаючі списки або селектори для швидкого вибору бажаного рівня деталізації або частин моделі;
- окремі вкладки або панелі редагування, що дозволяють вручну скорегувати положення або розмірність скелету після автоматичної обробки;

Приклад вигляду кнопок та інших функцій панелі аддону можна переглянути на рисунку 2.5.

Рисунок 2.5 – Панель аддону

Окрема увага приділяється простоті та швидкості доступу до ключових функцій. Весь інтерфейс розташовується у вкладці N-панелі в окремому підменю, що робить його завжди доступним під час роботи з моделлю у 3D-вікні.

2.2.3 Використання стандартів 3D-файлів

При розробці аддону для підготовки моделей MetaHuman у Blender варто звернути увагу на забезпечення сумісності з поширеними форматами 3D-файлів. Це необхідно для зручного обміну моделями між різними програмними середовищами, такими як Unreal Engine, Unity, Maya, Cinema 4D або інші редактори 3D-графіки.

Формати, з якими працює Unreal Engine та Blender:

1. FBX (Filmbox) — основний формат для передачі 3D-моделей разом зі скелетами, анімаціями, матеріалами та іншими метаданими. Blender має вбудовану підтримку імпорту та експорту у формат FBX, хоча при цьому можуть виникати деякі несумісності — наприклад, часткова втрата контролерів ригу або специфічні іменування кісток. Саме тому при роботі з MetaHuman важливо автоматично адаптувати структуру імпортованої моделі, що й реалізується в межах створеного аддона;
2. OBJ (Wavefront) — формат, який не зберігає анімацій, кісток, або матеріальних властивостей у повному обсязі. Проте він часто використовується в простих обмінних операціях, для збереження "статичних" моделей або тестування геометрії. Blender читає цей формат нативно і дозволяє легко виконувати попередню обробку моделі;
3. GLTF/GLB — сучасні формати для швидкого рендерингу в реальному часі, які також підтримують матеріали, текстури та анімації.

Найкращим форматом буде, звісно, FBX, тому що при імпорті він зберігає всю необхідну інформацію для подальшої роботи. Також, користувач зможе далі експортувати модель з своїми налаштуваннями до зовнішніх систем.

2.3 Рішення оптимізації справності аддону при розробці

2.3.1 Сумісності з різними версіями Blender

Під час розробки аддону для Blender важливою задачею є забезпечення стабільної роботи з різними версіями цього програмного забезпечення. Blender активно розвивається, і після кожного оновлення можуть змінюватися внутрішні механізми API, структура даних, принципи доступу до об'єктів сцени або навіть логіка імпорту/експорту.

Відповідно, щоб мінімізувати ці проблеми, у процесі розробки аддону враховуються наступні підходи:

1. використання перевірених та стабільних функцій API, які залишаються незмінними протягом декількох версій Blender;
2. передбачення потенційних невідповідностей здійснюється через перевірку версії Blender на етапі ініціалізації аддону. Цей підхід дозволяє своєчасно виявляти проблеми сумісності та попереджати користувача про можливі ускладнення;
3. регулярне тестування аддону на актуальних версіях Blender перед кожним релізом є найкращою практикою, яка мінімізує ризики несправностей.

Забезпечення сумісності — це постійний процес, що вимагає гнучкого підходу та уважності до змін у Blender. Але завдяки правильній організації коду та врахуванню специфіки API, можливо підтримувати стабільну роботу аддону у більшості актуальних середовищ.

2.3.2 Технічні питання при роботі з моделями MetaHuman

Основні технічні складнощі, що виникають:

1. Велика кількість об’єктів у сцені: типова MetaHuman-модель складається з десятків окремих елементів: основний тулуб, голова, рогівка, склера, одяг, волосся, брови, вії, внутрішня частина рота, зуби тощо. Кожен з цих елементів представлений як окремий меш і має власний матеріал. Це перевантажує показове вікно і робить сцену важкою в навігації, особливо для новачків;
2. Точність позиціонування: під час заміни скелету та переналаштування елементів сцени важливо зберігати точне позиціонування частин тіла, волосся, одягу тощо. Будь-яке зміщення може спричинити візуальні артефакти чи деформації;
3. Висока навантаженість сцени: через деталізацію моделі сцена може бути важкою для обробки, особливо на комп’ютерах з обмеженими ресурсами. Це створює потребу в оптимізації кількості полігонів або використанні LOD;
4. Ситуації з масштабуванням: іноді при імпорті з Unreal Engine або інших платформ масштаби моделі можуть відрізнятися, що ускладнює інтеграцію з новим скелетом або сценою в Blender.
5. Ієрархія скелета MetaHuman є закритою для модифікацій: оригінальний риг MetaHuman побудований під Control Rig в Unreal Engine, тому не придатний для прямої анімації в Blender. Більшість кісток мають імена, незрозумілі Blender-користувачам, і відсутня класична схема “Hips → Spine → Chest → Head”.

Приклад того, скільки виникає об’єктів при імпорті наведен на рисунку 2.6.

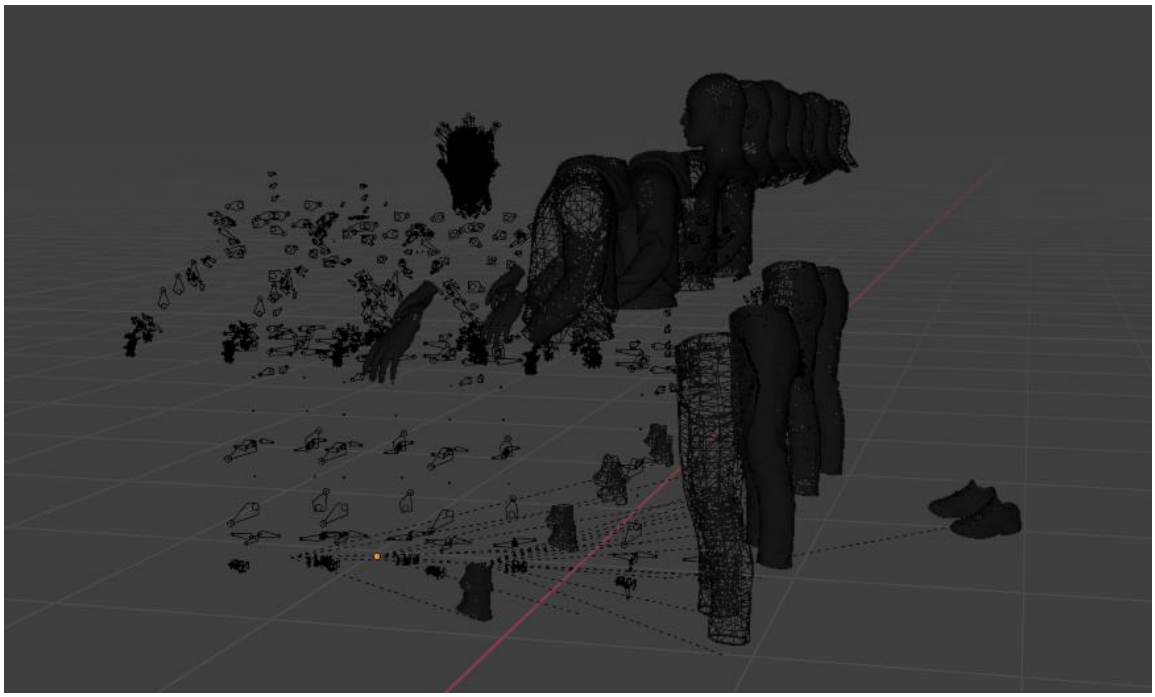


Рисунок 2.6 – Об’єкти MetaHuman

Для вирішення цих завдань аддон впроваджує низку технічних рішень: автоматично знаходить і видаляє старий скелет, оптимізує розташування об'єктів при додаванні нового, дозволяє коригувати розміри та положення скелета для точного узгодження з геометрією, а також забезпечує вибір рівня деталізації (LOD) для зниження системного навантаження.

2.4 Актуальність розробки

Тема створення інструментів для підготовки моделей MetaHuman у Blender набуває великої актуальності з кількох причин. Платформа MetaHuman від Epic Games дозволяє розробляти високореалістичні 3D-моделі людей, які знайшли широке застосування в сферах ігрової індустрії, віртуальної реальності, кіновиробництва та анімації. Проте, через складну структуру моделей і специфічну будову їхнього скелету, їх використання в інших програмах, зокрема Blender, стає доволі непростим завданням.

Наразі майже відсутні доступні інструменти чи аддони, які б автоматизували процес підготовки MetaHuman-моделей для роботи у Blender. Користувачам доводиться виконувати низку трудомістких ручних операцій: видалення оригінального скелету, налаштування геометрії, адаптація до сторонніх скелетів (наприклад, Mixamo), модифікація рівнів деталізації (LOD) тощо. Така ситуація значно ускладнює робочий процес навіть для досвідчених 3D-художників.

Хоча на ринку існує чимало інструментів для ретаргетингу анімацій або скелетів із Mixamo та інших платформ. Навіть в самому Unreal Engine є зручна система ретаргету, якраз у тій звязці систем скелетів, які беруться за основу у цьому аддоні. Зворотний процес — адаптація MetaHuman-моделей для сумісності з цими скелетами або платформами — практично не представлений. Це створює очевидну потребу в інструменті, який дозволив би швидко та просто підготувати MetaHuman для роботи у Blender із більш універсальними скелетами не виходячи з нього.

Розробка аддону, який автоматизує цей процес і забезпечує зручний інтерфейс, може значно спростити й прискорити взаємодію з моделями MetaHuman. Такий інструмент відкриє нові перспективи як для професіоналів, так і для початківців, забезпечуючи легшу інтеграцію сучасних 3D-моделей у проєкти без необхідності виконувати складні технічні операції.

2.5 Перспективи розвитку аддону

2.5.1 Розширення функціоналу

На даному етапі аддон вже виконує основні функції підготовки моделей MetaHuman у Blender — зокрема, заміну скелету, позиціонування та об’єднання мешів. Однак потенціал для подальшого розвитку цього інструменту залишається значним. Розширення функціоналу дозволить зробити процес підготовки моделей ще більш гнучким, швидким і зручним для користувачів із різним рівнем досвіду.

Можливі напрями розвитку:

1. додаткові режими редагування скелету: включення функцій точного налаштування окремих кісток безпосередньо через інтерфейс. Це дозволить користувачеві самостійно регулювати структуру під специфічні потреби проєкту;
2. збереження профілів налаштувань: реалізація системи шаблонів, яка дозволяє зберігати і завантажувати власні конфігурації для різних моделей або типів проєктів;
3. підтримка інших форматів скелетів: у майбутньому аддон може отримати підтримку не лише Mixamo, а й інших популярних скелетних систем;
4. більше опцій керування об'єднанням мешів: можливість точково обирати, які частини моделі об’єднувати зі скелетом, а які залишити окремими
5. перенесення анімацій з скелету MetaHuman на скелет Mixamo: розробити систему ретаргету саме для Blender, причому в обидві сторони.

2.5.2 Розширене управління текстурами та матеріалами

У процесі підготовки моделей MetaHuman до подальшого використання в середовищі Blender важливу роль відіграє не лише коректна геометрія та структура скелету, але й правильне застосування текстур і матеріалів. Якісна візуалізація моделі безпосередньо залежить від точності відображення шкіри, волосся, очей, одягу та інших компонентів. Тому перспективним напрямом розвитку аддону є автоматизація роботи з текстурними ресурсами та матеріалами.

Розширене управління текстурами передбачає впровадження механізмів автоматичного призначення матеріалів після завершення основних етапів обробки моделі. Аддон може бути доповнений функціями, які самостійно визначають і застосовують відповідні текстури до конкретних частин моделі. Це дозволить уникнути ручного налаштування та зекономить час користувача, особливо у випадках, коли обробляються кілька моделей одночасно.

Ще одним важливим напрямом є підтримка сучасних шейдерів, що широко використовується у Blender. Автоматичне створення матеріалів забезпечить більш реалістичне відображення моделей та дозволить швидко інтегрувати їх у фінальні сцени або інші програми.

2.5.3 Власна ретаргет система для переносу анімацій

На момент розробки доповнення Metahuman до Mixamo було виконано базову технічну підготовку моделей для подальшого використання в Blender, включаючи спрощення сітки, приєднання Mixamo-скелета, позиціонування та оптимізацію геометрії. Але водночас зі зростанням вимог користувачів до використання MetaHuman у повномасштабних анімаційних або ігрових проектах, наступним логічним кроком у розробці доповнення є впровадження системи ретаргетингу анімації.

Впровадження ретаргету це доволі складний процес, який вимагає врахування структурних відмінностей між скелетами MetaHuman та Mixamo. Для реалізації такої системи в межах аддона необхідно забезпечити декілька ключових етапів, кожен з яких відповідає за певний аспект переносу анімацій:

1. Першим етапом є автоматичне співставлення кісток. Оскільки структури скелетів у MetaHuman і Mixamo суттєво різняться, необхідно створити механізм, який автоматично визначає відповідність між іменами кісток обох систем. Наприклад, кістка mixamorig:Hips у Mixamo має відповідати root або pelvis у MetaHuman. Для цього в аддоні реалізовується словник відповідності, який може бути відредагований вручну через інтерфейс, якщо автоматичне визначення не спрацює коректно.
2. Другим етапом є масштабування та нормалізація позицій анімації. Через те, що моделі можуть відрізнятися за розмірами, довжиною кінцівок та загальним положенням у просторі, важливо адаптувати трансформації анімації так, щоб вони відповідали новій моделі. Цей етап передбачає

адаптацію координат анімаційних ключів, а також можливу корекцію “root motion” — тобто загального руху моделі по сцені.

3. Наступним кроком є перенос ключів анімації. Анімації Mixamo представляють собою набори ключових кадрів, які необхідно перенести на відповідні кістки MetaHuman. Після того як відповідність кісток встановлено, аддон зчитує ключі обертання, положення й масштабу з Mixamo-кісток і записує їх на відповідні кістки MetaHuman. При цьому важливо зберігати плавність рухів, уникати артефактів та враховувати відмінності у базових позах.
4. Для складніших випадків передбачається підтримка інверсної кінематики (IK). Деякі анімації, особливо з високим рівнем деталізації рухів, використовують контрольні цілі для кінцівок (наприклад, стопи або долоні). У таких ситуаціях важливо передбачити можливість коректного перенесення IK-ланцюгів або їх автоматичної конверсії у Forward Kinematics (FK), щоб забезпечити сумісність із MetaHuman.
5. Завершальним етапом є візуальний попередній перегляд результату ретаргету. Після перенесення анімації користувач повинен мати змогу переглянути результат прямо у вікні Blender до моменту його збереження. Аддон передбачає відображення застосованої анімації в режимі Viewport Playback, а також надає можливість остаточно “застосувати” або “відмінити” перенесення через інтерфейсну кнопку. Це дозволяє уникати помилок і дає змогу швидко тестувати результат.

Для розуміння, яке повинно бути співвідношення кісток, нижче наведена таблиця 2.1.

Таблиця 2.1 – Загальна структура проєкту

Mixamo кість	MetaHuman кість
Hips	root
Spine	spine_01
Neck	neck_01
LeftHand	hand_l
RightFoot	calf_r → foot_r

Впровадження системи ретаргету відкриє шлях до повноцінного використання MetaHuman у будь-яких анімаційних або ігрових проектах прямо з Blender, без потреби в зовнішньому ПЗ чи складних ручних конверсіях.

2.5.4 Автоматичне збереження нових анімацій після ретаргету в окрему бібліотеку

Після завершення процесу ретаргету є успішне перенесення анімації з MetaHuman на структуру Mixamo, важливо не тільки переглядати результат у вікні перегляду, але й зберігати створені анімаційні дані для подальшого використання. У цьому контексті аддон реалізує функцію автоматичного збереження нових анімаційних кліпів (Action) у спеціальній внутрішній бібліотеці Blender або окремій директорії проекту.

Це збереження реалізується так, щоб кожна нова Action мала унікальну назву, що включає ім’я джерельного файлу або короткий опис типу руху (наприклад, Walk_MixamoRetarget, Jump_MixamoRetarget). Завдяки цьому користувач може легко знаходити потрібні анімації серед десятків або сотень кліпів, що особливо актуально при роботі над великими сценами.

Аддон автоматично позначає ці Action як “Fake User”, що гарантує їх збереження в .blend-файлі навіть після перезапуску програми. Крім того,

користувач може обрати опцію — зберігати Action у вигляді зовнішніх FBX файлів, що дозволяє створити окрему бібліотеку анімацій для подальшого використання в інших проєктах або експорті в Unreal Engine.

Для зручності управління бібліотекою анімацій в інтерфейсі аддона передбачено список усіх збережених Action із можливістю попереднього перегляду, перейменування та видалення. Також є можливість групувати анімації за тегами — наприклад: "рух", "бойові", "жести", "емоції".

Приклад такої бібліотеки наведен в таблиці 2.2.

Таблиця 2.2 – Структури бібліотеки анімацій

Назва анімації	Джерело	Дата створення	Тип руху	Теги
Walk_MixamoRetarget	Walk.fbx	2025-05-07	Рух	walk, basic
Jump_MixamoRetarget	Jump.fbx	2025-05-04	Рух	jump, action
Greet_MixamoRetarget	Greet.fbx	2025-04-07	Жест	greet, hand

2.5.5 Можливість створення кастомного ригу, оптимізованого для анімацій

Ще одним перспективним напрямом розвитку аддону Metahuman to Mixamo є впровадження функції створення власного анімаційного ригу без обмеження лише на структуру Mixamo. Це відкриє ширші можливості для користувачів, які потребують більш гнучкого та оптимізованого контролю над анімацією в Blender. Такий риг повинен поєднувати в собі зручність системи Rigify, але бути адаптованим до специфіки моделей MetaHuman, зокрема до їх складної геометрії та великої кількості окремих об’єктів (волосся, очі, одяг, брови тощо).

Передбачається, що в майбутньому аддон зможе автоматично створювати кастомний риг на основі шаблону, обраного користувачем. Такий риг включатиме інверсну кінематику (IK) для рук і ніг, можливість перемикання між IK та FK режимами, повноцінну систему контролерів для позиціювання тіла, а також за

потреби — базову підтримку лицевої анімації. Це значно полегшить процес створення ключових поз, циклів ходьби, взаємодії з предметами тощо, дозволяючи зосередитись безпосередньо на творчій частині, а не на технічному налаштуванні.

Окрім цього, користувач зможе обрати тип ригу залежно від завдання:

- спрощений риг для швидких попередніх візуалізацій;
- повнофункціональний риг для продакшн-анімації або сумісний із Rigify.

Такі ригі можна буде легко налаштовувати далі з використанням інструментів самого Blender. Важливим аспектом стане і те, що структура кісток буде адаптована для експорту в ігрові рушії, такі як Unity або Unreal Engine, що дозволить використовувати створені анімації в інтерактивних проєктах. Приклад такого ригу наведен на рисунку 2.7.

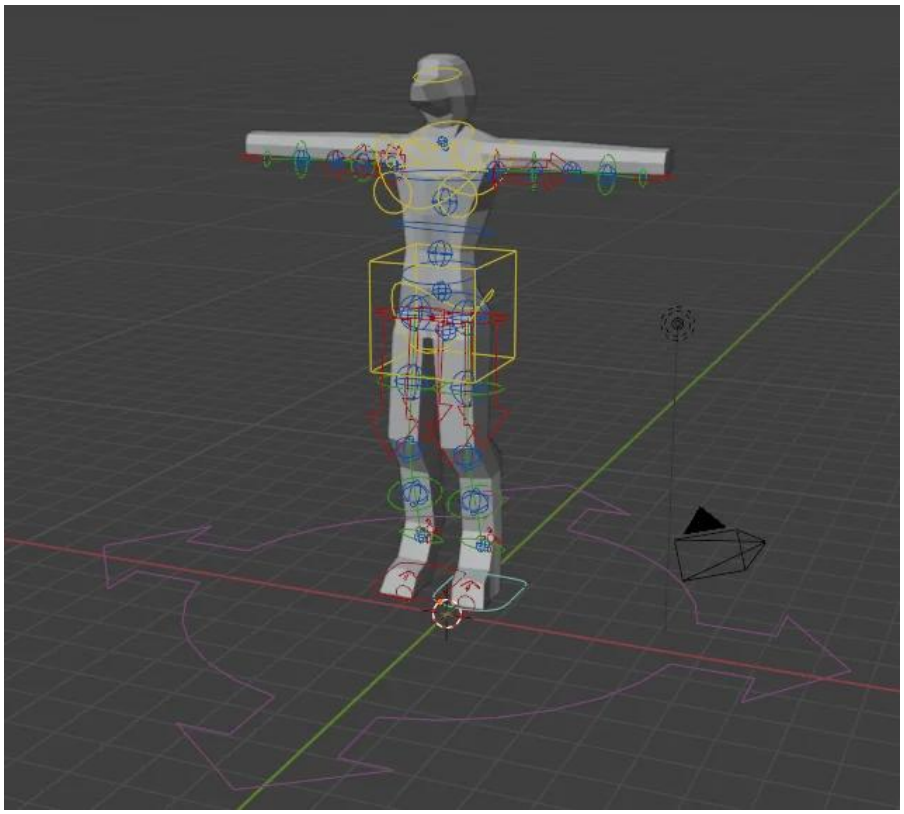


Рисунок 2.7 – Риг від Rigify

З технічної точки зору, така система має создавати оптимізовану ієрархію кісток, що не перевантажує сцену. Також дозволяти користувачеві через інтерфейс аддону змінювати розміри, положення та прив’язку контролерів до геометрії

персонажа. Таким чином, кастомний риг стане не лише альтернативою Mixamo, а й повноцінною платформою для створення високоякісних анімацій прямо у Blender.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Опис функціоналу аддона

Розроблений аддон для Blender реалізує низку автоматизованих дій, що істотно спрощують підготовку моделей MetaHuman до подальшого використання — зокрема, для анімації за допомогою Mixamo, інтеграції у сторонні рушії або експорту в оптимізованому вигляді. Основна мета аддона полягає у зменшенні кількості ручної праці при роботі з технічно складними моделями та уніфікації структури персонажа до загальноприйнятих форматів.

Нижче наведено детальний опис усіх функцій, реалізованих у межах поточної версії аддона:

1. Підключення аддона до Blender: аддон створено у вигляді .py-файлу з відповідною службовою структурою, що дозволяє легко інсталювати аддон через налаштування, відображати окрему вкладку та керувати видимістю і доступом до всіх інструментів аддона з однієї панелі;
2. автоматичне виявлення скелета MetaHuman: після імпорту моделі у сцену автоматично визначається скелет типу “Armature”, навіть якщо він має нестандартне або довільне ім’я. Це досягається за допомогою сканування об’єктів у сцені за ключовими ознаками;
3. видалення вихідного скелета MetaHuman та непотрібних LOD: аддон повністю видаляє вихідний риг, зберігаючи при цьому геометрію, що була до нього прив’язана;
4. вибір рівня деталізації (LOD): через інтерфейс користувач може обрати потрібний рівень, після чого всі інші автоматично видаляються зі сцени;
5. створення нового скелета Mixamo: після очищення сцени автоматично генерується скелет, що дозволяє зберігати коректне розташування суглобів та уникати деформацій і перекосів моделі;

6. оптимізація позиціонування моделі та скелету: скелет позиціонується автоматично за допомогою вичислення центру об’єкта, загальній висоті моделі та за коефіцієнтом, заданим через інтерфейс ;
7. об'єднання мешів за бажанням користувача: через галочку “Merge Mesh” користувач може злити всі окремі об’єкти моделі в один;
8. вікна точного редагування скелета після роботи аддону: надання можливості користувачу відкрити вікна точного редагування параметрів скелету.

3.2 Структура коду та опис інтерфейсу користувача

Розроблений аддон складається з декількох логічних частин, кожна з яких відповідає за певний етап підготовки моделі MetaHuman. Код організовано у вигляді блоків, що дозволяє підтримувати чисту архітектуру, легкість у розширенні та повторному використанні функцій. У таблиці 3.1 вказана загальна структура проєкту.

Таблиця 3.1 – Загальна структура проєкту

Файл/Блок	Призначення
__init__.py	Головний файл аддона, реєстрація операторів та панелей
mh_detect	Виявлення скелета MetaHuman у сцені
mh_cleanup	Видалення скелета MetaHuman та обробка мешів
mh_lod	Робота з LOD
mixamo_rig	Створення скелету Mixamo, позиціонування, масштабування
ui_panels	Створення панелей в інтерфейсі Blender
utils	Допоміжні функції

Щоб аддон функціонував у Blender, його необхідно правильно оформити та зареєструвати. Це передбачає створення спеціального файл __init__.py, який містить службову інформацію про аддон, а також перелік модулів, які потрібно

підключити. Крім того, користувач повинен встановити аддон через стандартний інтерфейс Blender. Файл __init__.py є центральним для будь-якого Blender-аддона. У ньому вказується службова інформація, а також виконується реєстрація всіх операторів, панелей та властивостей, які використовує аддон. На рисунку 3.1 та рисунку 3.2 можна побачити основну частину коду для відображення та як виглядає вікно підключення аддону у налаштуваннях.

```
import bpy
from mathutils import Vector, Matrix

bl_info = {
    "name": "Metahuman to Mixamo",
    "author": "Your Name",
    "version": (1, 4),
    "blender": (3, 0, 0),
    "location": "3D View > Sidebar > Metahuman",
    "description": "Prepares Metahuman model for Mixamo with real-time rig adjustment",
    "category": "Rigging",
}
```

Рисунок 3.1 – Основна частина коду

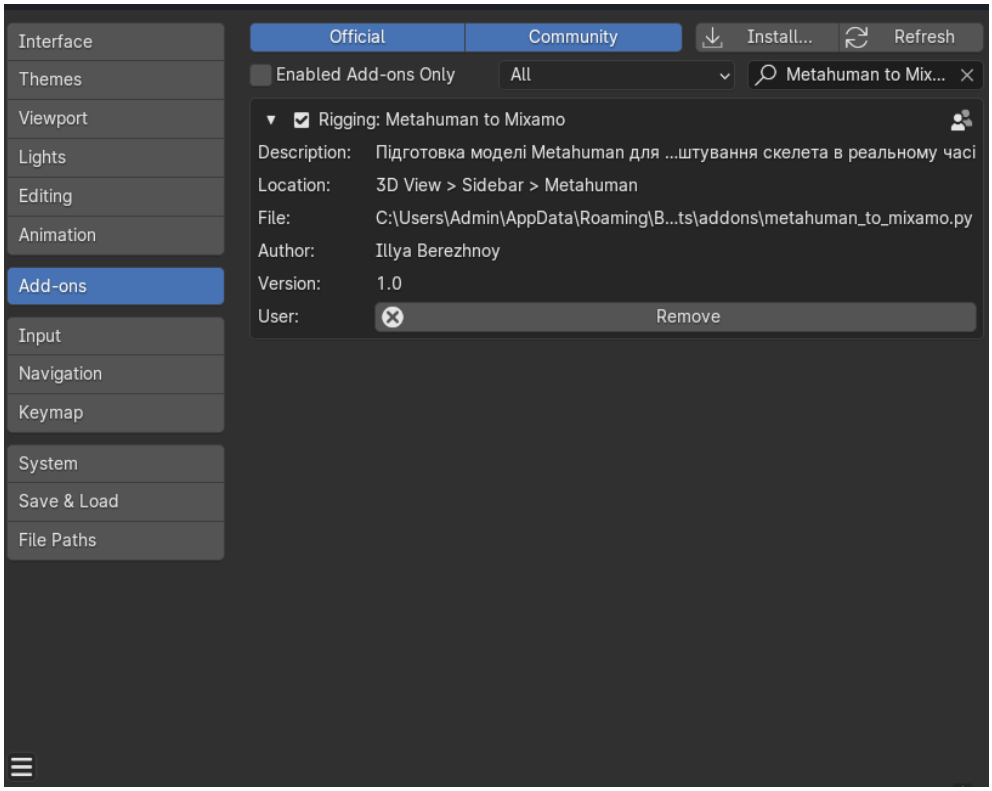


Рисунок 3.2 – Вікно підключення аддону

Після встановлення аддона, у правій панелі 3D View з'явиться відповідна секція Metahuman. Через неї можна буде взаємодіяти та вже починати використовувати. Вона зображена на рисунку 3.3.

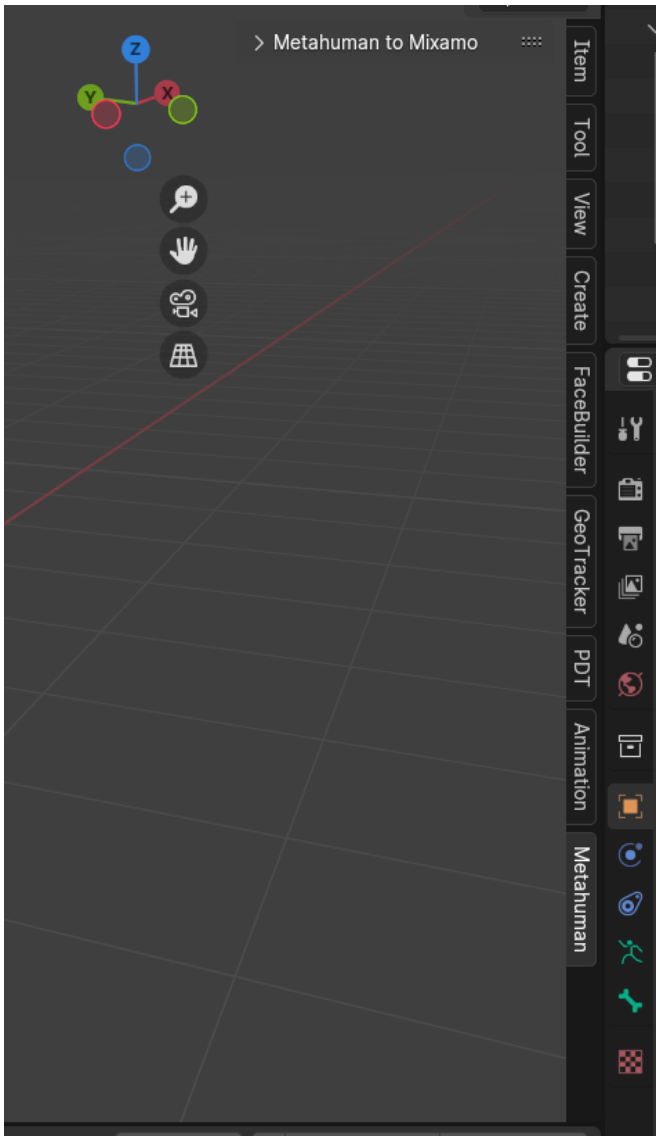


Рисунок 3.3 – Секція Metahuman

Блок mh_detect відповідає за автоматичний пошук об'єкта типу "Armature", який має характерну структуру MetaHuman. Ця логіка дозволяє автоматично ідентифікувати потрібний скелет без участі користувача. Основна функція зображена на рисунку 3.4.

```
def find_metahuman_armature():
    for obj in bpy.data.objects:
        if obj.type == 'ARMATURE' and "metahuman" in obj.name.lower():
            return obj
    return None
```

Рисунок 3.4 – Блок mh_detect

Блок mh_cleanup очищає сцену від старого скелета MetaHuman і зберігає пов’язані меші. Також може об’єднувати меші залежно від налаштування користувача. Основна функція зображена на рисунку 3.5.

```
def cleanup_metahuman_model(self, lod_level):
    """Видалення непотрібних об'єктів"""
    objects_to_remove = [
        obj for obj in bpy.data.objects
        if obj.type in {'MESH', 'EMPTY', 'ARMATURE'}
        and 'LOD' in obj.name
        and not obj.name.endswith(lod_level)
    ]
    for obj in objects_to_remove:
        bpy.data.objects.remove(obj, do_unlink=True)
```

Рисунок 3.5 – Блок mh_cleanup

Блок mixamo_rig генерує скелет за шаблоном Mixamo, підлаштовує його розміри та положення під наявну модель. Основна функція створення та вигляд скелету у текстовому форматі зображені на рисунку 3.6. Для самого написання скелету потрібно визначити правильні назви кісток, їх ієрархію та координати їх положення. Потрібно слідкувати за правильною послідовністю для коректного створення.

```
def create_bones(self, armature_obj):
    """Створення кісток скелета"""
    bpy.context.view_layer.objects.active = armature_obj
    bpy.ops.object.mode_set(mode='EDIT')

    # Дані для створення кісток Mixamo
    bones_data = [
        # Коренева кістка і хребет
        ("mixamorig:Hips", None,
 (0.004466414451599121, 2.385824203491211, 0.06472378224134445),
 (0.004466414451599121, 2.6421542167663574, 0.06472378224134445)),

("mixamorig:Spine", "mixamorig:Hips",
 (0.004466414451599121, 2.629953384399414, 0.04066042602062225),
 (0.004466414451599121, 2.9147708415985107, 0.012586494907736778)),

("mixamorig:Spine1", "mixamorig:Spine",
 (0.004466413985937834, 2.9147706031799316, 0.012586522847414017),
 (0.004466413985937834, 3.2402760982513428, -0.019497960805892944)),

("mixamorig:Spine2", "mixamorig:Spine1",
 (0.004466413985937834, 3.2402760982513428, -0.019497960805892944),
 (0.004466413985937834, 3.5810635089874268, -0.05308876931667328)),

("mixamorig:Neck", "mixamorig:Spine2",
 (0.004466413985937834, 3.6064693927764893, -0.05559299886226654),
 (0.004466413985937834, 3.8145534992218018, -0.05559299886226654)),

("mixamorig:Head", "mixamorig:Neck",
 (0.004466414451599121, 3.8085379600524902, -0.005921110510826111),
 (0.004466414451599121, 4.317898750305176, -0.005921110510826111)),
```

Рисунок 3.6 – Блок mixamo_rig

Блок mh_lod. робить всю роботу, яка пов'язана з видаленням лишніх LOD. Це потрібно, щоб користувачу не доводилось робити лишніх кроків після роботи аддону. Звісно з цим використовуються інші функції, такі як:

- об'єднання мешів для обраного LOD;
- застосування масштабу для LOD;

Все ці частини коду можна побачити на рисунках 3.7-3.9.

```
def remove_other_lods(self, lod_level):
    """Видалення інших LOD рівнів"""
    objects_to_remove = [
        obj for obj in bpy.data.objects
        if obj.type == 'MESH'
        and 'LOD' in obj.name
        and not obj.name.endswith(lod_level)
    ]
    for obj in objects_to_remove:
        bpy.data.objects.remove(obj, do_unlink=True)
```

Рисунок 3.7 – Блок mh_lod

```
def merge_lod(self, lod_level):
    """Об'єднання мешів для обраного LOD"""
    global last_mesh
    lod_meshes = [obj for obj in bpy.data.objects if obj.type == 'MESH' and lod_level in obj.name]
    if len(lod_meshes) > 1:
        bpy.ops.object.select_all(action='DESELECT')
        for obj in lod_meshes:
            obj.select_set(True)
        bpy.context.view_layer.objects.active = lod_meshes[0]
        bpy.ops.object.join()
        bpy.context.object.name = f"Metahuman_{lod_level}"
        last_mesh = bpy.context.object
```

Рисунок 3.8 – Об'єднання мешів для обраного LOD

```
def adjust_lod_scale(self, lod_level):
    """Застосування масштабу для LOD"""
    lod_mesh = next((obj for obj in bpy.data.objects if obj.type == 'MESH' and lod_level in obj.name), None)
    if lod_mesh:
        lod_mesh.scale = (0.025, 0.025, 0.025)
        bpy.context.view_layer.objects.active = lod_mesh
        bpy.ops.object.transform_apply(scale=True)
```

Рисунок 3.9 – Застосування масштабу для LOD

Блок ui_panels. відповідає за виведення інтерфейсу користувача у вкладці Blender. Він виводить усі кнопки та описання до них. На рисунках 3.10-3.12 зображені головні частини коду, які за це відповідають та самі приклади кнопок та інших засобів управління аддоном на основній панелі.

```
class VIEW3D_PT_CleanupPanel(bpy.types.Panel):
    """Панель інструментів у бічній панелі"""
    bl_label = "Metahuman to Mixamo"
    bl_idname = "VIEW3D_PT_cleanup_panel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "Metahuman"

    def draw(self, context):
        """Відображення елементів інтерфейсу"""
        layout = self.layout

        col = layout.column()
        col.prop(context.scene, "metahuman_lod_level")
        col.prop(context.scene, "merge_mesh", text="Merge mesh")
        col.operator("object.cleanup_metahuman")

        if last_armature:
            box = layout.box()
            box.label(text="Skeleton setup (real time)")

            row = box.row()
            row.prop(context.scene, "rig_scale_correction", text="Scale")
            row.prop(context.scene, "auto_scale_factor", text="Autoscale")

            box.prop(context.scene, "rig_y_offset_percent", text="Offset Y (%)")
            box.prop(context.scene, "rig_z_offset_percent", text="Offset Z (%)")

def register():
    """Реєстрація класів і властивостей"""
    bpy.utils.register_class(OBJECT_OT_CleanupMetahuman)
    bpy.utils.register_class(VIEW3D_PT_CleanupPanel)

    # Властивості сцени
    bpy.types.Scene.metahuman_lod_level = bpy.props.EnumProperty(
        name="Level LOD",
        items=[
            ('LOD0', "LOD0", "Highest Detail"),
            ('LOD1', "LOD1", "High Detail"),
            ('LOD2', "LOD2", "Medium Detail"),
            ('LOD3', "LOD3", "Low Detail"),
            ('LOD4', "LOD4", "Lowest Detail"),
        ],
        default='LOD2'
    )
```

Рисунок 3.10 – Основні кнопки блока ui_panels

```
bpy.types.Scene.merge_mesh = bpy.props.BoolProperty(
    name="Merge mesh",
    description="Merge mesh in one LOD",
    default=True
)

bpy.types.Scene.rig_scale_correction = bpy.props.FloatProperty(
    name="Scale correction",
    default=default_scale,
    min=0.5,
    max=2.0,
    precision=3,
    update=update_rig
)

bpy.types.Scene.auto_scale_factor = bpy.props.FloatProperty(
    name="Autoscale correction",
    default=0.21,
    min=0.1,
    max=3.0,
    precision=2,
    update=update_rig
)

bpy.types.Scene.rig_y_offset_percent = bpy.props.FloatProperty(
    name="Offset Y",
    default=default_y_offset,
    min=-50,
    max=50,
    precision=1,
    update=update_rig
)

bpy.types.Scene.rig_z_offset_percent = bpy.props.FloatProperty(
    name="Offset Z",
    default=-48.3,
    min=-50,
    max=50,
    precision=1,
    update=update_rig
)
```

Рисунок 3.11 – Допоміжні кнопки блока ui_panels

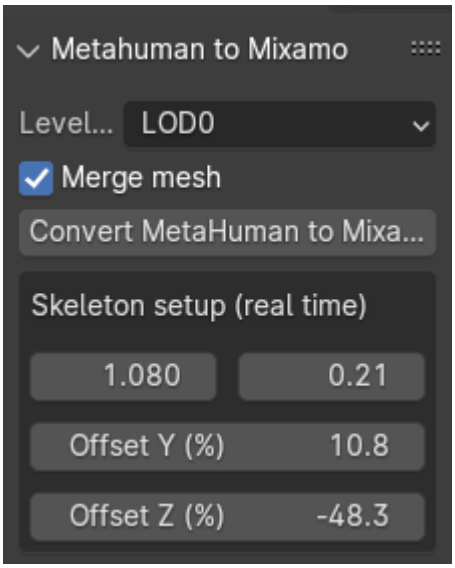


Рисунок 3.12 – Основна панель

Головна та єдина панель має назву MetaHuman to Mixamo та відображає всі керуючі елементи, які потрібні для базового використання. Також вході розробки були виставлені оптимальні налаштування, котрі йдуть відразу при роботі аддону для більшої зручності.

Блок utils містить функції, які не входять до основної логіки, але використовуються багатьма модулями. Ці функції не пов’язані безпосередньо з логікою створення або видалення скелетів, але виконують допоміжні задачі: математичні обчислення, обробку геометрії, знаходження меж моделі.

Основні функції:

- обчислення мінімального та максимального кута: використовується для розрахунку розміру моделі та її центру тяжіння;
- обчислення центру об’єкта;
- розрахунок масштабу скелета;
- виведення повідомлень у консоль.

Приклад можна побачити на рисунку 3.13.

```
if all_corners:
    mesh_min = Vector((min(v.x for v in all_corners), min(v.y for v in all_corners), min(v.z for v in all_corners)))
    mesh_max = Vector((max(v.x for v in all_corners), max(v.y for v in all_corners), max(v.z for v in all_corners)))
```

Рисунок 3.13 – Обчислення мінімального та максимального кута

3.3 Результати роботи аддона

Розроблений аддон для Blender автоматизує важливі етапи підготовки моделей MetaHuman до подальшої роботи в середовищі Blender або для експорту, зокрема в системи з анімаційною підтримкою Mixamo. У результаті його застосування користувач отримує сцену, в якій модель упорядкована, очищена від зайвих елементів і готова до використання з новим скелетом. На рисунку 3.14-3.15 буде наглядно показано, як спрацьовує аддон.

Результатом роботи є оптимізована 3D-модель, яка містить:

- тільки один активний LOD-меш;
- новий, чітко структурований скелет Mixamo, готовий до ретаргетингу або анімації;
- очищену сцену без зайвих об'єктів MetaHuman.

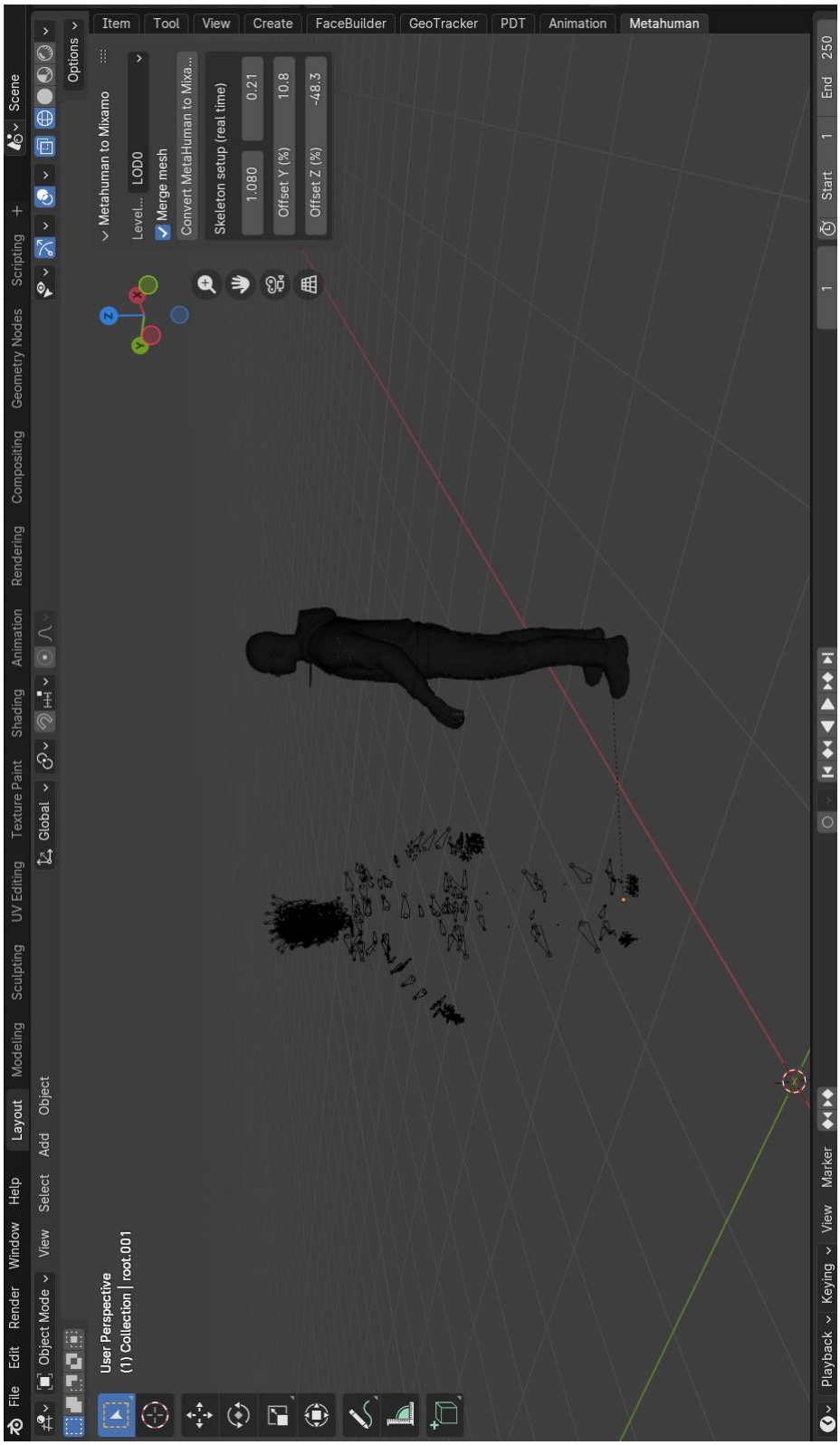


Рисунок 3.14 – Результат до роботи аддону

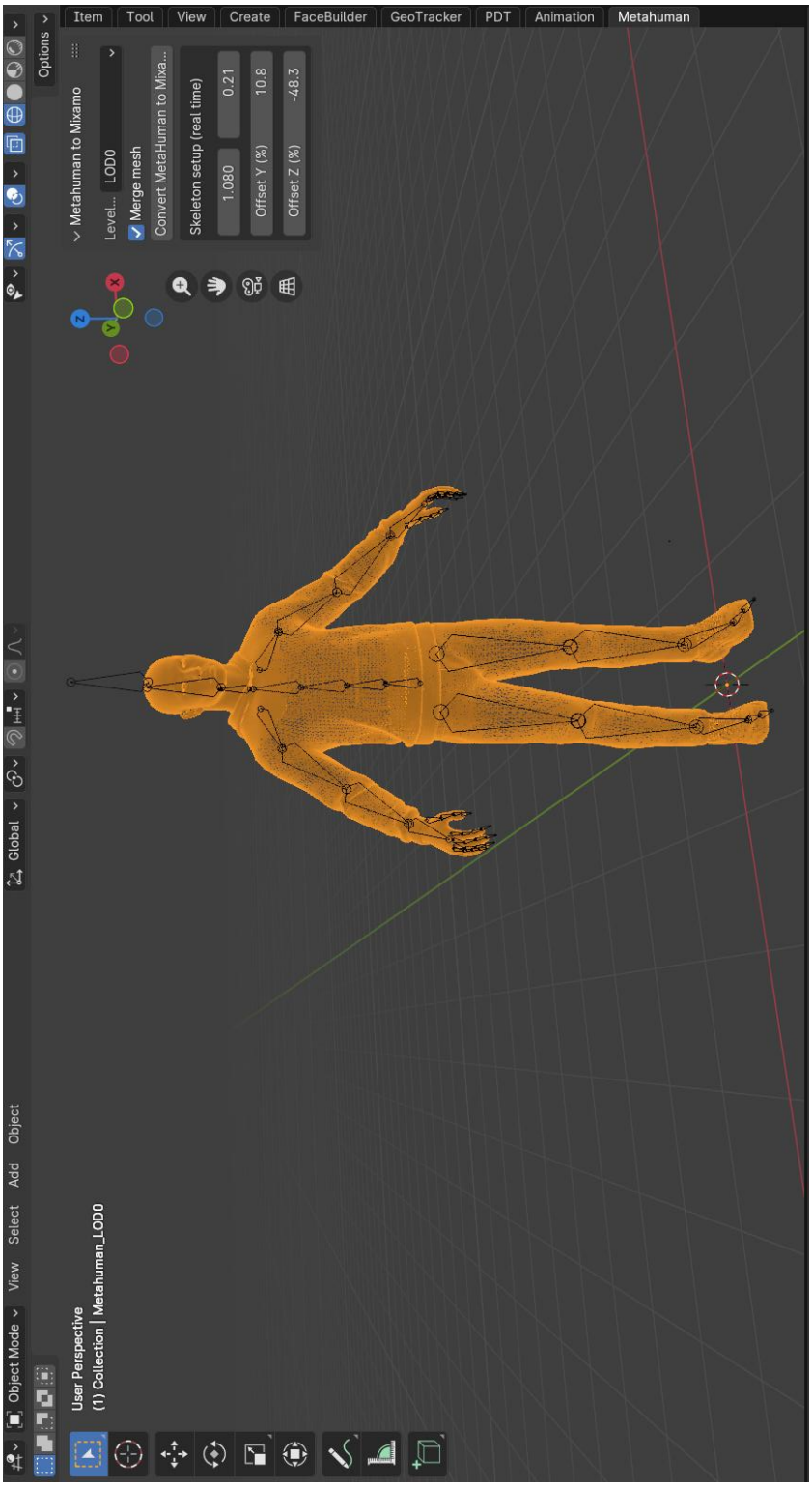


Рисунок 3.15 – Результат після роботи аддону

Особливістю є також можливість вибору об'єднання мешу чи залишити все різними частинами, як було при імпорті моделі. Це можна перевірити виділивши модель та подивитися, що обирається при взаємодії з моделлю. На рисунку 3.16 наведен приклад.

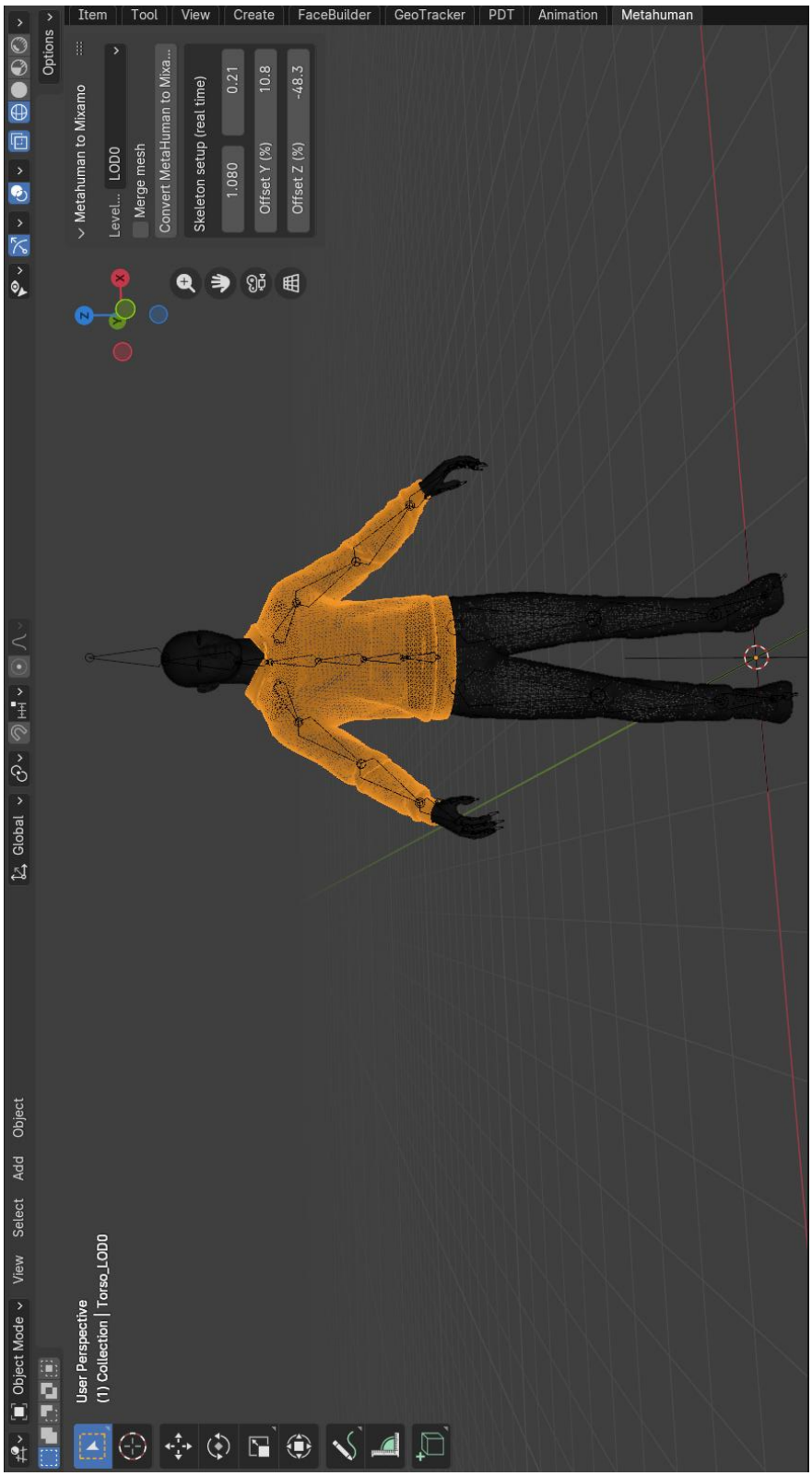


Рисунок 3.15 – Необ’єданний меш

ВИСНОВКИ

Метою цієї дипломної роботи було дослідження процесів підготовки 3D-моделей MetaHuman у Blender за допомогою спеціалізованого аддона для автоматизації роботи з цими моделями. Основна мета полягала в удосконаленні технології підготовки моделей для подальшого використання в анімаційних або ігрових проектах.

Для її реалізації було зроблено:

- аналіз існуючих рішень та технологій для роботи з 3D-моделями;
- дослідження функціональності аддона для Blender "Metahuman to Mixamo";
- розробка та оптимізація процесу інтеграції MetaHuman з Mixamo Rig для забезпечення зручного та ефективного використання моделей у Blender;
- створення функціоналу для редагування скелетних структур, об’єднання мешів, а також автоматизація позиціювання об'єктів.

У результаті реалізації проекту був розроблений аддон, що дозволяє автоматично видаляти старий скелет MetaHuman, створювати новий скелет Mixamo, а також зручний інтерфейс для коректного позиціонування та об’єднання мешів. Цей аддон значно спрощує роботу з моделями MetaHuman у Blender, дозволяючи зменшити час на підготовку моделей і підвищити точність їх використання для подальших анімацій чи ігрових сцен.

Практична цінність розробленого аддона полягає в наступному:

- підвищення ефективності підготовки 3D-моделей, завдяки автоматизації рутинних процесів;
- зменшення ймовірності помилок під час обробки моделей завдяки чітко налаштованому алгоритму роботи аддона;
- оптимізація робочого процесу для художників і розробників, які працюють з моделями MetaHuman в Blender.

Розроблений аддон дозволяє значно полегшити інтеграцію моделей MetaHuman в інші системи, спрощує процес налаштування анімацій і є важливим

інструментом для користувачів Blender, що працюють з високоякісними 3D-моделями. Таким чином, створення такого аддона є доцільним, оскільки він відповідає вимогам автоматизації процесу підготовки моделей і підвищує ефективність роботи користувачів, зменшуючи час і зусилля, необхідні для виконання завдань з 3D-моделювання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Rossney B., Kavanagh C. Reimagining Characters with Unreal Engine's MetaHuman Creator. – Packt Publishing, 2023. – 240 с.

2. How To Get Started With Unreal Engine 5: Unleash Your Creativity. – Independently Published, 2024. – 240 с.

3. MetaHuman Creator Overview – [Електронний ресурс]. – Режим доступу: https://dev.epicgames.com/documentation/en-us/metahuman/metahuman-creator

4. Unreal Engine 5.5 Documentation – [Електронний ресурс]. – Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation

5. MetaHuman | Realistic Person Creator - Unreal Engine – [Електронний ресурс]. – Режим доступу: https://www.unrealengine.com/en-US/metahuman

6. Create your own MetaHuman | Unreal Engine 5 tutorial – [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=7lAWhk_aVvc

7. MetaHuman Creator: Tutorial for Beginners - Epic Games Developers – [Електронний ресурс]. – Режим доступу: https://dev.epicgames.com/community/learning/tutorials/y431/unreal-engine-metahuman-creator-tutorial-for-beginners

8. Full custom metahuman workflow tutorial Blender Unreal Substance – [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=Pwm2zc4k1Ms

9. MetaHuman to Blender (Full video, EN) – [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=FRQ-3FzZVuE

10. MetaHuman Editing in Blender: Complete Workflow with MetaReForge – [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=e8CCRm2SFNM

11. Can I Improve My Metahuman in Blender? A Comprehensive Guide – [Електронний ресурс]. – Режим доступу: https://yelzkizi.org/improve-metahuman-in-blender/

12. Blender Python API Documentation – [Електронний ресурс]. – Режим доступу: https://docs.blender.org/api/current/

13. Williamson A. The Blender Python API: Build 3D Tools and Add-ons Using Python. – Independently published, 2021. – 312 с.

14. Blender Foundation. Blender Scripting: Automate Tasks and Extend Functionality with Python. – [Електронний ресурс]. – Режим доступу: https://docs.blender.org/manual/en/latest/advanced/scripting/index.html

15. Takahashi S. Beginning Blender: Open Source 3D Modeling, Animation, and Game Design. – Apress, 2012. – 432 с.

16. Roman Volodin. Blender 3D Add-on Cookbook. – Packt Publishing, 2020. – 382 с.

17. Blender Developers Blog: Writing Add-ons for Blender 2.8+ – [Електронний ресурс]. – Режим доступу: https://code.blender.org/

18. Beardsmore D. Game Development with Blender and Godot: Making 3D Games with Free Software. – Apress, 2021. – 415 с.

19. MetaHuman Assets Overview – [Електронний ресурс]. – Режим доступу: https://dev.epicgames.com/documentation/en-us/metahuman/metahuman-assets-overview?utm_source=chatgpt.com

20. MetaHuman Documentation | Epic Developer Community – [Електронний ресурс]. – Режим доступу: https://dev.epicgames.com/documentation/en-us/metahuman/metahuman-documentation?utm_source=chatgpt.com

ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

```
import bpy
from mathutils import Vector, Matrix
import copy

# Інформація про аддон
bl_info = {
    "name": "Metahuman to Mixamo",
    "author": "Illya Berezhnoy",
    "version": (1, 0),
    "blender": (3, 0, 0),
    "location": "3D View > Sidebar > Metahuman",
    "description": "Підготовка моделі Metahuman для Mixamo з
можливістю налаштування скелета в реальному часі",
    "category": "Rigging",
}

# Глобальні змінні
last_armature = None  # Останній створений скелет
last_mesh = None      # Останній створений меш
original_bone_data = {}  # Оригінальні дані кісток
reference_dimensions = None  # Розміри моделі для прив'язки
reference_scale = (0.025, 0.025, 0.025)  # Стандартний масштаб
merged_armature_data = None  # Дані об'єднаного скелета
default_scale = 1.08  # Стандартний коефіцієнт масштабування
default_y_offset = 10.8  # Стандартне зміщення по Y

def update_rig(self, context):
    """Оновлення скелета при зміні параметрів"""
    if last_armature and last_mesh:
        bpy.app.timers.register(
            lambda: apply_rig_adjustments(context),
            first_interval=0.001
        )

def apply_rig_adjustments(context):
    """Застосування налаштувань скелета"""
    global last_armature, last_mesh, original_bone_data,
reference_dimensions, merged_armature_data

    if not last_armature or not last_mesh:
        return

    # Якщо меш не об'єднано - просто застосовуємо збережений скелет
    if not context.scene.merge_mesh and merged_armature_data:
        copy_merged_armature(context)
        return

    # Отримуємо параметри з інтерфейсу
    scale_corr = context.scene.rig_scale_correction
```

```
    y_offset = context.scene.rig_y_offset_percent / 100
    z_offset = context.scene.rig_z_offset_percent / 100
    auto_scale = context.scene.auto_scale_factor

    # Отримуємо розміри моделі
    if reference_dimensions and not context.scene.merge_mesh:
        mesh_dim = reference_dimensions['size']
        mesh_center = reference_dimensions['center']
    else:
        bbox = [last_mesh.matrix_world @ Vector(corner) for corner
in last_mesh.bound_box]
        mesh_min = Vector((min(v.x for v in bbox), min(v.y for v in
bbox), min(v.z for v in bbox)))
        mesh_max = Vector((max(v.x for v in bbox), max(v.y for v in
bbox), max(v.z for v in bbox)))
        mesh_dim = mesh_max - mesh_min
        mesh_center = (mesh_max + mesh_min) * 0.5

    # Зберігаємо поточний режим
    current_mode = last_armature.mode
    active_obj = bpy.context.view_layer.objects.active

    try:
        # Переходимо в режим редагування
        bpy.context.view_layer.objects.active = last_armature
        if last_armature.mode != 'EDIT':
            bpy.ops.object.mode_set(mode='EDIT')

        # Матриця повороту на 90 градусів по X
        rotation = Matrix.Rotation(1.5708, 4, 'X')

        # Застосовуємо налаштування до кожної кістки
        for bone in last_armature.data.edit_bones:
            if bone.name not in original_bone_data:
                continue

            orig = original_bone_data[bone.name]
            bone.head = orig['head'].copy()
            bone.tail = orig['tail'].copy()

            # Застосовуємо поворот
            bone.head = rotation @ bone.head
            bone.tail = rotation @ bone.tail

            # Застосовуємо масштабування
            scale = (mesh_dim.z * auto_scale) * scale_corr
            bone.head = bone.head * scale + mesh_center
            bone.tail = bone.tail * scale + mesh_center

            # Застосовуємо зміщення
            bone.head.y += y_offset * mesh_dim.y
            bone.head.z += z_offset * mesh_dim.z
            bone.tail.y += y_offset * mesh_dim.y
```

```
            bone.tail.z += z_offset * mesh_dim.z

    finally:
        # Повертаємо попередній режим
        if current_mode != 'EDIT':
            bpy.ops.object.mode_set(mode=current_mode)
        bpy.context.view_layer.objects.active = active_obj

def copy_merged_armature(context):
    """Копіювання скелета з об'єднаного меша з поворотом на 90°"""
    global last_armature, merged_armature_data

    if not merged_armature_data:
        return {'CANCELLED'}

    # Зберігаємо поточний стан
    current_active = context.active_object
    current_mode = current_active.mode if current_active else
'OBJECT'

    try:
        # Видаляємо старий скелет
        if last_armature:
            bpy.data.objects.remove(last_armature, do_unlink=True)

        # Створюємо новий скелет
        armature = bpy.data.armatures.new("Mixamo_Armature")
        last_armature = bpy.data.objects.new("Mixamo_Armature",
armature)
        context.collection.objects.link(last_armature)

        # 1. Створюємо кістки в режимі редагування
        context.view_layer.objects.active = last_armature
        if bpy.ops.object.mode_set.poll():
            bpy.ops.object.mode_set(mode='EDIT')

        bone_map = {}
        edit_bones = last_armature.data.edit_bones

        # Створюємо всі кістки
        for bone_name, bone_data in merged_armature_data.items():
            bone = edit_bones.new(bone_name)
            bone.head = Vector(bone_data['head'])
            bone.tail = Vector(bone_data['tail'])
            bone_map[bone_name] = bone

        # Встановлюємо батьківські зв'язки
        for bone_name, bone_data in merged_armature_data.items():
            if bone_data['parent'] in bone_map:
                bone_map[bone_name].parent =
bone_map[bone_data['parent']]

        # 2. Виходимо в режим об'єкта
```

```
        if bpy.ops.object.mode_set.poll():
            bpy.ops.object.mode_set(mode='OBJECT')

        # 3. Повертаємо скелет на 90° по X
        last_armature.rotation_euler.x = 1.5708  # 90° в радіанах
        if bpy.ops.object.transform_apply.poll():
            bpy.ops.object.transform_apply(rotation=True)

        # 4. Прив'язуємо меші до скелета
        for obj in context.scene.objects:
            if obj.type == 'MESH':
                obj.parent = last_armature
                if "Armature" not in obj.modifiers:
                    mod = obj.modifiers.new("Armature", 'ARMATURE')
                    mod.object = last_armature
                    mod.use_vertex_groups = True

        return {'FINISHED'}

    except Exception as e:
        print(f"Помилка в copy_merged_armature: {str(e)}")
        # Гарантовано виходимо в режим об'єкта
        if last_armature and last_armature.mode != 'OBJECT':
            if bpy.ops.object.mode_set.poll():
                bpy.ops.object.mode_set(mode='OBJECT')
        return {'CANCELLED'}

    finally:
        # Відновлюємо початковий стан
        try:
            if current_active:
                context.view_layer.objects.active = current_active
                if current_mode != 'OBJECT' and current_active.mode
!= current_mode:
                    if bpy.ops.object.mode_set.poll():
                        bpy.ops.object.mode_set(mode=current_mode)
            elif last_armature:
                context.view_layer.objects.active = last_armature
        except Exception:
            pass

class OBJECT_OT_CleanupMetahuman(bpy.types.Operator):
    """Оператор для підготовки моделі Metahuman"""
    bl_idname = "object.cleanup_metahuman"
    bl_label = "Convert MetaHuman to Mixamo"
    bl_description = "Cleans up the model and creates a correctly
scaled Mixamo skeleton"

    def execute(self, context):
        global last_armature, last_mesh, original_bone_data,
reference_dimensions, merged_armature_data

        # Встановлюємо значення за замовчуванням
```

```
        context.scene.rig_scale_correction = default_scale
        context.scene.rig_y_offset_percent = default_y_offset

        lod_level = context.scene.metahuman_lod_level

        # Очищаємо модель
        self.cleanup_metahuman_model(lod_level)
        self.delete_all_bones()
        self.remove_other_lods(lod_level)

        # Отримуємо всі меші для обраного LOD
        lod_meshes = [obj for obj in bpy.data.objects if obj.type ==
'MESH' and lod_level in obj.name]

        # Розраховуємо розміри моделі
        all_corners = []
        for obj in lod_meshes:
            all_corners.extend([obj.matrix_world @ Vector(corner)
for corner in obj.bound_box])

        if all_corners:
            mesh_min = Vector((min(v.x for v in all_corners),
min(v.y for v in all_corners), min(v.z for v in all_corners)))
            mesh_max = Vector((max(v.x for v in all_corners),
max(v.y for v in all_corners), max(v.z for v in all_corners)))

            reference_dimensions = {
                'size': mesh_max - mesh_min,
                'center': (mesh_max + mesh_min) * 0.5
            }

        if context.scene.merge_mesh:
            # Об'єднуємо меші
            self.merge_lod(lod_level)
            self.merge_by_distance()

            # Застосовуємо стандартний масштаб
            if last_mesh:
                last_mesh.scale = reference_scale
                bpy.context.view_layer.objects.active = last_mesh
                bpy.ops.object.transform_apply(scale=True)

            # Створюємо скелет
            if not last_mesh:
                self.report({'ERROR'}, "Меш не знайдено!")
                return {'CANCELLED'}

            # Видаляємо існуючі групи вершин
            for obj in [obj for obj in bpy.data.objects if obj.type
== 'MESH']:
                if obj.vertex_groups:
                    for vg in obj.vertex_groups:
                        obj.vertex_groups.remove(vg)
```

```
            # Створюємо скелет
            armature = bpy.data.armatures.new("Mixamo_Armature")
            last_armature = bpy.data.objects.new("Mixamo_Armature",
armature)
            bpy.context.collection.objects.link(last_armature)

            self.create_bones(last_armature)
            self.store_original_bone_positions(last_armature)
            apply_rig_adjustments(context)

            # Зберігаємо дані скелета
            merged_armature_data = {}
            bpy.context.view_layer.objects.active = last_armature
            bpy.ops.object.mode_set(mode='EDIT')

            for bone in last_armature.data.edit_bones:
                merged_armature_data[bone.name] = {
                    'head': bone.head.copy(),
                    'tail': bone.tail.copy(),
                    'parent': bone.parent.name if bone.parent else
None
                }

            bpy.ops.object.mode_set(mode='OBJECT')

            # Прив'язуємо всі меші до скелета
            for obj in [obj for obj in bpy.data.objects if obj.type
== 'MESH']:
                obj.parent = last_armature
                modifier = obj.modifiers.new(name="Armature",
type='ARMATURE')
                modifier.object = last_armature
                modifier.use_vertex_groups = True

                # Створюємо групи вершин для автоматичної прив'язки
                for bone in last_armature.pose.bones:
                    obj.vertex_groups.new(name=bone.name)
        else:
            # Для необ'єднаних мешів просто застосовуємо збережений
скелет
            if merged_armature_data:
                copy_merged_armature(context)
            else:
                # Якщо немає збереженого скелета, створюємо новий
                armature = bpy.data.armatures.new("Mixamo_Armature")
                last_armature =
bpy.data.objects.new("Mixamo_Armature", armature)
                bpy.context.collection.objects.link(last_armature)

                self.create_bones(last_armature)
                self.store_original_bone_positions(last_armature)
                apply_rig_adjustments(context)
```

```
                # Зберігаємо дані скелета
                merged_armature_data = {}
                bpy.context.view_layer.objects.active =
last_armature
                bpy.ops.object.mode_set(mode='EDIT')

                for bone in last_armature.data.edit_bones:
                    merged_armature_data[bone.name] = {
                        'head': bone.head.copy(),
                        'tail': bone.tail.copy(),
                        'parent': bone.parent.name if bone.parent
else None
                    }

                bpy.ops.object.mode_set(mode='OBJECT')

            # Застосовуємо масштаб до кожного об'єкта окремо
            for obj in lod_meshes:
                # Зберігаємо оригінальну позицію
                original_location = obj.location.copy()
                original_rotation = obj.rotation_euler.copy()

                # Застосовуємо масштаб
                obj.scale = reference_scale
                bpy.context.view_layer.objects.active = obj
                bpy.ops.object.transform_apply(scale=True)

                # Відновлюємо оригінальну позицію і поворот
                obj.location = original_location
                obj.rotation_euler = original_rotation

                last_mesh = obj  # Використовуємо останній меш для
прив'язки

            # Прив'язуємо всі меші до скелета
            for obj in [obj for obj in bpy.data.objects if obj.type
== 'MESH']:
                obj.parent = last_armature
                modifier = obj.modifiers.new(name="Armature",
type='ARMATURE')
                modifier.object = last_armature
                modifier.use_vertex_groups = True

                # Створюємо групи вершин для автоматичної прив'язки
                for bone in last_armature.pose.bones:
                    obj.vertex_groups.new(name=bone.name)

        # Налаштовуємо параметри для редагування
        if last_armature:
            last_armature.show_in_front = True
            for obj in bpy.context.selected_objects:
                obj.select_set(False)
```

```
            last_armature.select_set(True)
            bpy.context.view_layer.objects.active = last_armature

            # Налаштовуємо параметри кісток у режимі редагування
            bpy.ops.object.mode_set(mode='EDIT')
            for bone in last_armature.data.edit_bones:
                bone.use_relative_parent = False
                bone.use_inherit_rotation = True
                bone.use_inherit_scale = True

            # Переходимо в режим позу для вільного редагування
            bpy.ops.object.mode_set(mode='POSE')
            bpy.ops.pose.select_all(action='SELECT')
            bpy.ops.pose.transforms_clear()

        self.report({'INFO'}, f"Metahuman {lod_level} готовий!
Модель очищено і створено скелет.")
        return {'FINISHED'}

    def merge_by_distance(self):
        """Об'єднання вершин, що збігаються"""
        if last_mesh:
            bpy.context.view_layer.objects.active = last_mesh
            bpy.ops.object.mode_set(mode='EDIT')
            bpy.ops.mesh.select_all(action='SELECT')
            bpy.ops.mesh.remove_doubles(threshold=0.0001)
            bpy.ops.object.mode_set(mode='OBJECT')

    def store_original_bone_positions(self, armature_obj):
        """Зберігаємо оригінальні позиції кісток"""
        global original_bone_data
        original_bone_data.clear()

        current_mode = armature_obj.mode
        if current_mode != 'OBJECT':
            bpy.ops.object.mode_set(mode='OBJECT')

        bpy.context.view_layer.objects.active = armature_obj
        bpy.ops.object.mode_set(mode='EDIT')

        for bone in armature_obj.data.edit_bones:
            original_bone_data[bone.name] = {
                'head': bone.head.copy(),
                'tail': bone.tail.copy(),
                'matrix': bone.matrix.copy()
            }

        bpy.ops.object.mode_set(mode=current_mode)

    def cleanup_metahuman_model(self, lod_level):
        """Видалення непотрібних об'єктів"""
        objects_to_remove = [
            obj for obj in bpy.data.objects
```

```
            if obj.type in {'MESH', 'EMPTY', 'ARMATURE'}
            and 'LOD' in obj.name
            and not obj.name.endswith(lod_level)
        ]
        for obj in objects_to_remove:
            bpy.data.objects.remove(obj, do_unlink=True)

    def delete_all_bones(self):
        """Видалення всіх скелетів"""
        armatures = [obj for obj in bpy.data.objects if obj.type ==
'ARMATURE']
        for armature in armatures:
            bpy.data.objects.remove(armature, do_unlink=True)

    def remove_other_lods(self, lod_level):
        """Видалення інших LOD рівнів"""
        objects_to_remove = [
            obj for obj in bpy.data.objects
            if obj.type == 'MESH'
            and 'LOD' in obj.name
            and not obj.name.endswith(lod_level)
        ]
        for obj in objects_to_remove:
            bpy.data.objects.remove(obj, do_unlink=True)

    def merge_lod(self, lod_level):
        """Об'єднання мешів для обраного LOD"""
        global last_mesh
        lod_meshes = [obj for obj in bpy.data.objects if obj.type ==
'MESH' and lod_level in obj.name]
        if len(lod_meshes) > 1:
            bpy.ops.object.select_all(action='DESELECT')
            for obj in lod_meshes:
                obj.select_set(True)
            bpy.context.view_layer.objects.active = lod_meshes[0]
            bpy.ops.object.join()
            bpy.context.object.name = f"Metahuman_{lod_level}"
            last_mesh = bpy.context.object

    def adjust_lod_scale(self, lod_level):
        """Застосування масштабу для LOD"""
        lod_mesh = next((obj for obj in bpy.data.objects if obj.type
== 'MESH' and lod_level in obj.name), None)
        if lod_mesh:
            lod_mesh.scale = (0.025, 0.025, 0.025)
            bpy.context.view_layer.objects.active = lod_mesh
            bpy.ops.object.transform_apply(scale=True)

    def create_bones(self, armature_obj):
        """Створення кісток скелета"""
        bpy.context.view_layer.objects.active = armature_obj
        bpy.ops.object.mode_set(mode='EDIT')
```

```
        # Дані для створення кісток Mixamo
        bones_data = [
            # Коренева кістка і хребет
            ("mixamorig:Hips", None,
     (0.004466414451599121, 2.385824203491211, 0.06472378224134445),
     (0.004466414451599121, 2.6421542167663574,
0.06472378224134445)),

    ("mixamorig:Spine", "mixamorig:Hips",
     (0.004466414451599121, 2.629953384399414, 0.04066042602062225),
     (0.004466414451599121, 2.9147708415985107,
0.012586494907736778)),

    ("mixamorig:Spine1", "mixamorig:Spine",
     (0.004466413985937834, 2.9147706031799316,
0.012586522847414017),
     (0.004466413985937834, 3.2402760982513428, -
0.019497960805892944)),

    ("mixamorig:Spine2", "mixamorig:Spine1",
     (0.004466413985937834, 3.2402760982513428, -
0.019497960805892944),
     (0.004466413985937834, 3.5810635089874268, -
0.05308876931667328)),

    ("mixamorig:Neck", "mixamorig:Spine2",
     (0.004466413985937834, 3.6064693927764893, -
0.05559299886226654),
     (0.004466413985937834, 3.8145534992218018, -
0.05559299886226654)),

    ("mixamorig:Head", "mixamorig:Neck",
     (0.004466414451599121, 3.8085379600524902, -
0.005921110510826111),
     (0.004466414451599121, 4.317898750305176, -
0.005921110510826111)),

    ("mixamorig:HeadTop_End", "mixamorig:Head",
     (0.004466414451599121, 4.303173542022705, 0.11566891521215439),
     (0.004466414451599121, 4.812534332275391,
0.11566891521215439)),

    # Левая рука (полная иерархия)
    ("mixamorig:LeftShoulder", "mixamorig:Spine2",
     (0.150476336479187, 3.533641815185547, -0.05562683939933777),
     (0.44572964310646057, 3.3877341747283936, -
0.055694542825222015)),

    ("mixamorig:LeftArm", "mixamorig:LeftShoulder",
     (0.44572964310646057, 3.3877341747283936, -
0.055694542825222015),
     (0.7558693885803223, 2.9473185539245605, -
0.03695397078990936)),
```

```
("mixamorig:LeftForeArm", "mixamorig:LeftArm",
 (0.7558692097663879, 2.9473185539245605, -0.03695392608642578),
 (1.0749115943908691, 2.5717883110046387, 0.2656782865524292)),

("mixamorig:LeftHand", "mixamorig:LeftForeArm",
 (1.0749115943908691, 2.5717883110046387, 0.2656779885292053),
 (1.2222678661346436, 2.3509602546691895, 0.4761013090610504)),

# Пальцы левой руки (все привязаны к LeftHand)
("mixamorig:LeftHandThumb1", "mixamorig:LeftHand",
 (1.0776318311691284, 2.4774763584136963, 0.4015352725982666),
 (1.0879580974578857, 2.3947877883911133, 0.5043256282806396)),

("mixamorig:LeftHandThumb2", "mixamorig:LeftHandThumb1",
 (1.0798081159591675, 2.402027130126953, 0.5102209448814392),
 (1.0752830505371094, 2.335273265838623, 0.5767773389816284)),

("mixamorig:LeftHandThumb3", "mixamorig:LeftHandThumb2",
 (1.0742988586425781, 2.3361175060272217, 0.5775400996208191),
 (1.0743309259414673, 2.289005756378174, 0.6286320686340332)),

("mixamorig:LeftHandThumb4", "mixamorig:LeftHandThumb3",
 (1.0834197998046875, 2.282243013381958, 0.6204847693443298),
 (1.0834518671035767, 2.23513126373291, 0.671576738357544)),

("mixamorig:LeftHandIndex1", "mixamorig:LeftHand",
 (1.199359655380249, 2.3573005199432373, 0.566932737827301),
 (1.210255742073059, 2.295619487762451, 0.6074967980384827)),

("mixamorig:LeftHandIndex2", "mixamorig:LeftHandIndex1",
 (1.2113642692565918, 2.294872760772705, 0.6060146689414978),
 (1.2090190649032593, 2.2323248386383057, 0.6369946002960205)),

("mixamorig:LeftHandIndex3", "mixamorig:LeftHandIndex2",
 (1.209075689315796, 2.232285976409912, 0.6369199156761169),
 (1.1917074918746948, 2.175877809524536, 0.6532955169677734)),

("mixamorig:LeftHandIndex4", "mixamorig:LeftHandIndex3",
 (1.1905485391616821, 2.1767184734344482, 0.6548281311988831),
 (1.173180341720581, 2.1203103065490723, 0.6712037324905396)),

("mixamorig:LeftHandMiddle1", "mixamorig:LeftHand",
 (1.2411388158798218, 2.3322720527648926, 0.53094881772995),
 (1.2591801881790161, 2.266444206237793, 0.571134090423584)),

("mixamorig:LeftHandMiddle2", "mixamorig:LeftHandMiddle1",
 (1.260070562362671, 2.265887975692749, 0.5697866082191467),
 (1.2469531297683716, 2.1964476108551025, 0.5906816720962524)),

("mixamorig:LeftHandMiddle3", "mixamorig:LeftHandMiddle2",
 (1.246472716331482, 2.196760654449463, 0.5914004445075989),
 (1.2353589534759521, 2.137199878692627, 0.6094145178794861)),
```

```
    ("mixamorig:LeftHandMiddle4", "mixamorig:LeftHandMiddle3",
     (1.234946846961975, 2.1374683380126953, 0.6100308299064636),
     (1.2238330841064453, 2.0779075622558594, 0.6280449032783508)),

    ("mixamorig:LeftHandRing1", "mixamorig:LeftHand",
     (1.267598271369934, 2.3206799030303955, 0.4812694787979126),
     (1.290547251701355, 2.2578372955322266, 0.5124799013137817)),

    ("mixamorig:LeftHandRing2", "mixamorig:LeftHandRing1",
     (1.2905583381652832, 2.257831335067749, 0.5124598741531372),
     (1.2865325212478638, 2.190828323364258, 0.5301330089569092)),

    ("mixamorig:LeftHandRing3", "mixamorig:LeftHandRing2",
     (1.2865279912948608, 2.190830707550049, 0.5301414728164673),
     (1.2786998748779297, 2.1341724395751953, 0.5426634550094604)),

    ("mixamorig:LeftHandRing4", "mixamorig:LeftHandRing3",
     (1.2786933183670044, 2.134176015853882, 0.542675256729126),
     (1.2708652019500732, 2.0775177478790283, 0.5551972389221191)),

    ("mixamorig:LeftHandPinky1", "mixamorig:LeftHand",
     (1.2733303308486938, 2.3172593116760254, 0.4275699853897095),
     (1.2941370010375977, 2.2544517517089844, 0.4466521739959717)),

    ("mixamorig:LeftHandPinky2", "mixamorig:LeftHandPinky1",
     (1.2935724258422852, 2.2546815872192383, 0.4479690492153168),
     (1.2924147844314575, 2.2006611824035645, 0.45630258321762085)),

    ("mixamorig:LeftHandPinky3", "mixamorig:LeftHandPinky2",
     (1.2925773859024048, 2.2006001472473145, 0.4559188783168793),
     (1.2842098474502563, 2.1527233123779297, 0.4601916968822479)),

    ("mixamorig:LeftHandPinky4", "mixamorig:LeftHandPinky3",
     (1.2846094369888306, 2.152580738067627, 0.45925310254096985),
     (1.2762418985366821, 2.104703903198242, 0.4635259211063385)),

    # Права рука (полная иерархия)
    ("mixamorig:RightShoulder", "mixamorig:Spine2",
     (-0.14154350757598877, 3.533641815185547, -
0.05555913969874382),
     (-0.4367963373661041, 3.3877341747283936, -
0.055491428822278976)),

    ("mixamorig:RightArm", "mixamorig:RightShoulder",
     (-0.4367963373661041, 3.3877341747283936, -
0.055491428822278976),
     (-0.7469359636306763, 2.9473183155059814, -
0.04118301719427109)),

    ("mixamorig:RightForeArm", "mixamorig:RightArm",
     (-0.7469359636306763, 2.9473183155059814, -
0.04118301719427109),
```

```
 (-1.065978765487671, 2.5717880725860596, 0.2608737349510193)),

("mixamorig:RightHand", "mixamorig:RightForeArm",
 (-1.065978765487671, 2.5717880725860596, 0.2608737349510193),
 (-1.2197452783584595, 2.3519091606140137, 0.4742776155471802)),

# Пальцы правой руки (все привязаны к RightHand)
("mixamorig:RightHandThumb1", "mixamorig:RightHand",
 (-1.0732243061065674, 2.477506160736084, 0.40001171827316284),
 (-1.0871096849441528, 2.3949854373931885, 0.5054159164428711)),

("mixamorig:RightHandThumb2", "mixamorig:RightHandThumb1",
 (-1.0790213346481323, 2.40208101272583, 0.5113220810890198),
 (-1.0755380392074585, 2.3354809284210205, 0.5777959227561951)),

("mixamorig:RightHandThumb3", "mixamorig:RightHandThumb2",
 (-1.074903964996338, 2.336017608642578, 0.5782934427261353),
 (-1.0778849124908447, 2.2924070358276367, 0.6284819841384888)),

("mixamorig:RightHandThumb4", "mixamorig:RightHandThumb3",
 (-1.0864903926849365, 2.285991668701172, 0.6206351518630981),
 (-1.0894713401794434, 2.2423810958862305, 0.6708236932754517)),

("mixamorig:RightHandIndex1", "mixamorig:RightHand",
 (-1.2015851736068726, 2.3576226234436035, 0.5671381950378418),
 (-1.2106187343597412, 2.2944769859313965, 0.6074252128601074)),

("mixamorig:RightHandIndex2", "mixamorig:RightHandIndex1",
 (-1.2116276025772095, 2.2937896251678467, 0.6060804128646851),
 (-1.210224986076355, 2.233306407928467, 0.6370405554771423)),

("mixamorig:RightHandIndex3", "mixamorig:RightHandIndex2",
 (-1.2103497982025146, 2.2332193851470947, 0.6368758082389832),
 (-1.1940972805023193, 2.176095962524414, 0.6547911763191223)),

("mixamorig:RightHandIndex4", "mixamorig:RightHandIndex3",
 (-1.1929699182510376, 2.176919937133789, 0.6562793850898743),
 (-1.1767174005508423, 2.1197965145111084, 0.6741947531700134)),

("mixamorig:RightHandMiddle1", "mixamorig:RightHand",
 (-1.2351456880569458, 2.3370351791381836, 0.5260183811187744),
 (-1.2585639953613281, 2.2744314670562744, 0.5668630599975586)),

("mixamorig:RightHandMiddle2", "mixamorig:RightHandMiddle1",
 (-1.2594224214553833, 2.2738969326019287, 0.5655171871185303),
 (-1.248205304145813, 2.2023708820343018, 0.5876500606536865)),

("mixamorig:RightHandMiddle3", "mixamorig:RightHandMiddle2",
 (-1.2477225065231323, 2.202683687210083, 0.5883963704109192),
 (-1.239039421081543, 2.139045000076294, 0.6089265942573547)),

("mixamorig:RightHandMiddle4", "mixamorig:RightHandMiddle3",
 (-1.2386599779129028, 2.1392903327941895, 0.6095137000083923),
```

```
     (-1.2299768924713135, 2.0756516456604004, 0.6300439238548279)),

    ("mixamorig:RightHandRing1", "mixamorig:RightHand",
     (-1.2626978158950806, 2.319765090942383, 0.47919395565986633),
     (-1.2908254861831665, 2.256944417953491, 0.5132892727851868)),

    ("mixamorig:RightHandRing2", "mixamorig:RightHandRing1",
     (-1.290846586227417, 2.2569329738616943, 0.5132506489753723),
     (-1.2860709428787231, 2.191033363342285, 0.5304581522941589)),

    ("mixamorig:RightHandRing3", "mixamorig:RightHandRing2",
     (-1.286062479019165, 2.191037893295288, 0.5304738879203796),
     (-1.2764484882354736, 2.131370782852173, 0.5431901812553406)),

    ("mixamorig:RightHandRing4", "mixamorig:RightHandRing3",
     (-1.2764360904693604, 2.131377696990967, 0.5432130694389343),
     (-1.266822099685669, 2.0717105865478516, 0.5559293627738953)),

    ("mixamorig:RightHandPinky1", "mixamorig:RightHand",
     (-1.274278163909912, 2.3174073696136475, 0.42785149812698364),
     (-1.2948473691940308, 2.2558741569519043, 0.4468253254890442)),

    ("mixamorig:RightHandPinky2", "mixamorig:RightHandPinky1",
     (-1.2943261861801147, 2.256096363067627, 0.4480627179145813),
     (-1.2923272848129272, 2.2011420726776123, 0.4565178155899048)),

    ("mixamorig:RightHandPinky3", "mixamorig:RightHandPinky2",
     (-1.2925646305084229, 2.201049327850342, 0.4559479355812073),
     (-1.288816213607788, 2.1518704891204834, 0.4626975953578949)),

    ("mixamorig:RightHandPinky4", "mixamorig:RightHandPinky3",
     (-1.289096713066101, 2.1517624855041504, 0.4620249271392822),
     (-1.2853482961654663, 2.102583646774292, 0.46877458691596985)),

    # Левая нога (полная иерархия)
    ("mixamorig:LeftUpLeg", "mixamorig:Hips",
     (0.22844594717025757, 2.2502598762512207, 0.06726624071598053),
     (0.29450535774230957, 1.2094610929489136,
0.04907715320587158)),

    ("mixamorig:LeftLeg", "mixamorig:LeftUpLeg",
     (0.29450535774230957, 1.2094610929489136, 0.04907715320587158),
     (0.33830592036247253, 0.25452154874801636, -
0.013541527092456818)),

    ("mixamorig:LeftFoot", "mixamorig:LeftLeg",
     (0.33830589056015015, 0.25452160835266113, -
0.013541498221457005),
     (0.38819319009780884, 0.011022031307220459,
0.32779422402381897)),

    ("mixamorig:LeftToeBase", "mixamorig:LeftFoot",
     (0.3881932199001312, 0.011022018268704414, 0.3277941942214966),
```

```
     (0.4066004455089569, 0.016046937555074692,
0.5012807250022888)),

    ("mixamorig:LeftToe_End", "mixamorig:LeftToeBase",
     (0.4066005349159241, 0.016046879813075066, 0.5012807250022888),
     (0.42500776052474976, 0.021071793511509895,
0.674767255783081)),

    # Правая нога (полная иерархия)
    ("mixamorig:RightUpLeg", "mixamorig:Hips",
     (-0.21951311826705933, 2.2502598762512207,
0.06961363554000854),
     (-0.28557252883911133, 1.209460973739624,
0.039354562759399414)),

    ("mixamorig:RightLeg", "mixamorig:RightUpLeg",
     (-0.28557252883911133, 1.209460973739624,
0.039354562759399414),
     (-0.32937324047088623, 0.2545180320739746, -
0.012424513697624207)),

    ("mixamorig:RightFoot", "mixamorig:RightLeg",
     (-0.32937324047088623, 0.2545180320739746, -
0.012424513697624207),
     (-0.37925970554351807, 0.011034220457077026,
0.32504573464393616)),

    ("mixamorig:RightToeBase", "mixamorig:RightFoot",
     (-0.37925970554351807, 0.011034241877496243,
0.32504573464393616),
     (-0.3976665139198303, 0.01606699451804161,
0.499215185642242 43)),

    ("mixamorig:RightToe_End", "mixamorig:RightToeBase",
     (-0.3976665139198303, 0.01606699451804161,
0.49921518564224243),
     (-0.4160732626914978, 0.021099740639328957,
0.6733846068382263))
        ]

        edit_bones = armature_obj.data.edit_bones
        for bone_name, parent_name, head, tail in bones_data:
            bone = edit_bones.new(bone_name)
            if parent_name:
                bone.parent = edit_bones[parent_name]
            bone.head = Vector(head)
            bone.tail = Vector(tail)

        bpy.ops.object.mode_set(mode='OBJECT')

class VIEW3D_PT_CleanupPanel(bpy.types.Panel):
    """Панель інструментів у бічній панелі"""
    bl_label = "Metahuman to Mixamo"
```

```
    bl_idname = "VIEW3D_PT_cleanup_panel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "Metahuman"

    def draw(self, context):
        """Відображення елементів інтерфейсу"""
        layout = self.layout

        col = layout.column()
        col.prop(context.scene, "metahuman_lod_level")
        col.prop(context.scene, "merge_mesh", text="Merge mesh")
        col.operator("object.cleanup_metahuman")

        if last_armature:
            box = layout.box()
            box.label(text="Skeleton setup (real time)")

            row = box.row()
            row.prop(context.scene, "rig_scale_correction",
text="Scale")
            row.prop(context.scene, "auto_scale_factor",
text="Autoscale")

            box.prop(context.scene, "rig_y_offset_percent",
text="Offset Y (%)")
            box.prop(context.scene, "rig_z_offset_percent",
text="Offset Z (%)")

def register():
    """Реєстрація класів і властивостей"""
    bpy.utils.register_class(OBJECT_OT_CleanupMetahuman)
    bpy.utils.register_class(VIEW3D_PT_CleanupPanel)

    # Властивості сцени
    bpy.types.Scene.metahuman_lod_level = bpy.props.EnumProperty(
        name="Level LOD",
        items=[
            ('LOD0', "LOD0", "Highest Detail"),
            ('LOD1', "LOD1", "High Detail"),
            ('LOD2', "LOD2", "Medium Detail"),
            ('LOD3', "LOD3", "Low Detail"),
            ('LOD4', "LOD4", "Lowest Detail"),
        ],
        default='LOD2'
    )

    bpy.types.Scene.merge_mesh = bpy.props.BoolProperty(
        name="Merge mesh",
        description="Merge mesh in one LOD",
        default=True
    )
```

```
    bpy.types.Scene.rig_scale_correction = bpy.props.FloatProperty(
        name="Scale correction",
        default=default_scale,
        min=0.5,
        max=2.0,
        precision=3,
        update=update_rig
    )

    bpy.types.Scene.auto_scale_factor = bpy.props.FloatProperty(
        name="Autoscale correction",
        default=0.21,
        min=0.1,
        max=3.0,
        precision=2,
        update=update_rig
    )

    bpy.types.Scene.rig_y_offset_percent = bpy.props.FloatProperty(
        name="Offset Y",
        default=default_y_offset,
        min=-50,
        max=50,
        precision=1,
        update=update_rig
    )

    bpy.types.Scene.rig_z_offset_percent = bpy.props.FloatProperty(
        name="Offset Z",
        default=-48.3,
        min=-50,
        max=50,
        precision=1,
        update=update_rig
    )

def unregister():
    """Відміна реєстрації класів і властивостей"""
    bpy.utils.unregister_class(OBJECT_OT_CleanupMetahuman)
    bpy.utils.unregister_class(VIEW3D_PT_CleanupPanel)

    # Видалення властивостей сцени
    del bpy.types.Scene.metahuman_lod_level
    del bpy.types.Scene.merge_mesh
    del bpy.types.Scene.rig_scale_correction
    del bpy.types.Scene.auto_scale_factor
    del bpy.types.Scene.rig_y_offset_percent
    del bpy.types.Scene.rig_z_offset_percent

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

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Комп'ютерних наук

ДИПЛОМНА РОБОТА
на ступінь вищої освіти бакалавр
із спеціальності 122 Комп'ютерні науки

Аддон в Blender на мові Python для зручного переносу моделей MetaHuman

Виконав: студент 4 курсу, групи КНД-41
Бережний Ілля Валентинович
Керівник: керівник кафедри Комп'ютерних наук
д.т.н., професор Вишнівський В.В.

Київ - 2025

АКТУАЛЬНІСТЬ РОБОТИ

Сучасні інструменти 3D-моделювання, зокрема Blender, дозволяють створювати складні візуальні сцени з використанням реалістичних персонажів MetaHuman. Проте підготовка таких моделей до подальшої роботи часто потребує складних технічних дій, що ускладнює процес для пересічного користувача. Створення аддону, який автоматизує підготовку моделей MetaHuman у Blender, зменшує технічні бар'єри, спрощує роботу та прискорює виробничий процес. Це робить тему розробки такого інструменту актуальною.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об`єкт дослідження – програмне середовище тривимірного моделювання Blender.

Предмет дослідження – процеси автоматизації імпорту та підготовки моделей MetaHuman.

Мета роботи – дослідження процесу адаптації тривимірних моделей MetaHuman до середовища Blender та розробка програмного засобу (аддону) на мові Python для зручного імпорту, переносу та підготовки моделей MetaHuman до використання в анімаційних проєктах.

Наукове завдання – огляд особливостей підготовки 3D-моделей MetaHuman у Blender; аналіз проблем при інтеграції моделей у виробничі процеси; розробка аддону для автоматизації підготовки моделей у середовищі Blender.

Методи дослідження – мова програмування Python, API Blender, середовище Blender, Unreal Engine, MetaHuman.

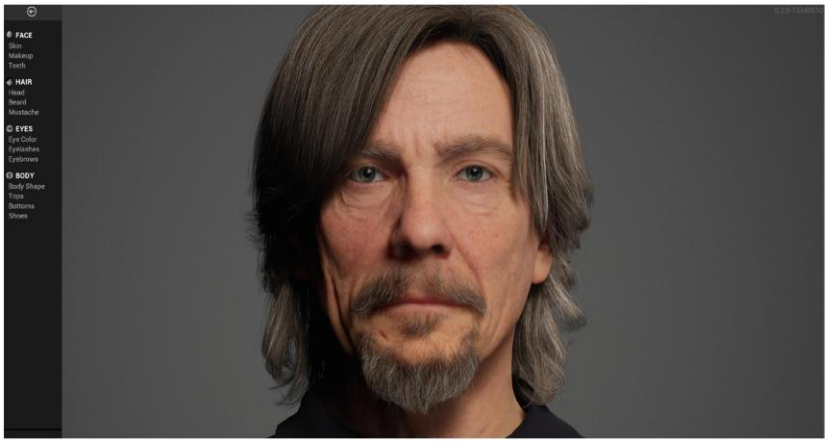
1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОБОТИ З МОДЕЛЯМИ METAHUMAN У BLENDER

Одним із ключових напрямків розвитку стало створення цифрових персонажів, які зовнішньо та поведінково імітують людину. Зросла популярність процедурного моделювання, де значна частина рутинної роботи автоматизується за допомогою параметричних систем. Це дозволяє скоротити час на створення складних моделей та забезпечити гнучкість у зміні їх характеристик.

Таким чином, сучасні тенденції у сфері 3D-моделювання вказують на все більшу взаємодію різних програмних платформ. Blender і Unreal Engine 5 виступають не лише як інструменти, а як можливе ядро кросплатформенних робочих процесів. Їх інтеграція забезпечує новий рівень якості, зручності та швидкості виробництва 3D-контенту, що є особливо актуальним в умовах зростання вимог до візуального реалізму, інтерактивності та персоналізації цифрових персонажів.

1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОБОТИ З МОДЕЛЯМИ METAHUMAN У BLENDER

Набирає обертів в сучасному процесі створення персонажів MetaHuman Creator — інструмент від Epic Games, який дозволяє створювати фотореалістичних людей з уже налаштованою мімікою, скелетом, текстурами та анімацією.



1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОБОТИ З МОДЕЛЯМИ METAHUMAN У BLENDER

Структура моделі MetaHuman з технічної точки зору

Ця модель складається з кількох ключових компонентів:

- геометрія: тіло, голова, очі, ротова порожнина, волосся, одяг та аксесуари;
- система скелетної анімації
- матеріали та шейдери

1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОБОТИ З МОДЕЛЯМИ METAHUMAN У BLENDER

Програмне середовище Blender та формат FBX

Blender — це безкоштовне 3D-програмне забезпечення з відкритим кодом. Воно використовується найбільшою кількістю 3D-художників, аніматорів, розробників ігор та фахівців з візуалізації. Завдяки своїй гнучкості, модульності та великому набору функцій, які можна назвати розширеними можливостями, Blender став універсальною платформою як для нових користувачів, так і для професіоналів.

Формат FBX є одним із поширених та найфункціональніших стандартів для обміну 3D-даними між різними програмними середовищами. У випадку MetaHuman Creator та Unreal Engine 5 можна вважати, що формат FBX відіграє ключову роль у правильному перенесенні моделей до Blender. Це дуже важливо для розробки персональних цифрових персонажів, яких потім потрібно анімувати або налаштовувати. Він підтримує перенесення складної геометрії, текстур, скінінгу та повного скелетного риггінгу

1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОБОТИ З МОДЕЛЯМИ METAHUMAN У BLENDER

Можливості кастомізації та автоматизації у Blender

Однією з ключових переваг Blender як відкритого 3D-редактора є широка підтримка кастомізації моделей і гнучка система автоматизації, яка дозволяє суттєво підвищити продуктивність роботи з персонажами.

Основні можливості:

- модифікацію геометрії: користувачі можуть редагувати форму обличчя, тіла, пальців, створювати унікальні риси або змінювати анатомічні пропорції;
- роботу з текстурами і матеріалами: інструменти Blender дозволяють легко змінювати колір шкіри, очей, волосся, застосовувати мапи нормалей, шорсткості;
- налаштування одягу та аксесуарів: користувачі можуть імпортувати готовий одяг, або змоделювати його безпосередньо в Blender;
- індивідуальна стилізація персонажа: шляхом комбінування геометричних змін, нових шейдерів і аксесуарів, можна створити повністю унікального героя.

1 ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОБОТИ З МОДЕЛЯМИ METAHUMAN У BLENDER

Порівняння скелетних структур MetaHuman та Mixamo

Основні відмінності		
Характеристика	MetaHuman	Mixamo
Кількість кісток	Більше 150	Близько 70
Анімація обличчя	Так (кістки + blendshape-и)	Ні
Сумісність з UE5	Повна	Часткова
Точність анатомії	Висока	Середня
Призначення	Кінематографічні проєкти, ігри	Швидка анімація, прототипування

2 АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ АДДОНУ ДЛЯ ПІДГОТОВКИ МОДЕЛЕЙ METAHUMAN У BLENDER

Підготовка моделей MetaHuman:

- створення інтерфейсу для користувача
- видалення наявного скелету MetaHuman та створення нового Mixamo;
- оптимізація моделі за рівнями деталізації (LOD);
- додаткові фунції для більш гнучкого налаштування скелету після роботи аддону.

2 АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ АДДОНУ ДЛЯ ПІДГОТОВКИ МОДЕЛЕЙ METAHUMAN У BLENDER

Технічні питання при роботі з моделями MetaHuman:

- Велика кількість об’єктів у сцені;
- Точність позиціонування
- Висока навантаженість сцени
- Ситуації з масштабуванням
- Ієрархія скелета MetaHuman є закритою для модифікацій

2 АНАЛІЗ ТА ШЛЯХИ СТВОРЕННЯ АДДОНУ ДЛЯ ПІДГОТОВКИ МОДЕЛЕЙ METAHUMAN У BLENDER

Перспективи розвитку аддону:

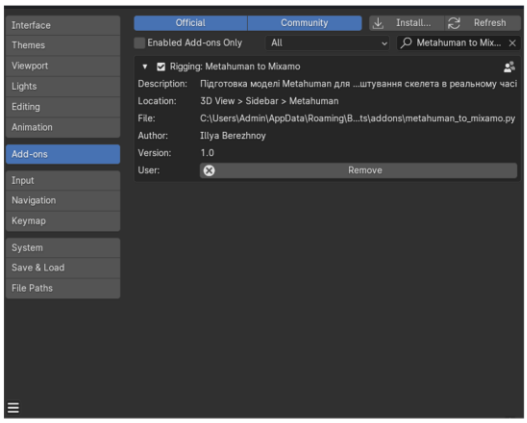
- Розширення функціоналу;
- Розширення функціоналу;
- Власна ретаргет система для переносу анімацій;
- Автоматичне збереження нових анімацій після ретаргету в окрему бібліотеку;
- Можливість створення кастомного ригу, оптимізованого для анімацій.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Файл/Блок	Призначення
__init__.py	Головний файл аддона, реєстрація операторів та панелей
mh_detect	Виявлення скелета MetaHuman у сцені
mh_cleanup	Видалення скелета MetaHuman та обробка мешів
mh_lod	Робота з LOD
mixamo_rig	Створення скелету Mixamo, позиціонування, масштабування
ui_panels	Створення панелей в інтерфейсі Blender
utils	Допоміжні функції

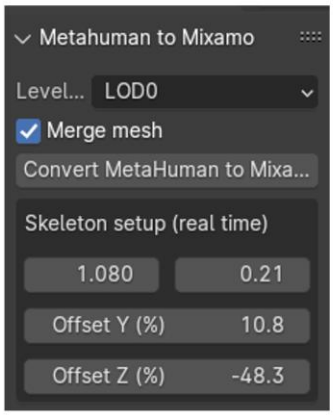
Загальна структура проєкту

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ



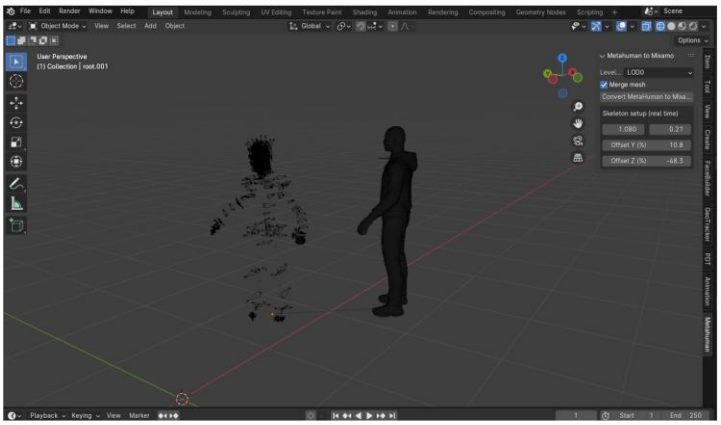
__init__.py

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ



ui_panels

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

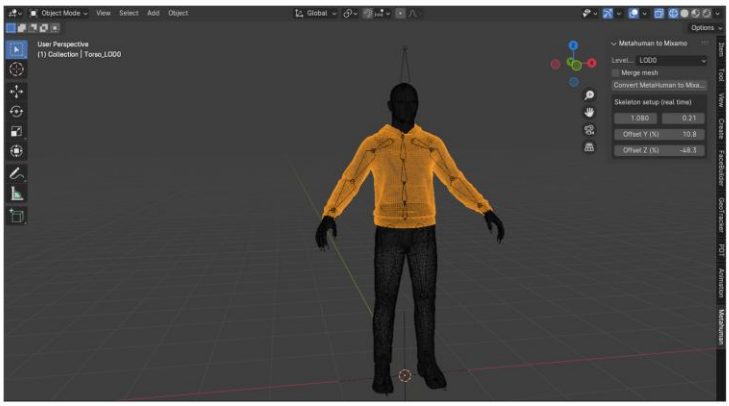


Результат до роботи аддону

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

Результат після роботи аддону

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ



Необ’єданний меш

ВИСНОВКИ

Метою цієї дипломної роботи було дослідження процесів підготовки 3D-моделей MetaHuman у Blender за допомогою спеціалізованого аддона для автоматизації роботи з цими моделями. Основна мета полягала в удосконаленні технології підготовки моделей для подальшого використання в анімаційних або ігрових проектах.

Для її реалізації було зроблено:

- аналіз існуючих рішень та технологій для роботи з 3D-моделями;
- дослідження функціональності аддона для Blender "Metahuman to Mixamo";
- розробка та оптимізація процесу інтеграції MetaHuman з Mixamo Rig для забезпечення зручного та ефективного використання моделей у Blender;
- створення функціоналу для редагування скелетних структур, об’єднання мешів, а також автоматизація позиціювання об'єктів.

ВИСНОВКИ

У результаті реалізації проекту був розроблений аддон, що дозволяє автоматично видаляти старий скелет MetaHuman, створювати новий скелет Mixamo, а також зручний інтерфейс для коректного позиціонування та об’єднання мешів. Цей аддон значно спрощує роботу з моделями MetaHuman у Blender, дозволяючи зменшити час на підготовку моделей і підвищити точність їх використання для подальших анімацій чи ігрових сцен.

ВИСНОВКИ

Практична цінність розробленого аддона полягає в наступному:

- підвищення ефективності підготовки 3D-моделей, завдяки автоматизації рутинних процесів;
- зменшення ймовірності помилок під час обробки моделей завдяки чітко налаштованому алгоритму роботи аддона;
- оптимізація робочого процесу для художників і розробників, які працюють з моделями MetaHuman в Blender.

ВИСНОВКИ

Розроблений аддон дозволяє значно полегшити інтеграцію моделей MetaHuman в інші системи, спрощує процес налаштування анімацій і є важливим інструментом для користувачів Blender, що працюють з високоякісними 3D-моделями. Таким чином, створення такого аддона є доцільним, оскільки він відповідає вимогам автоматизації процесу підготовки моделей і підвищує ефективність роботи користувачів, зменшуючи час і зусилля, необхідні для виконання завдань з 3D-моделювання.