НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі»
Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Віктор Панкратьєв**
ст. викладач (підпис)

Виконав _____ **Олександр Троян**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Трояну Олександру Володимировичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі»

затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи:
алгоритми та методи автоматизованої перебудови топологічної сітки тривимірних моделей 3D-моделей, засоби розробки програмного забезпечення C++, Qt, OpenGL, Assimp, а також вимоги до функціональності пз.
Перелік питань, що підлягають дослідженню:
тривимірна полігональна модель як об'єкт дослідження, методи автоматизованої перебудови топологічної сітки, алгоритми спрощення, підрозбиття та ізотропної обробки сітки, розробка програмного додатку для імпорту, візуалізації, обробки та експорту 3D-моделей.
Перелік графічних документів (за необхідності).

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Віктор Панкратьєв**
(підпис)

Завдання взяв до виконання _____ **Олександр Троян**
(підпис)

ЗМІСТ

ВСТУП

У сучасній комп’ютерній графіці тривимірні моделі широко використовуються у сферах ігрової індустрії, кіновиробництва, архітектурної візуалізації, інженерного проєктування, 3D-друку та віртуальної реальності. Якість топологічної сітки тривимірної моделі суттєво впливає на подальшу роботу з об’єктом: його відображення, оптимізацію, редагування, експорт до інших програмних середовищ та використання в реальному часі. При цьому моделі, отримані з різних джерел, можуть мати надмірну кількість полігонів, нерівномірну структуру сітки, вироджені елементи або інші недоліки, що ускладнюють їх подальше застосування.

Актуальність теми полягає в необхідності створення зручного програмного засобу для автоматизованої перебудови топологічної сітки тривимірної моделі. Такий додаток дає змогу користувачеві імпортувати 3D-модель, переглянути її у візуальному середовищі, застосувати один із доступних алгоритмів обробки сітки та експортувати отриманий результат. Особливо важливим є поєднання алгоритмічної обробки геометрії з простим графічним інтерфейсом, оскільки це дозволяє зменшити кількість ручних дій і зробити процес підготовки моделі більш зрозумілим для користувача.

Метою бакалаврської кваліфікаційної роботи є розробка програмного забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі з можливістю імпорту, візуального перегляду, застосування алгоритмів обробки та експорту результату.

Для досягнення поставленої мети необхідно виконати такі завдання: проаналізувати предметну область обробки тривимірних моделей; визначити функціональні та нефункціональні вимоги до програмного додатку; спроєктувати архітектуру системи та її основні програмні компоненти; розробити графічний інтерфейс користувача; реалізувати модуль завантаження та відображення 3D-моделі; реалізувати алгоритми обробки топологічної сітки;

забезпечити можливість збереження параметрів проєкту та експорту обробленої моделі; провести тестування розробленого програмного забезпечення.

Об’єктом дослідження є процес обробки та оптимізації топологічної сітки тривимірних моделей.

Предметом дослідження є методи, алгоритми та програмні засоби автоматизованої перебудови топологічної сітки 3D-моделі в desktop-додатку.

У процесі розробки використано мову програмування C++, фреймворк Qt для створення графічного інтерфейсу користувача, OpenGL для візуалізації тривимірної сцени, бібліотеку Assimp для імпорту моделей, а також SQLite для збереження інформації про проєкт, модель та параметри обробки. У реалізації програми передбачено окремий клас для роботи з внутрішнім представленням геометрії на основі Surface_mesh, а також методи перетворення між ядром геометричної обробки та буфером даних для відображення. Для візуалізації використовується OpenGL-віджет, який відповідає за ініціалізацію графічного контексту, налаштування шейдерів, обертання моделі, масштабування та відображення каркасного режиму.

У програмному додатку реалізовано декілька підходів до обробки топологічної сітки. Для зменшення кількості полігонів використовується алгоритм спрощення моделі Decimate, для збільшення деталізації — алгоритм Subdivide, а для вирівнювання структури сітки — ізотропна перебудова топологічної сітки. Параметри обробки винесено в окрему структуру, яка містить значення довжини ребра, кількості ітерацій, рівня поділу та коефіцієнта спрощення. Такий підхід спрощує налаштування алгоритмів і дозволяє гнучко розширювати функціональність програми в майбутньому.

Практична цінність роботи полягає у створенні desktop-додатку, який може бути використаний для базової підготовки тривимірних моделей перед подальшим редагуванням, оптимізацією або експортом. Розроблена система поєднує функції перегляду 3D-моделі, вибору алгоритму обробки, налаштування параметрів, збереження даних про проєкт та експорту результату. Це робить програму корисною для користувачів, які працюють із 3D-графікою

та потребують швидкого інструменту для автоматизованої зміни структури сітки.

Апробація результатів роботи здійснювалася шляхом тестування програмного додатку на різних 3D-моделях, аналізу змін кількості вершин, граней і ребер після застосування алгоритмів, а також перевірки коректності імпорту, відображення та експорту моделі. Також результати роботи були представлені у вигляді тез на студентській науковій конференції.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто предметну область, проаналізовано вимоги до програмної системи, виконано огляд існуючих рішень і сформульовано постановку завдання. У другому розділі наведено проєктування інформаційного та програмного забезпечення, зокрема логічну модель даних, діаграми класів, компонентів і пакетів. У третьому розділі описано вибір системи управління інформаційною базою, розробку бази даних, інструментальні засоби створення додатку та реалізацію основних програмних модулів. У четвертому розділі наведено результати тестування, вимоги до апаратного і програмного забезпечення, а також рекомендації щодо впровадження та експлуатації системи.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметною областю бакалаврської кваліфікаційної роботи є обробка тривимірних моделей, зокрема автоматизована перебудова топологічної сітки полігональних 3D-об’єктів. Тривимірна модель є цифровим поданням об’єкта у просторі та використовується для візуалізації, моделювання, проєктування, анімації, створення ігрових ресурсів, підготовки моделей до 3D-друку та інших задач комп’ютерної графіки.

У більшості випадків поверхня 3D-моделі подається у вигляді полігональної сітки. Така сітка складається з вершин, ребер і граней. Вершини визначають координати точок у тривимірному просторі, ребра задають зв’язки між цими точками, а грані формують поверхню об’єкта. Найчастіше у практичних задачах використовуються трикутні або чотирикутні полігони, оскільки вони зручні для обчислень, відображення та подальшої обробки.

Якість топологічної сітки має значний вплив на можливість подальшого використання 3D-моделі. Коректно побудована сітка повинна достатньо точно передавати форму об’єкта, мати зрозумілу структуру, не містити зайвих або пошкоджених елементів і відповідати вимогам задачі, для якої модель створюється. Якщо модель використовується в реальному часі, важливим є зменшення кількості полігонів для підвищення продуктивності. Якщо модель готується до згладження, деталізації або подальшого редагування, важливою стає рівномірність і передбачуваність структури її сітки.

На практиці 3D-моделі часто мають недоліки, які ускладнюють їх використання. До таких недоліків належать надмірна кількість полігонів, нерівномірний розподіл граней, занадто довгі або занадто короткі ребра, вироджені грані, ізольовані вершини, відкриті межі, дублювання елементів та інші геометричні або топологічні проблеми. Такі дефекти можуть виникати

після сканування об’єктів, експорту з різних редакторів, автоматичної генерації моделей або багаторазового редагування сітки.

Однією з важливих задач у цій предметній області є приведення топологічної сітки до стану, придатного для подальшого використання. Для цього застосовуються різні методи обробки полігональної геометрії. Спрощення сітки дозволяє зменшити кількість полігонів і зробити модель менш ресурсомісткою. Поділ полігонів, навпаки, збільшує деталізацію моделі та робить її поверхню більш гладкою. Вирівнювання сітки спрямоване на отримання більш рівномірного розміру та форми елементів, що покращує загальну якість геометричної структури.

Перебудова топологічної сітки є процесом зміни структури полігональної моделі без суттєвої втрати її початкової форми. Вона може включати зміну кількості вершин, ребер і граней, перерозподіл полігонів по поверхні, усунення дефектних елементів, покращення форми трикутників або чотирикутників, а також підготовку моделі до подальшої обробки. Основна мета цього процесу полягає в тому, щоб отримати сітку, яка краще відповідає вимогам конкретного використання.

У сфері комп’ютерної графіки існує декілька типових сценаріїв використання перебудови топологічної сітки. У розробці ігор та інтерактивних застосунків вона потрібна для оптимізації моделей і зменшення навантаження на систему. У 3D-моделюванні та анімації якісна сітка полегшує редагування, деформацію та згладження об’єктів. У 3D-друці правильна структура моделі допомагає уникати помилок під час підготовки до друку. В інженерних і наукових задачах якість сітки може впливати на точність візуалізації або подальших обчислень.

Користувачами програмних засобів для обробки топологічних сіток можуть бути 3D-художники, дизайнери, інженери, розробники ігор, фахівці з візуалізації, студенти та дослідники, які працюють із цифровими тривимірними об’єктами. Для таких користувачів важливо мати можливість швидко

завантажити модель, оцінити її стан, виконати необхідну обробку та отримати результат, придатний для подальшого використання.

Загальний процес роботи з тривимірною моделлю у цій предметній області можна подати як послідовність дій: отримання або імпорт моделі, аналіз її геометричної структури, вибір способу обробки, зміна топологічної сітки, візуальна оцінка результату та збереження або експорт обробленої моделі. На кожному з цих етапів важливо забезпечити збереження форми об’єкта, коректність геометрії та відповідність результату потребам користувача.

1.2 Аналіз вимог до програмної системи

Аналіз вимог до програмної системи передбачає визначення потреб користувача, очікуваної поведінки системи та обмежень, які необхідно врахувати під час розв’язання задачі. У межах даної роботи вимоги формуються на основі особливостей предметної області обробки тривимірних моделей та необхідності автоматизованої перебудови топологічної сітки.

Розроблювана система орієнтована на користувачів, які працюють із полігональними 3D-моделями та потребують засобу для зміни структури їхньої сітки. До таких користувачів можуть належати 3D-художники, студенти, дизайнери, розробники ігор, фахівці з комп’ютерної графіки, інженери та інші спеціалісти, які використовують тривимірні об’єкти у своїй діяльності. Основна потреба таких користувачів полягає в можливості швидко завантажити модель, оцінити її геометричну структуру, виконати необхідну обробку та отримати результат, придатний для подальшого використання.

З погляду предметної області система повинна забезпечувати повний цикл роботи з тривимірною моделлю: отримання моделі, її перегляд, аналіз структури сітки, вибір способу обробки, зміну топологічної структури та збереження отриманого результату. Цей процес має бути зрозумілим для користувача і не повинен вимагати від нього виконання великої кількості ручних дій.

Основні вимоги до системи можна поділити на функціональні та нефункціональні. Функціональні вимоги описують, які дії система повинна дозволяти виконувати користувачу. Нефункціональні вимоги визначають загальні властивості системи, зокрема зручність, стабільність, зрозумілість роботи та відповідність очікуванням користувача. У таблиці 1.1 можна переглянути функціональні вимоги до системи.

Таблиця 1.1

Функціональні вимоги до програмної системи

№	**Вимога**	**Опис**
1	Завантаження тривимірної моделі	Користувач повинен мати можливість обрати 3D-модель для подальшої роботи
2	Перегляд тривимірної моделі	Система повинна надавати можливість візуально оцінити форму моделі
3	Вибір способу обробки	Система повинна надавати користувачу декілька способів зміни топологічної сітки
4	Налаштування параметрів обробки	Користувач повинен мати можливість задавати параметри, які впливають на результат обробки
5	Зменшення кількості полігонів	Система повинна дозволяти спрощувати модель для зменшення її складності
6	Збільшення деталізації	Система повинна дозволяти підвищувати деталізацію моделі шляхом поділу елементів сітки
7	Вирівнювання структури сітки	Система повинна дозволяти отримувати більш рівномірний розподіл елементів сітки
8	Перевірка результату обробки	Користувач повинен мати можливість оцінити результат після зміни сітки
9	Збереження обробленої моделі	Система повинна дозволяти зберегти результат для подальшого використання
10	Інформування про помилки	Система повинна повідомляти користувача про ситуації, коли виконання дії неможливе

Основними діями користувача є завантаження моделі, перегляд її форми, оцінка якості сітки, вибір способу обробки, встановлення параметрів і отримання результату. Після виконання обробки користувач повинен мати можливість порівняти очікуваний та фактичний результат візуально, звертаючи

увагу на зміну кількості полігонів, рівномірність сітки, збереження загальної форми об’єкта та відсутність помітних дефектів.

Окремо слід визначити вимоги до способів обробки топологічної сітки. Оскільки різні задачі потребують різного результату, система повинна підтримувати кілька підходів до зміни структури моделі. Якщо модель містить надмірну кількість полігонів, користувачеві потрібне спрощення сітки. Якщо модель має недостатню деталізацію, доцільним є поділ її елементів. Якщо сітка нерівномірна або має полігони різного розміру, потрібне вирівнювання її структури.

Під час обробки тривимірної моделі важливо враховувати параметри, які впливають на кінцевий результат. Вимоги до параметрів повинні забезпечувати баланс між зручністю налаштування та контролем над результатом. Нефункціональні вимоги до програмної системи наведено у таблиці 1.2.

Таблиця 1.2

Нефункціональні вимоги до програмної системи

№	**Вимога**	**Опис**
1	Зручність використання	Основні дії мають бути зрозумілими для користувача без тривалого навчання
2	Послідовність роботи	Процес обробки моделі має виконуватися у логічному порядку
3	Стабільність	Система повинна коректно реагувати на помилки та не завершувати роботу аварійно
4	Передбачуваність	Результат обробки має відповідати вибраному способу та заданим параметрам
5	Захист від некоректних дій	Користувач не повинен мати можливості запускати обробку без завантаженої моделі або з явно неправильними параметрами
6	Інформативність	Система повинна повідомляти користувача про стан виконання дій і можливі помилки

До вхідних даних у межах предметної області належить тривимірна модель, яка містить геометричну інформацію про об’єкт. Також вхідними даними є параметри, які користувач задає перед виконанням обробки.

Вихідними даними є оброблена тривимірна модель, структура якої змінена відповідно до обраного способу обробки.

Важливою вимогою є збереження геометричної цілісності моделі. Після обробки модель не повинна втрачати основну форму, мати помітні розриви поверхні, випадкові отвори або інші дефекти, які ускладнюють її подальше використання. Також необхідно враховувати, що надмірне спрощення може призвести до втрати деталей, а надмірне збільшення деталізації — до ускладнення моделі та зростання обсягу даних.

Отже, аналіз вимог показує, що система для автоматизованої перебудови топологічної сітки повинна забезпечувати зрозумілий процес роботи з тривимірною моделлю, підтримувати основні способи зміни структури сітки, надавати користувачу можливість керувати параметрами обробки та візуально оцінювати результат. Головними вимогами є збереження форми моделі, покращення або зміна структури сітки відповідно до потреб користувача, наочність результату та придатність обробленої моделі для подальшого використання.

1.3 Моделювання предметної області

Моделювання предметної області є важливим етапом системного аналізу, оскільки дозволяє формалізувати основні сутності, процеси та взаємозв’язки, що характерні для задачі автоматизованої перебудови топологічної сітки тривимірної моделі. На відміну від етапів проєктування, де увага приділяється архітектурі та реалізації програмної системи, моделювання предметної області зосереджується на відображенні самої логіки предметної області: які об’єкти в ній існують, які дії виконуються та в якій послідовності відбувається робота з тривимірною моделлю.

Для опису предметної області доцільно використати сукупність UML-діаграм, які дозволяють розглянути її з різних точок зору. Зокрема, діаграма прецедентів дає змогу визначити основні дії користувача, діаграма активності

— відобразити послідовність виконання процесу обробки моделі, діаграма послідовності — показати порядок взаємодії між основними сутностями в межах окремого сценарію, а діаграма класів предметної області — представити основні поняття та зв’язки між ними.

1.3.1 Діаграма прецедентів предметної області. Діаграма прецедентів використовується для опису основних можливостей системи з погляду користувача. Саму діаграму можна побачити на рисунку 1. У межах предметної області основним актором є 3D-художник або інший користувач, який працює з тривимірною моделлю. Його основна мета полягає в тому, щоб завантажити модель, переглянути її, виконати зміну структури сітки та отримати готовий результат для подальшого використання.

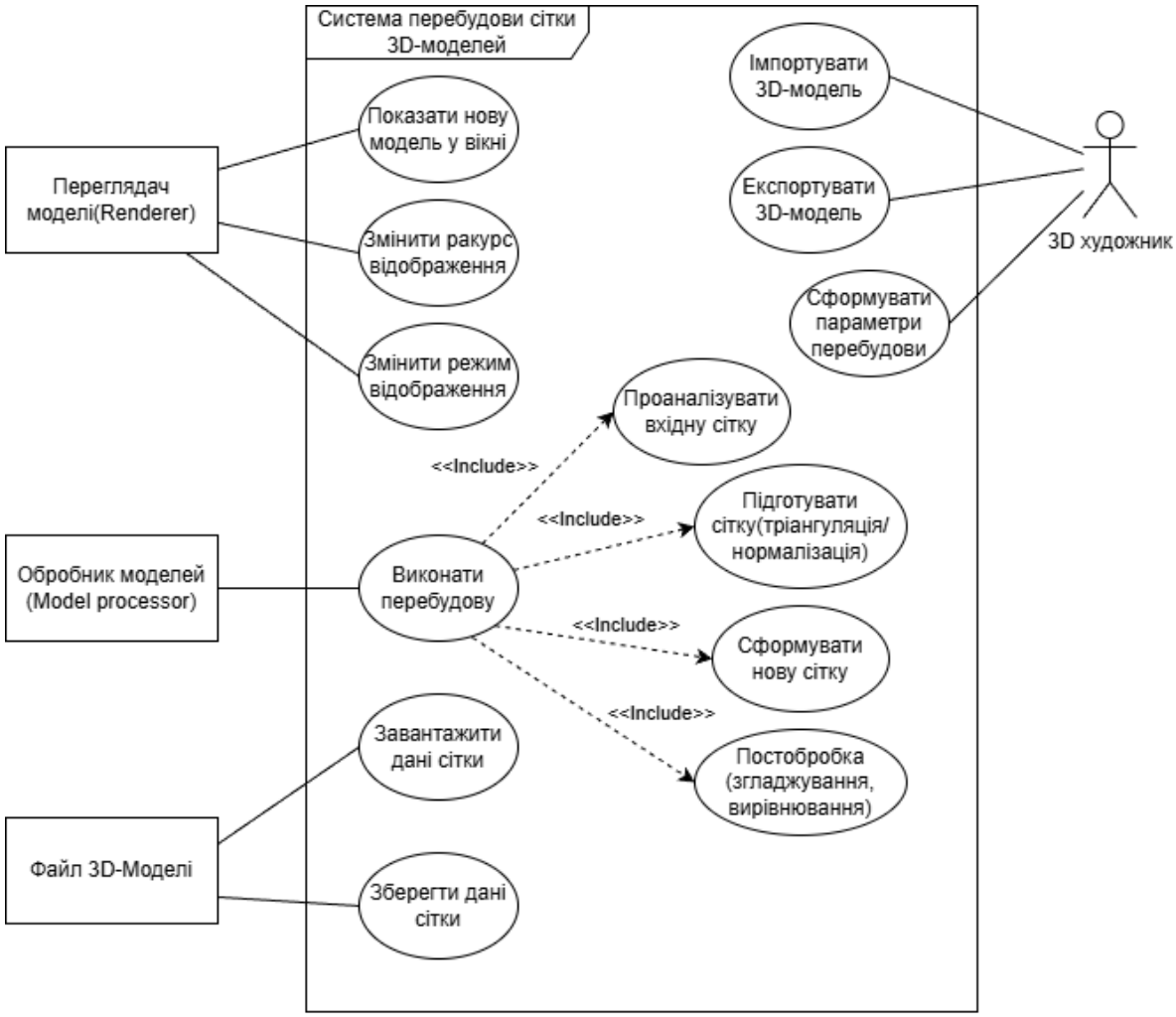


Рис. 1 Діаграма прецедентів предметної області

На діаграмі прецедентів доцільно показати такі основні варіанти використання:

- імпортувати 3D-модель;
- переглянути 3D-модель;
- змінити ракурс відображення;
- змінити режим відображення;
- сформувати параметри перебудови;
- виконати перебудову топологічної сітки;
- переглянути оновлену модель;
- експортувати 3D-модель;
- зберегти дані сітки.

Також на діаграмі можна виділити допоміжні сутності предметної області, які беруть участь у процесі: файл 3D-моделі, переглядач моделі та обробник моделі. Файл 3D-моделі є джерелом початкових даних і місцем збереження результату. Переглядач моделі відповідає за візуальне представлення об’єкта для користувача. Обробник моделі відповідає за зміну структури топологічної сітки.

Важливим прецедентом є виконання перебудови топологічної сітки. Він включає декілька внутрішніх етапів: підготовку сітки, аналіз її початкового стану, формування нової структури та постобробку результату. Саме цей прецедент є центральним для предметної області, оскільки він відображає основну задачу дипломної роботи.

1.3.2 Діаграма послідовності. Діаграма послідовності деталізує типовий сценарій роботи користувача з тривимірною моделлю. У ній відображається порядок дій, які виконуються від моменту вибору файлу моделі до отримання обробленого результату. На рисунку 2 зображено діаграму послідовності для системи.

У представленому сценарії користувач спочатку обирає файл 3D-моделі. Після цього відбувається завантаження даних сітки та формування моделі для

візуального перегляду. Користувач отримує можливість переглянути модель і оцінити її початковий стан. Далі він формує параметри перебудови топологічної сітки, після чого модель передається на обробку.

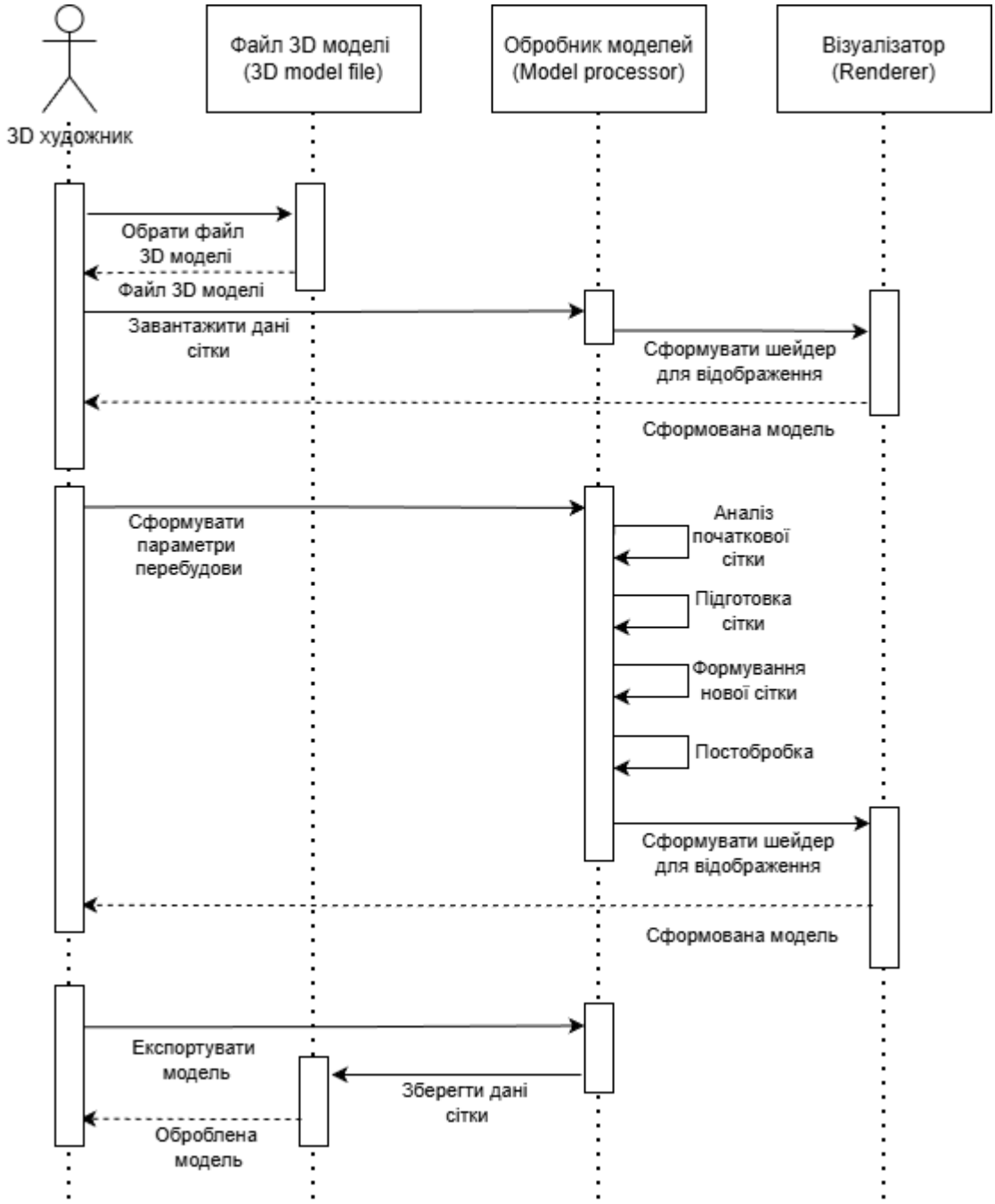


Рис. 2 Діаграма послідовності процесу обробки 3D-моделі

Процес обробки включає аналіз початкової сітки, її підготовку, формування нової сітки та постобробку результату. Після завершення перебудови оновлена модель знову передається для візуального перегляду. Користувач оцінює результат і, якщо він є задовільним, експортує модель у файл. Таким чином,

діаграма послідовності демонструє повний цикл взаємодії користувача з моделлю в межах основного сценарію.

Таке представлення дозволяє зрозуміти, які об’єкти предметної області беруть участь у процесі та в якій послідовності відбувається робота з моделлю.

1.3.3 Діаграма активності. Діаграма активності використовується для відображення загальної логіки процесу роботи з тривимірною моделлю. Вона показує послідовність дій користувача, а також можливі переходи у випадку помилок або незадовільного результату. Діаграма активності зображена на рисунку 3.

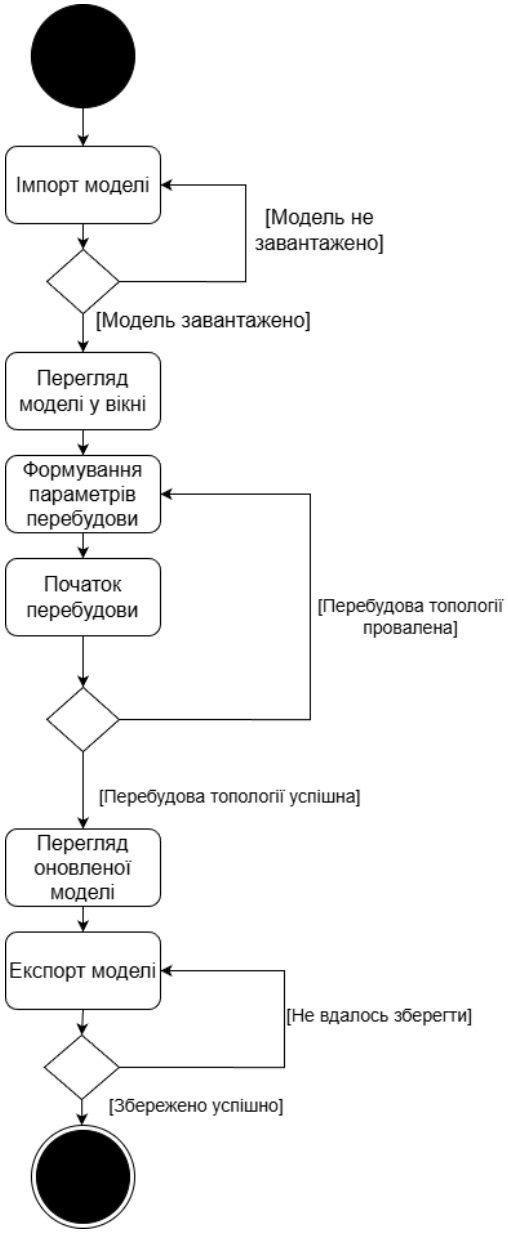


Рис. 3 Діаграма активності процесу роботи з моделлю

Процес починається з імпорту моделі. Якщо модель не була завантажена, користувач повертається до етапу імпорту. Якщо модель завантажена успішно, виконується її перегляд у вікні. Після цього користувач формує параметри перебудови та запускає процес зміни топологічної сітки.

Після виконання перебудови перевіряється її результат. Якщо перебудова топологічної сітки завершилася невдало або отриманий результат не задовольняє користувача, він може повернутися до етапу формування параметрів і повторити процес. Якщо результат є успішним, користувач переглядає оновлену модель і переходить до її експорту. У випадку, якщо модель не вдалося зберегти, процес експорту може бути повторений. Якщо збереження виконано успішно, робота завершується.

Таким чином, діаграма активності демонструє не лише основний сценарій роботи, а й альтернативні переходи, які виникають у разі помилок. Це важливо для предметної області, оскільки робота з 3D-моделями може супроводжуватися некоректними файлами, невдалими параметрами обробки або помилками збереження результату.

1.3.4 Абстракції предметної області. Для побудови моделей предметної області необхідно виділити основні абстракції, які описують процес роботи з тривимірною моделлю. У межах даної роботи можна виділити такі абстракції: 3D-модель, файл 3D-моделі, переглядач 3D-моделі та обробник моделі. Всі абстракції можна переглянути на рисунку 4.

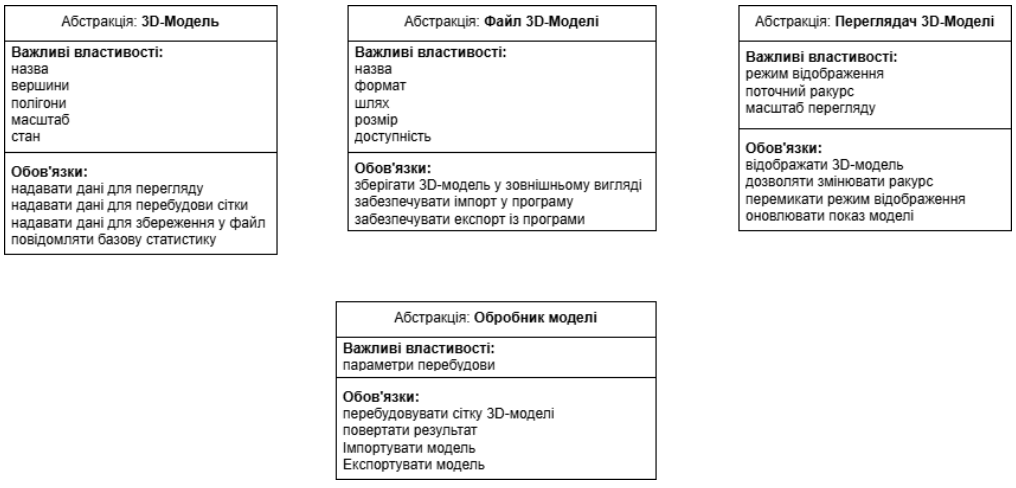


Рис. 4 Абстракції для предметної області проєкту

3D-модель є центральною сутністю предметної області. Вона описує цифровий тривимірний об’єкт, який має назву, кількість вершин, кількість полігонів, масштаб і поточний стан. Основними обов’язками цієї сутності є надання даних для перегляду, надання даних для перебудови сітки, надання даних для збереження у файл та повідомлення базової статистики.

Файл 3D-моделі описує зовнішнє представлення моделі. Він має назву, формат, шлях, розмір і доступність. Основним призначенням цієї сутності є зберігання 3D-моделі у зовнішньому вигляді, забезпечення імпорту моделі та забезпечення експорту результату.

Переглядач 3D-моделі відповідає за візуальну оцінку моделі користувачем. До його властивостей належать режим відображення, поточний ракурс і масштаб перегляду. Основними обов’язками є відображення 3D-моделі, надання можливості змінювати ракурс, перемикати режим відображення та оновлювати показ моделі після обробки.

Обробник моделі описує процес зміни структури топологічної сітки. Його важливою властивістю є параметри перебудови. До обов’язків належать перебудова сітки 3D-моделі, повернення результату, імпорт моделі та експорт моделі.

1.4 Огляд інформаційних джерел та існуючих рішень

1.4.1 Основні напрями обробки полігональних сіток. Одним з основних напрямів обробки полігональних сіток є спрощення сітки. Його метою є зменшення кількості полігонів при збереженні загальної форми моделі. Такий підхід особливо важливий для задач, де потрібно зменшити навантаження на систему, наприклад у комп’ютерних іграх, інтерактивних застосунках або веб-візуалізації. Надмірно деталізована модель може мати велику кількість вершин і граней, що ускладнює її відображення та обробку. Спрощення дозволяє зменшити обсяг геометричних даних і зробити модель більш придатною для використання в реальному часі.

Іншим напрямом є збільшення деталізації сітки. У цьому випадку кількість елементів моделі, навпаки, збільшується. Така обробка може бути потрібна тоді, коли початкова модель має занадто грубу форму або потребує подальшого згладжування. Поділ полігонів дозволяє зробити поверхню моделі більш деталізованою та підготувати її до подальших операцій, наприклад деформації, скульптингу або візуального покращення. При цьому важливо враховувати, що надмірне збільшення кількості полігонів може призвести до зростання складності моделі.

Важливе місце займає вирівнювання структури полігональної сітки. Його метою є отримання більш рівномірного розподілу елементів по поверхні моделі. У багатьох випадках початкова сітка може містити полігони різного розміру, довгі вузькі трикутники або ділянки з нерівномірною щільністю. Це може негативно впливати на якість подальшої обробки, згладжування або деформації. Вирівнювання сітки дозволяє зробити її структуру більш передбачуваною, а елементи — ближчими за розміром і формою.

Окремим напрямом є триангуляція поверхні. Вона полягає у перетворенні граней моделі на трикутники. Трикутні полігони часто використовуються в комп’ютерній графіці, оскільки вони є простими для обчислень і стабільними для відображення. Навіть якщо модель створена з використанням багатокутників або чотирикутників, на певних етапах обробки її поверхня може бути перетворена на трикутну сітку. Це спрощує подальші геометричні операції та забезпечує однозначне представлення поверхні.

Ще одним важливим напрямом є очищення геометрії. Воно передбачає виявлення та усунення некоректних елементів сітки. До таких елементів можуть належати ізольовані вершини, дубльовані точки, вироджені грані, самоперетини, відкриті або пошкоджені ділянки поверхні. Наявність таких дефектів може призводити до помилок під час обробки моделі або до некоректного результату після експорту. Очищення геометрії допомагає підготувати модель до подальшої роботи та зменшити ризик виникнення помилок.

Також важливим є перерахунок нормалей поверхні. Нормалі використовуються для визначення напряму поверхні та впливають на те, як модель виглядає при освітленні. Якщо нормалі задані неправильно, модель може відображатися з візуальними дефектами: затемненими ділянками, неправильним освітленням або вивернутими поверхнями. Після зміни структури сітки нормалі часто потрібно оновлювати, щоб відображення моделі залишалося коректним.

Усі зазначені напрями можуть використовуватися як окремо, так і послідовно. Наприклад, перед вирівнюванням сітки може виконуватися очищення геометрії, після зміни структури — перерахунок нормалей. У практичній роботі з 3D-моделями часто застосовується не один окремий метод, а комбінація кількох підходів, що дозволяє отримати якісніший результат.

1.4.2 Огляд існуючих програмних рішень. У сфері обробки полігональних 3D-моделей існує значна кількість програмних рішень, які дозволяють виконувати редагування, оптимізацію, очищення, згладжування, спрощення та перебудову топологічної сітки. Частина таких інструментів є повноцінними середовищами для 3D-моделювання, інші орієнтовані саме на технічну обробку сіток або підготовку моделей до подальшого використання. Для аналізу було обрано декілька рішень, які найбільш наближені до теми роботи: Blender, MeshLab та Autodesk Meshmixer.

Blender є одним із найвідоміших безкоштовних програмних комплексів для роботи з тривимірною графікою. Він використовується для моделювання, анімації, рендерингу, скульптингу, створення візуальних ефектів та підготовки 3D-ресурсів. У контексті обробки полігональних сіток Blender надає широкий набір інструментів для редагування геометрії, згладжування, поділу поверхні, зменшення кількості полігонів та зміни структури сітки. Зокрема, модифікатор Decimate призначений для зменшення кількості вершин і граней моделі з мінімальними змінами її форми.

Перевагою Blender є його універсальність і велика кількість інструментів для роботи з 3D-моделями. Користувач може не лише виконувати обробку сітки,

а й створювати моделі з нуля, редагувати матеріали, налаштовувати освітлення, виконувати рендеринг та анімацію. Водночас саме універсальність Blender може бути недоліком для користувача, якому потрібна лише швидка автоматизована обробка топологічної сітки. Інтерфейс програми містить багато можливостей, які не стосуються безпосередньо задачі перебудови сітки, тому для початківця або для вузької задачі програма може бути надмірно складною.

MeshLab — це спеціалізована система з відкритим кодом, призначена для обробки та редагування трикутних 3D-сіток. Вона містить інструменти для редагування, очищення, відновлення, інспектування, рендерингу, текстурування та перетворення полігональних моделей. Також MeshLab використовується для обробки даних, отриманих у результаті 3D-оцифрування, і підготовки моделей до 3D-друку.

Сильною стороною MeshLab є орієнтація саме на роботу з полігональними сітками. Програма добре підходить для очищення моделей, спрощення геометрії, аналізу структури поверхні та виконання різних фільтрів обробки. Вона є корисною для технічних задач, де потрібно працювати з уже готовою сіткою, а не створювати модель з нуля. Недоліком MeshLab можна вважати менш інтуїтивний інтерфейс для новачків, а також те, що велика кількість фільтрів і параметрів може ускладнювати швидке виконання простої задачі.

Autodesk Meshmixer — це інструмент для роботи з полігональними моделями, який часто використовувався для редагування STL-моделей, підготовки до 3D-друку, виправлення дефектів сітки та простого моделювання. Серед типових можливостей Meshmixer можна виділити аналіз і виправлення пошкоджених сіток, роботу з отворами, підтримками та підготовку моделей до друку. За наявною інформацією, активний розвиток Meshmixer припинено.

Перевагою Meshmixer є його зручність для окремих практичних задач, особливо пов’язаних із 3D-друком і виправленням моделей. Програма дозволяє швидко виконувати базове редагування сітки та аналізувати модель перед друком. Недоліком є те, що рішення вже не розвивається як сучасний

інструмент, а отже не є найкращою основою для довгострокового використання або орієнтації на нові робочі процеси.

Узагальнюючи огляд, можна зазначити, що кожне з розглянутих рішень має власну спеціалізацію. Blender є потужним універсальним середовищем для 3D-графіки, MeshLab орієнтований на технічну обробку полігональних сіток, а Meshmixer історично використовувався для редагування та підготовки моделей до 3D-друку. Проте жодне з цих рішень не є повністю орієнтованим саме на простий навчально-практичний сценарій, у якому користувач швидко завантажує модель, обирає один із кількох способів обробки топологічної сітки, задає параметри, переглядає результат і зберігає його для подальшого використання.

Саме тому розробка окремого програмного засобу для автоматизованої перебудови топологічної сітки є доцільною. Такий застосунок може бути простішим за універсальні 3D-редактори, більш сфокусованим на конкретній задачі та зручним для демонстрації різних підходів до обробки полігональної сітки.

1.4.3 Обґрунтування необхідності власної розробки. Проведений огляд існуючих програмних рішень показує, що на сьогодні існує багато інструментів для роботи з тривимірними моделями та полігональними сітками. Частина з них є універсальними 3D-редакторами, які охоплюють широкий спектр задач: моделювання, анімацію, текстурування, рендеринг, скульптинг і підготовку моделей до експорту. Інші рішення мають більш вузьку спеціалізацію та орієнтовані на аналіз, очищення або перебудову структури сітки. Проте кожен із розглянутих інструментів має певні обмеження з погляду задачі, що розглядається у даній бакалаврській кваліфікаційній роботі.

Універсальні 3D-редактори, такі як Blender, мають велику кількість можливостей, однак через це можуть бути надмірно складними для користувача, якому потрібно виконати лише базову обробку топологічної сітки. Для виконання простої операції користувачеві часто потрібно орієнтуватися в значній кількості панелей, режимів роботи, модифікаторів і налаштувань. Це

робить такі програми потужними, але не завжди зручними для швидкого виконання вузькоспеціалізованої задачі.

Спеціалізовані рішення для обробки полігональних сіток, наприклад MeshLab, краще відповідають задачам аналізу та зміни геометричної структури моделі. Водночас вони також мають свої обмеження. MeshLab містить багато фільтрів і параметрів, що може ускладнювати роботу для користувачів без достатнього досвіду.

Окремі інструменти, такі як Autodesk Meshmixer, історично використовувалися для підготовки моделей до 3D-друку та виправлення дефектів сітки. Проте такі рішення не завжди є актуальними з погляду подальшого розвитку, підтримки та використання у сучасних робочих процесах. Крім того, їхня функціональність може бути більше спрямована на конкретну прикладну сферу, наприклад 3D-друк, а не на демонстрацію та дослідження різних способів обробки топологічної сітки.

Необхідність власної розробки зумовлена потребою у створенні простого й цілеспрямованого програмного засобу, який буде зосереджений саме на автоматизованій перебудові топологічної сітки тривимірної моделі. Такий додаток не повинен замінювати повноцінні 3D-редактори, а має виконувати конкретну задачу: надати користувачу можливість завантажити модель, переглянути її, обрати спосіб обробки, задати параметри, отримати результат і зберегти оброблену модель.

Власна розробка також є доцільною з навчальної та дослідницької точки зору. Вона дозволяє не просто використати готові алгоритми, а краще зрозуміти процес роботи з полігональною геометрією: структуру 3D-моделі, особливості її сітки, вплив параметрів на результат обробки, зміну кількості вершин і граней, а також можливі проблеми, що виникають під час перебудови топології. Це робить систему корисною не лише як практичний інструмент, а й як засіб демонстрації принципів обробки 3D-моделей.

Важливою перевагою власної розробки є можливість орієнтувати її саме на поставлену задачу дипломної роботи. У межах такого програмного засобу

можна передбачити лише необхідні для користувача функції, не перевантажуючи інтерфейс зайвими можливостями. Основна увага приділяється роботі з моделлю, вибору способу обробки, налаштуванню параметрів і візуальній оцінці результату. Завдяки цьому користувач отримує більш зрозумілий і послідовний сценарій роботи.

Таким чином, аналіз існуючих програмних рішень показав, що наявні інструменти або є занадто універсальними й складними, або мають вузьку спеціалізацію, яка не повністю відповідає поставленій задачі. Тому розробка окремого програмного забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі є обґрунтованою. Такий програмний засіб дозволить поєднати простий сценарій роботи, наочне представлення результату та можливість застосування декількох підходів до обробки полігональної сітки в межах одного додатку.

1.5 Постановка завдання

На основі проведеного аналізу предметної області, визначення вимог та огляду існуючих програмних рішень сформульовано завдання на розробку програмного забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі. У попередніх підрозділах було встановлено, що робота з полігональними 3D-моделями часто потребує зміни структури сітки, зменшення або збільшення деталізації, вирівнювання елементів поверхні та підготовки моделі до подальшого використання.

1.5.1 Формулювання загальної задачі. Необхідно розробити настільний програмний застосунок, призначений для роботи з полігональними тривимірними моделями. Система повинна забезпечувати завантаження 3D-моделі з файлу, її візуальний перегляд, вибір способу обробки топологічної сітки, задання параметрів обробки, виконання перебудови сітки, перегляд отриманого результату та збереження обробленої моделі.

Основною задачею розробки є створення засобу, який дозволяє користувачу виконувати автоматизовану зміну структури полігональної сітки без необхідності ручного редагування окремих вершин, ребер або граней. Програмний засіб має бути орієнтований на базову підготовку 3D-моделі до подальшого використання в інших середовищах, а також на демонстрацію різних підходів до обробки сітки.

У межах роботи необхідно передбачити кілька способів обробки топологічної сітки: спрощення моделі, збільшення деталізації та вирівнювання структури сітки. Кожен спосіб повинен мати параметри, які користувач може змінювати перед запуском обробки. Після виконання операції користувач повинен мати змогу оцінити результат візуально та зберегти його у файл.

1.5.2 Вхідні дані. Вхідними даними для розроблюваної системи є тривимірна полігональна модель і параметри, які визначають спосіб її обробки. Модель надходить у вигляді файлу, що містить геометричний опис поверхні об’єкта. До такого опису можуть входити координати вершин, зв’язки між вершинами, грані, нормалі та інші дані, необхідні для представлення форми об’єкта.

До вхідних даних також належать параметри обробки, які задаються користувачем. Вони впливають на результат перебудови топологічної сітки. Наприклад, для спрощення сітки важливим є ступінь зменшення кількості полігонів, для поділу — кількість рівнів деталізації, а для вирівнювання структури — бажаний розмір елементів сітки та кількість повторень процесу.

1.5.3 Вихідні дані. Вихідними даними системи є оброблена тривимірна модель, структура сітки якої була змінена відповідно до вибраного способу обробки та заданих параметрів. Результат повинен бути доступний користувачу для візуального перегляду одразу після завершення операції. Це дозволяє оцінити, чи відповідає отримана модель очікуваному результату.

Оброблена модель повинна зберігати загальну форму початкового об’єкта, якщо обраний спосіб обробки не передбачає істотної зміни геометрії. Після

завершення роботи користувач повинен мати можливість експортувати результат у файл для подальшого використання в інших програмах.

Важливою частиною вихідних даних є не лише сам файл моделі, а й можливість оцінити зміни, що відбулися після обробки. До таких змін належать кількість полігонів, щільність сітки, рівномірність розподілу елементів і візуальна якість поверхні.

1.5.4 Сценарій використання. Основний сценарій використання системи описує типовий процес роботи користувача з тривимірною моделлю. Спочатку користувач запускає програму та обирає файл моделі для завантаження. Після успішного завантаження модель відображається у вікні перегляду. Користувач може оглянути її форму, змінити ракурс, наблизити або віддалити модель, а також оцінити структуру сітки.

Після первинного перегляду користувач обирає спосіб обробки топологічної сітки. Якщо модель має надмірну кількість полігонів, може бути обране спрощення. Якщо потрібно підвищити деталізацію, обирається поділ сітки. Якщо необхідно зробити структуру більш рівномірною, використовується вирівнювання сітки. Далі користувач задає параметри відповідного способу обробки та запускає процес.

Після завершення перебудови топологічної сітки модель оновлюється для перегляду. Користувач оцінює отриманий результат. Якщо результат є задовільним, модель експортується у файл або зберігається як частина проєкту. Якщо результат не відповідає очікуванням, користувач може змінити параметри або обрати інший спосіб обробки та повторити процес.

Основний сценарій можна подати у вигляді послідовності кроків:

1. Користувач запускає програмний застосунок.
2. Користувач обирає файл тривимірної моделі у форматі OBJ.
3. Модель завантажується та стає доступною для перегляду.
4. Користувач оцінює форму моделі та структуру її сітки.
5. Користувач обирає спосіб обробки топологічної сітки.
6. Користувач задає параметри обраного способу обробки.

7. Виконується перебудова топологічної сітки.
8. Оновлена модель відображається для перегляду.
9. Користувач оцінює результат.
10. У разі задовільного результату модель експортується або зберігається.
11. У разі незадовільного результату користувач змінює параметри та повторює обробку.

1.5.5 Кількісні критерії. Для конкретизації поставленого завдання необхідно визначити кількісні критерії, за якими можна оцінити відповідність розробленої системи очікуваним вимогам. Ці критерії зазначені у таблиці 1.3. Такі критерії дозволяють уникнути нечітких формулювань та замінити їх вимірюваними показниками.

Таблиця 1.3

Кількісні критерії виконання завдання

Показник	**Критерій**
Підтримуваний формат імпорту	OBJ
Підтримуваний формат експорту	OBJ
Рекомендований максимальний розмір моделі	До 100 000 полігонів
Час завантаження моделі	До 2 секунд для моделі до 100 000 полігонів
Час оновлення відображення після завантаження	До 1 секунди
Час виконання спрощення сітки	До 5 секунд для моделі до 100 000 полігонів
Час виконання поділу сітки	До 5 секунд для одного рівня поділу на моделі до 50 000 полігонів
Час виконання вирівнювання структури сітки	До 10 секунд для моделі до 50 000 полігонів
Час експорту моделі	До 1 секунд для моделі до 100 000 полігонів

Зазначені критерії є орієнтовними для моделей середньої складності. Фактичний час виконання обробки може залежати від кількості полігонів, складності поверхні, якості початкової сітки та заданих параметрів. Особливо

це стосується операцій, пов’язаних із вирівнюванням структури сітки, оскільки вони можуть потребувати більшої кількості обчислень.

1.5.6 Критерії успішності роботи. Розробка вважається успішною, якщо програмна система забезпечує повний базовий цикл роботи з тривимірною моделлю: завантаження, перегляд, вибір способу обробки, задання параметрів, перебудову топологічної сітки, перегляд результату та збереження отриманої моделі. Крім того, система повинна коректно реагувати на типові помилки та не завершувати роботу аварійно.

Критерії успішності:

1. Модель успішно завантажується та відображається.
2. Користувач може переглядати модель, змінювати ракурс і масштаб.
3. Доступні три способи обробки топологічної сітки.
4. Користувач може задавати параметри для кожного способу обробки.
5. Після обробки змінюється структура сітки або кількість її елементів.
6. Оброблена модель зберігає загальну форму початкового об’єкта.
7. Користувач може переглянути оновлену модель після обробки.
8. Результат можна експортувати у файл.
9. Експортований файл відкривається у сторонньому 3D-редакторі.
10. Дані про проєкт і параметри обробки можуть бути збережені.
11. У разі помилки користувач отримує повідомлення про проблему.
12. Програма не завершує роботу аварійно під час типових помилкових дій.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Для забезпечення збереження інформації про роботу з тривимірними моделями у програмній системі необхідно спроєктувати логічну модель даних. Логічна модель даних описує основні інформаційні сутності системи, їхні атрибути та зв’язки між ними без прив’язки до конкретної фізичної реалізації бази даних. Вона є проміжним етапом між аналізом предметної області та створенням фізичної структури інформаційної бази.

У межах розроблюваного програмного забезпечення необхідно зберігати інформацію про проєкт користувача, файл тривимірної моделі, саму модель як об’єкт обробки, а також параметри, які використовуються під час перебудови топологічної сітки. Такий підхід дозволяє не обмежуватися лише відкриттям і збереженням окремого 3D-файлу, а працювати з поняттям проєкту, у якому поєднуються дані про модель, її джерело та налаштування обробки.

Основними сутностями логічної моделі даних є:

- Project_file — проєкт користувача;
- File_3D-Model — файл тривимірної моделі;
- 3DModel — тривимірна модель;
- Remesh_Parameters — параметри перебудови топологічної сітки.

Сутність Project_file призначена для збереження загальної інформації про проєкт. Вона містить назву проєкту, шлях до файлу проєкту, дату створення, дату оновлення та версію. Проєкт є головною сутністю інформаційної моделі, оскільки саме навколо нього організовується робота користувача з моделлю.

Сутність File_3D-Model описує файл, з якого була імпортована або до якого може бути експортована тривимірна модель. Вона містить назву файлу, шлях до нього та тип файлу. Виділення цієї сутності є доцільним, оскільки файл моделі

та сама модель не є повністю тотожними поняттями. Файл є зовнішнім носієм даних, тоді як модель є об’єктом, що має власні характеристики, наприклад кількість вершин і граней.

Сутність 3D-Model описує тривимірну модель як об’єкт, з яким працює користувач. Вона містить назву моделі, кількість вершин і кількість граней. Ці показники важливі для аналізу результатів обробки, оскільки після застосування алгоритмів перебудови топологічної сітки кількість елементів моделі може змінюватися.

Сутність Remesh_Parameters призначена для збереження параметрів обробки моделі. До таких параметрів належать назва обраного алгоритму, цільова довжина ребра, кількість рівнів поділу та коефіцієнт спрощення. Збереження цих даних дозволяє фіксувати, з якими налаштуваннями користувач виконував обробку моделі в межах конкретного проєкту. На рисунку 5 зображено ER-діаграму із зазначеними сутностями.

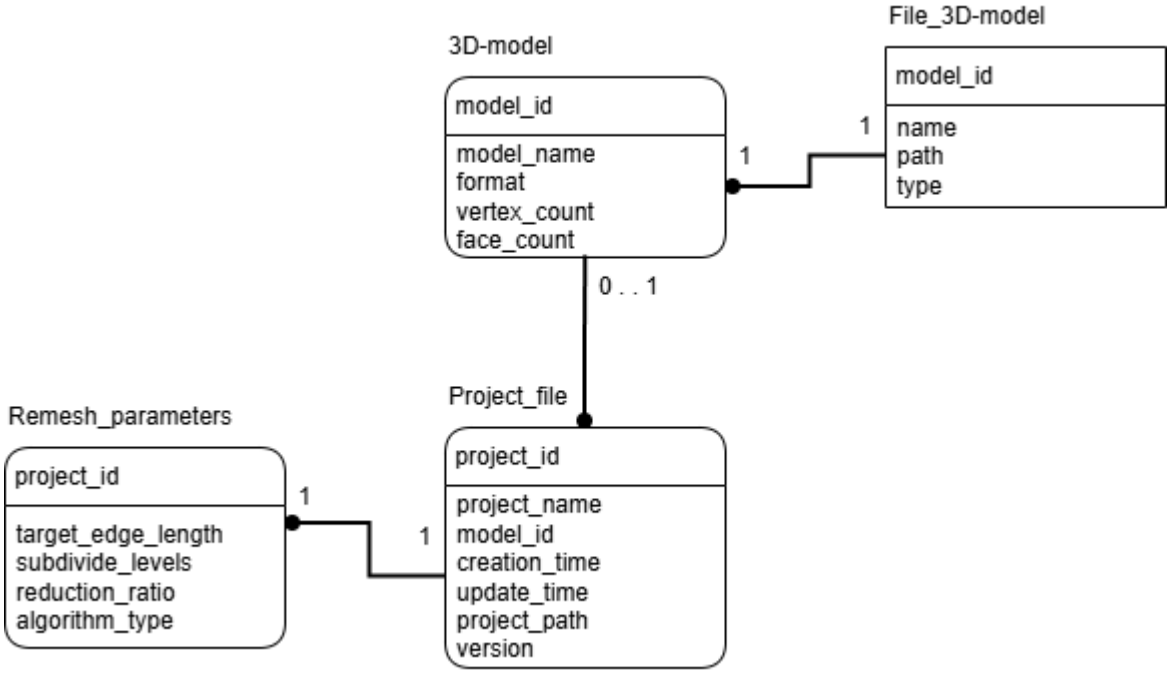


Рис. 5 Логічна модель даних у вигляді ER-діаграми

Побудована логічна модель даних має реляційну структуру, оскільки всі інформаційні об’єкти предметної області подані у вигляді окремих сутностей, які в подальшому можуть бути перетворені на таблиці реляційної бази даних.

Кожна сутність має власний набір атрибутів, первинний ключ та логічні зв’язки з іншими сутностями.

Сутність Project_file має первинний ключ project_id, який однозначно ідентифікує кожен проєкт. Сутність 3D-model має первинний ключ model_id, що використовується для ідентифікації конкретної тривимірної моделі. Сутність File_3D-model також пов’язана з моделлю через атрибут model_id, що дозволяє встановити відповідність між моделлю та її файловим поданням. Сутність Remesh_parameters має ключ project_id, який одночасно визначає належність параметрів до конкретного проєкту. Таким чином, кожен запис у кожній сутності може бути однозначно ідентифікований.

Модель відповідає вимогам першої нормальної форми, оскільки всі атрибути сутностей є атомарними, тобто не містять списків або повторюваних груп значень. Наприклад, назва проєкту, шлях до файлу, тип файлу, кількість вершин, кількість граней, цільова довжина ребра та коефіцієнт спрощення зберігаються як окремі значення. Це дозволяє представити дані у вигляді таблиць, де кожна комірка містить одне неподільне значення.

Модель відповідає вимогам другої нормальної форми, оскільки всі неключові атрибути повністю залежать від первинного ключа відповідної сутності. Наприклад, у сутності Project_file атрибути project_name, creation_time, update_time, project_path і version залежать від project_id, а не від будь-якого іншого атрибута. У сутності 3D-model атрибути model_name, format, vertex_count і face_count залежать від model_id.

Модель також відповідає вимогам третьої нормальної форми, оскільки неключові атрибути не залежать один від одного транзитивно. Дані, що описують різні об’єкти, винесені в окремі сутності. Наприклад, інформація про файл моделі не зберігається безпосередньо в сутності Project_file, а винесена в окрему сутність File_3D-model. Характеристики тривимірної моделі зберігаються в сутності 3D-model, а параметри обробки — у сутності Remesh_parameters. Завдяки цьому зменшується дублювання даних і спрощується підтримка їхньої цілісності.

Зв’язок між Project_file і 3D-model дозволяє зберігати у проєкті посилання на одну модель або залишати проєкт без моделі на початковому етапі. Зв’язок між 3D-model і File_3D-model відокремлює логічне представлення моделі від її файлового розташування. Зв’язок між Project_file і Remesh_parameters забезпечує збереження параметрів перебудови топологічної сітки для конкретного проєкту. Такий поділ сутностей є доцільним, оскільки кожна таблиця відповідає одному типу об’єктів і не містить надлишкових груп атрибутів.

Отже, представлена ER-діаграма описує реляційну логічну модель даних, яка може бути реалізована у реляційній СУБД. Модель відповідає вимогам реляційності, оскільки складається з окремих сутностей із визначеними ключами та зв’язками між ними. Крім того, вона відповідає третій нормальній формі, оскільки атрибути є атомарними, не мають часткових залежностей від ключів і не утворюють транзитивних залежностей між неключовими атрибутами.

2.2 Діаграма класів та кооперацій

2.2.1 Діаграма класів. На початковому етапі проєктування прикладного програмного забезпечення було побудовано загальну діаграму класів програмної системи. Вона відображає основні класи додатку, їхні атрибути, методи, асоціації між ними та потужності зв’язків. Ця діаграма описує саме структуру програмної системи та взаємодію її основних програмних частин. Діаграма класів зображена на рисунку 6.

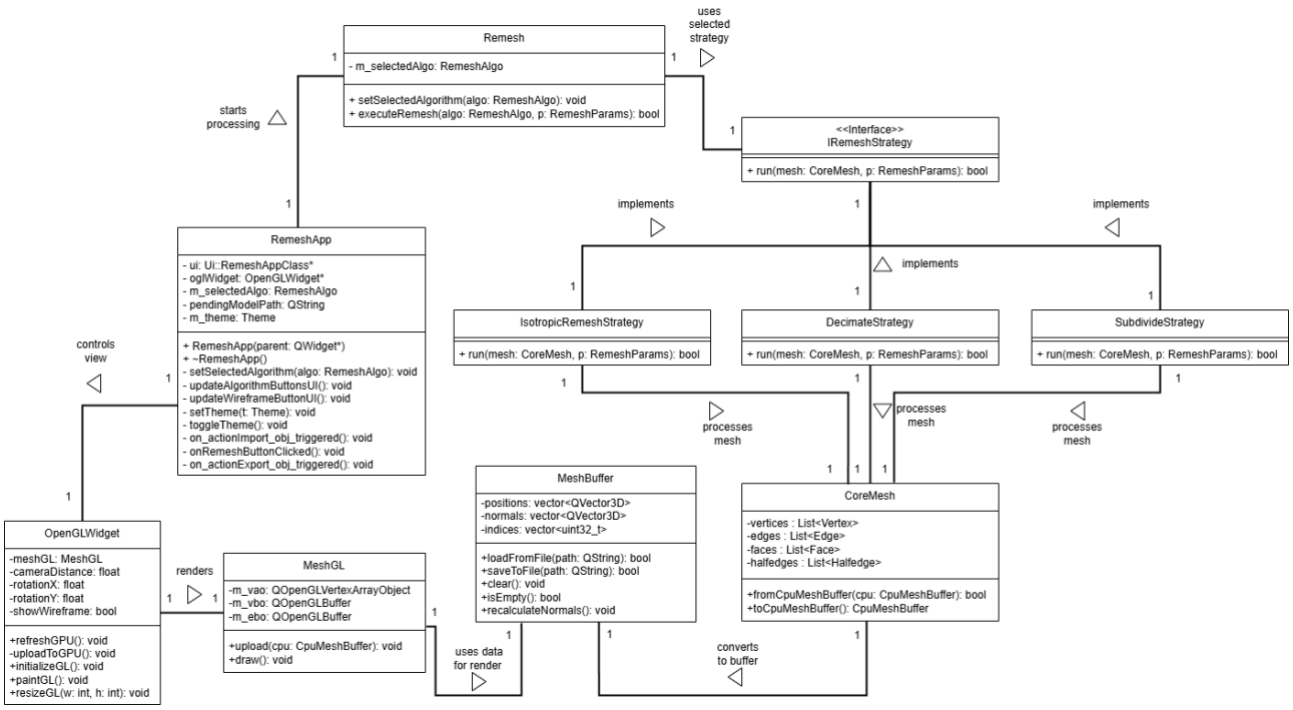


Рис. 6 Діаграма класів програмної системи

На діаграмі виділено класи, що відповідають за взаємодію з користувачем, візуалізацію моделі, зберігання геометричних даних, передавання даних до графічного представлення та виконання алгоритмів перебудови топологічної сітки. Основними класами системи є RemeshApp, Remesh, OpenGLWidget, MeshGL, MeshBuffer, CoreMesh, IRemeshStrategy, IsotropicRemeshStrategy, DecimateStrategy та SubdivideStrategy.

Клас RemeshApp є головним класом програмної системи, який відповідає за взаємодію користувача з додатком. Він містить посилання на об’єкт графічного інтерфейсу, віджет відображення моделі, інформацію про вибраний алгоритм, шлях до моделі, яка очікує завантаження, а також поточну тему інтерфейсу. До основних методів класу належать методи вибору алгоритму, оновлення кнопок інтерфейсу, перемикання теми, імпорту моделі, запуску перебудови сітки та експорту результату.

Клас Remesh виконує роль керівника процесу перебудови топологічної сітки. Він зберігає інформацію про вибраний алгоритм і надає методи для встановлення поточної стратегії та запуску обробки. Зв’язок між RemeshApp і Remesh має потужність 1 : 1 та підпис starts processing, оскільки одне головне

вікно використовує один об’єкт керування процесом обробки для запуску перебудови сітки.

Клас OpenGLWidget відповідає за відображення тривимірної моделі та керування її переглядом. Він містить параметри камери, кути обертання моделі, ознаку показу каркасного режиму та об’єкт MeshGL, який використовується для графічного представлення. Також клас містить методи оновлення графічних даних, завантаження їх у графічну пам’ять, ініціалізації графічного середовища, відмальовування сцени та зміни розміру області перегляду. Зв’язок RemeshApp — OpenGLWidget має потужність 1 : 1 і підпис controls view, оскільки головне вікно керує одним вікном перегляду моделі.

Клас MeshGL відповідає за графічне представлення моделі. Він містить об’єкти буферів, необхідних для зберігання геометричних даних у форматі, придатному для відображення. До його основних методів належать upload() і draw(), які забезпечують завантаження даних і відмальовування моделі. Зв’язок між OpenGLWidget і MeshGL має потужність 1 : 1 та підпис renders, оскільки один віджет відображення використовує один об’єкт графічного представлення для виведення моделі на екран.

Клас MeshBuffer є проміжним контейнером геометричних даних. Він містить списки позицій вершин, нормалей та індексів граней. Таке представлення використовується для передавання даних між внутрішньою моделлю та графічним представленням. Серед основних методів класу можна виділити завантаження даних із файлу, збереження у файл, очищення буфера, перевірку на порожність і перерахунок нормалей. Зв’язок MeshGL — MeshBuffer має потужність 1 : 1 та підпис uses data for render, оскільки графічне представлення використовує буфер як джерело даних для відображення.

Клас CoreMesh відповідає за внутрішнє представлення полігональної сітки. У діаграмі він містить списки вершин, ребер, граней та напівребер, які описують структуру тривимірної моделі. Основними методами класу є перетворення даних із MeshBuffer у внутрішнє представлення та зворотне перетворення у буфер. Зв’язок CoreMesh — MeshBuffer має потужність 1 : 1 і

підпис converts to buffer, оскільки внутрішнє представлення моделі може бути перетворене у буфер даних для подальшого відображення.

Інтерфейс IRemeshStrategy визначає загальний спосіб взаємодії з алгоритмами перебудови топологічної сітки. Він містить метод run(), який приймає об’єкт CoreMesh і параметри обробки. Зв’язок Remesh — IRemeshStrategy має потужність 1 : 1 та підпис uses selected strategy, оскільки під час одного запуску обробки використовується одна вибрана стратегія.

Класи IsotropicRemeshStrategy, DecimateStrategy та SubdivideStrategy представляють конкретні способи перебудови топологічної сітки. Клас IsotropicRemeshStrategy відповідає за вирівнювання структури сітки відповідно до заданих параметрів. Клас DecimateStrategy виконує спрощення сітки шляхом зменшення кількості полігонів. Клас SubdivideStrategy відповідає за збільшення деталізації шляхом поділу елементів сітки. Усі три класи пов’язані з IRemeshStrategy зв’язком із підписом implements, оскільки вони реалізують спільний інтерфейс виконання алгоритму.

Усі основні асоціації на діаграмі мають потужність 1 : 1, оскільки в межах одного активного сценарію роботи програми використовується один головний екземпляр вікна, один віджет перегляду, одна поточна модель, один буфер геометричних даних, один об’єкт графічного представлення та одна вибрана стратегія обробки. Такий підхід спрощує структуру системи та відповідає логіці роботи настільного застосунку, який у поточній версії орієнтований на обробку однієї активної 3D-моделі.

Побудована діаграма класів також демонструє використання шаблону проєктування Strategy. Інтерфейс IRemeshStrategy задає спільний спосіб запуску алгоритму, а конкретні класи стратегій реалізують різні варіанти обробки. Завдяки цьому клас Remesh може запускати перебудову топологічної сітки без прив’язки до конкретного алгоритму. У майбутньому це дозволяє додати нові способи обробки сітки без суттєвої зміни загальної структури системи.

Отже, діаграма класів програмної системи відображає основні структурні елементи додатку та асоціації між ними. Вона показує, що система поділена на

частини, відповідальні за керування діями користувача, візуалізацію, внутрішнє представлення геометрії, передавання даних для відображення та виконання алгоритмів перебудови топологічної сітки. Побудована діаграма є основою для подальшого розбиття системи на окремі кооперації класів.

2.2.2 Прості кооперації. Перша важлива кооперація класів пов’язана з імпортом і відображенням тривимірної моделі. Цю кооперацію зображено на рисунку 7. У ній беруть участь класи OpenGLWidget, CoreMesh, CpuMeshBuffer та MeshGL.

Процес починається з того, що користувач через головне вікно обирає файл моделі. Після цього модель завантажується у віджет перегляду. Дані моделі перетворюються у внутрішнє представлення CoreMesh, після чого на його основі формується буфер CpuMeshBuffer. Далі цей буфер передається до MeshGL, який завантажує геометричні дані у графічні буфери та забезпечує відображення моделі у вікні перегляду.

Таким чином, у цій кооперації OpenGLWidget координує відображення, CoreMesh відповідає за геометричну структуру, CpuMeshBuffer виконує роль проміжного представлення, а MeshGL забезпечує графічне виведення моделі.

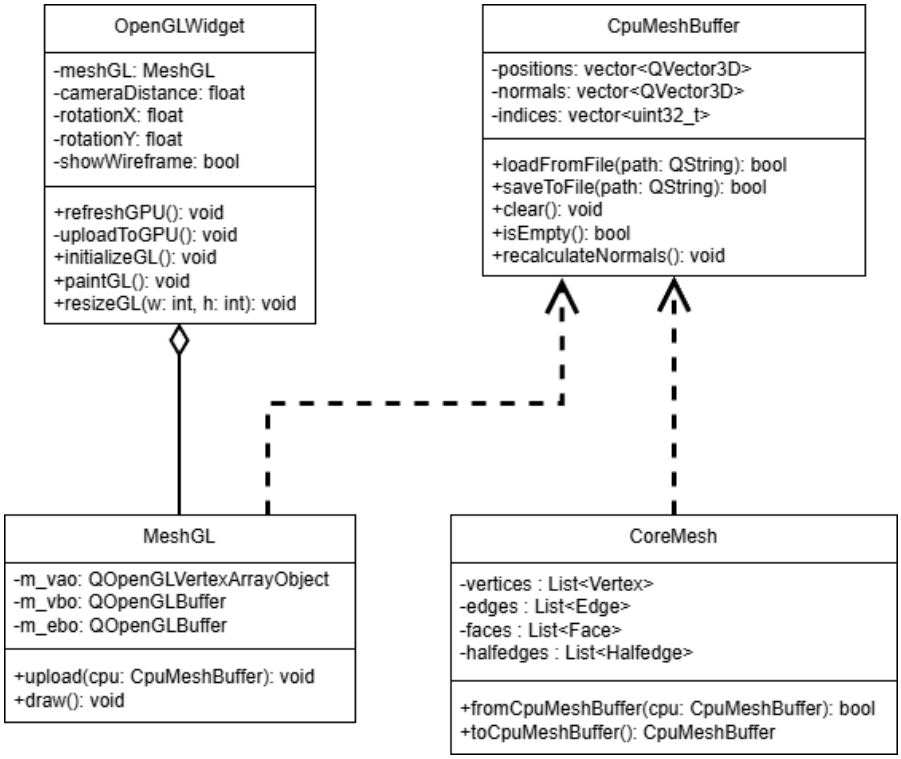


Рис. 7 Кооперація імпорту та відображення моделі

Друга кооперація описує процес виконання перебудови топологічної сітки. Цю кооперацію можна переглянути на рисунку 8. У ній беруть участь Remesh, IRemeshStrategy, CoreMesh та конкретні класи алгоритмів: IsotropicRemeshStrategy, DecimateStrategy, SubdivideStrategy.

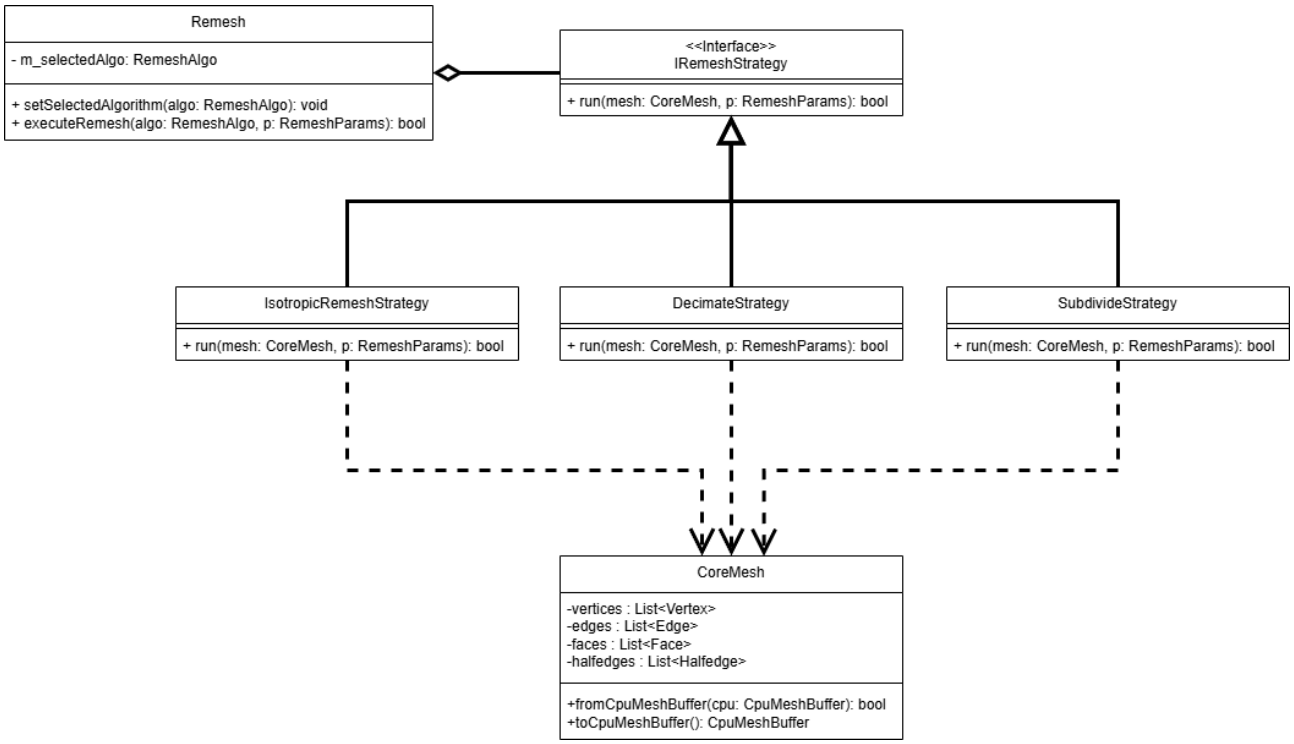


Рис. 8 Кооперація перебудови топологічної сітки

Користувач обирає алгоритм обробки та задає параметри. Після натискання кнопки клас керування перебудовою визначає, яку саме стратегію потрібно використати. Вибрана стратегія отримує посилання на CoreMesh і змінює структуру сітки відповідно до заданих параметрів.

Після завершення обробки змінена модель знову передається до кооперації відображення: з CoreMesh формується оновлений CpuMeshBuffer, який завантажується у MeshGL для візуалізації результату. Такий підхід дозволяє розділити відповідальність між класами: одні класи відповідають за вибір і запуск алгоритму, інші — за зберігання геометрії, а окремі класи — за відображення результату.

2.3 Діаграма пакетів

Для відображення загальної архітектурної організації програмної системи було побудовано діаграму пакетів. Діаграма пакетів дозволяє показати, як класи програмного забезпечення згруповані за функціональним призначенням, а також які залежності існують між окремими частинами системи. Цю діаграму зображено на рисунку 9.

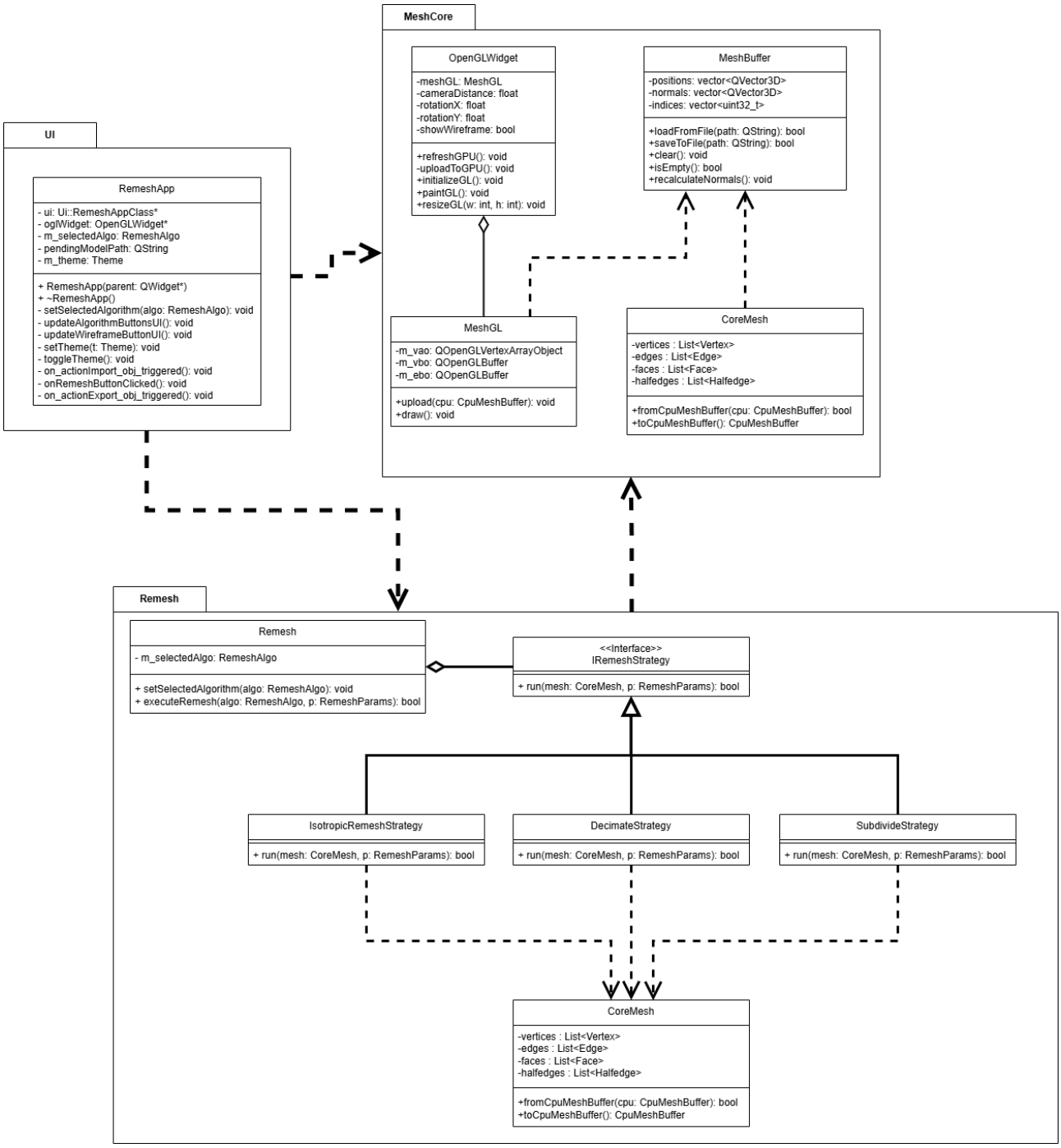


Рис. 9 Діаграма пакетів програмної системи

На діаграмі виділено три основні пакети: UI, MeshCore та Remesh. Такий поділ відповідає загальній логіці роботи програмного застосунку: користувач взаємодіє з інтерфейсом, інтерфейс звертається до ядра роботи з моделлю та запускає алгоритми перебудови топологічної сітки, а після завершення обробки оновлена модель знову передається для відображення.

Пакет UI містить клас RemeshApp, який відповідає за головне вікно програми та взаємодію користувача з основними функціями системи. Пакет UI є верхнім рівнем системи, оскільки саме він координує дії користувача та передає запити до інших частин програми.

Пакет MeshCore об’єднує класи, пов’язані з представленням, підготовкою та відображенням тривимірної моделі. До нього входять OpenGLWidget, MeshGL, MeshBuffer та CoreMesh.

Пакет Remesh містить класи, які відповідають за виконання алгоритмів перебудови топологічної сітки. До нього входять клас Remesh, інтерфейс IRemeshStrategy та конкретні реалізації алгоритмів: IsotropicRemeshStrategy, DecimateStrategy і SubdivideStrategy.

Залежності між пакетами на діаграмі показані пунктирними стрілками. Пакет UI залежить від пакета MeshCore, оскільки головне вікно програми використовує OpenGLWidget для відображення моделі та отримання доступу до поточної геометрії. Також пакет UI залежить від пакета Remesh, оскільки саме з інтерфейсу користувача запускається процес обробки сітки та передаються параметри вибраного алгоритму.

Пакет Remesh залежить від пакета MeshCore, оскільки алгоритми перебудови топологічної сітки працюють із класом CoreMesh. Вони отримують поточну сітку, змінюють її структуру відповідно до вибраного алгоритму, а після завершення обробки оновлена модель знову може бути перетворена в буфер для візуалізації. Таким чином, CoreMesh є спільною точкою взаємодії між алгоритмічною частиною системи та частиною, що відповідає за відображення моделі.

Отже, діаграма пакетів демонструє логічний поділ програмної системи на три основні частини: інтерфейс користувача, ядро роботи з моделлю та підсистему алгоритмів перебудови топологічної сітки. Така структура робить архітектуру програми більш зрозумілою, спрощує супровід коду та створює можливість для подальшого розширення функціональності системи.

2.4 Діаграма компонентів

Діаграма компонентів використовується для відображення фізичної та логічної структури програмної системи на рівні її складових частин. На відміну від діаграми класів, вона не деталізує внутрішню будову окремих класів, а показує, з яких компонентів складається програмний продукт, які підсистеми в ньому виділено та через які інтерфейси вони взаємодіють між собою. На рисунку 10 зображено діаграму компонентів для розроблюваної системи.

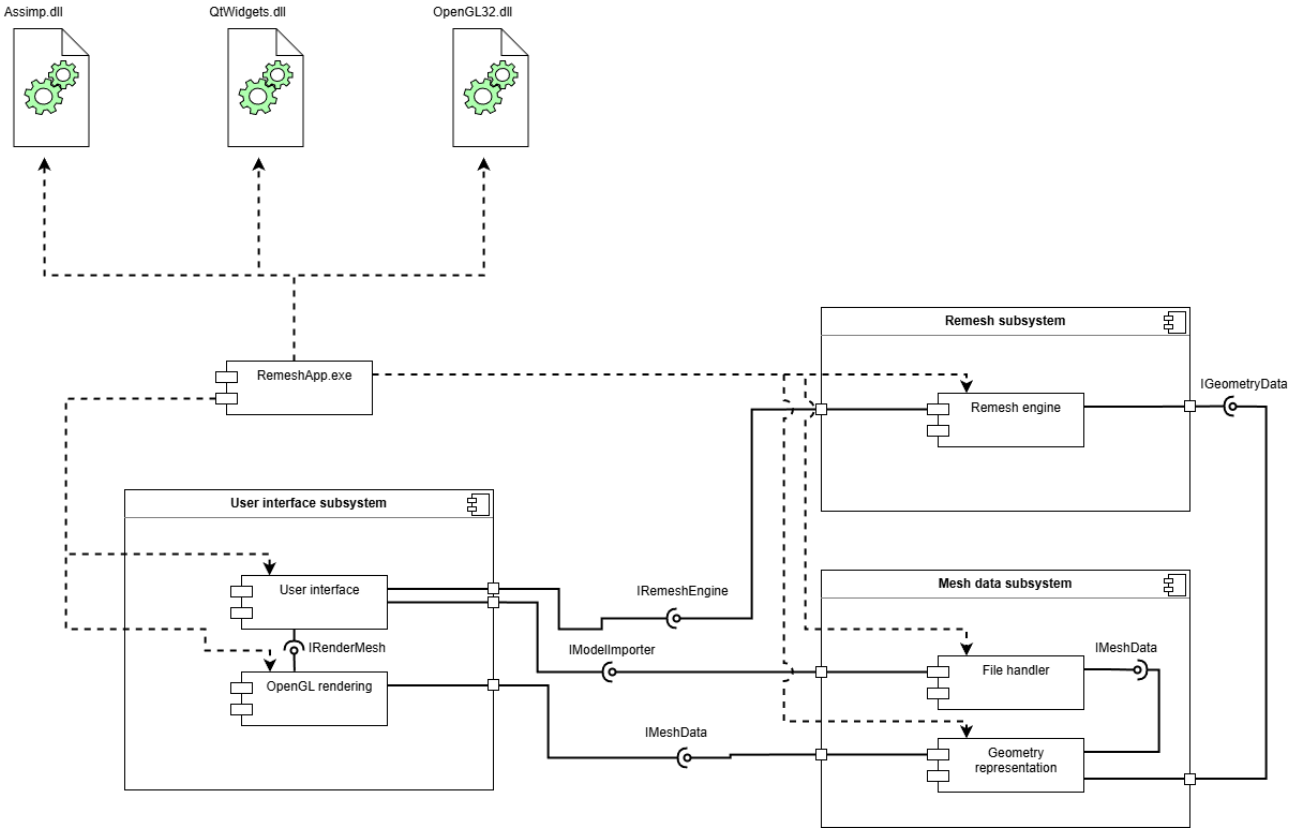


Рис. 10 Діаграма компонентів програмної системи

У розроблюваній програмній системі основним виконуваним компонентом є RemeshApp.exe. Він об’єднує основні частини застосунку та забезпечує запуск усієї системи. Для роботи програми також використовуються зовнішні динамічні бібліотеки, зокрема QtWidgets.dll, OpenGL32.dll та Assimp.dll. Бібліотека QtWidgets використовується для побудови графічного інтерфейсу, OpenGL32 — для відображення тривимірної сцени, а Assimp — для роботи з файлами 3D-моделей.

На діаграмі виділено три основні підсистеми: User interface subsystem, Remesh subsystem та Mesh data subsystem.

Підсистема User interface subsystem відповідає за взаємодію користувача з програмою. До її складу входить компонент User interface, який забезпечує доступ до основних дій: імпорту моделі, вибору способу обробки, налаштування параметрів, запуску перебудови топологічної сітки та експорту результату. Також у межах цієї підсистеми розміщено компонент OpenGL rendering, який відповідає за візуальне відображення тривимірної моделі та оновлення сцени після зміни геометрії.

Підсистема Mesh data subsystem відповідає за роботу з геометричними даними моделі. Вона містить компонент Model file handler, який забезпечує отримання даних із файлу моделі та збереження результату. Також до цієї підсистеми входить компонент Geometry representation, який відповідає за внутрішнє представлення полігональної сітки, зберігання вершин, граней, нормалей та інших геометричних даних, необхідних для візуалізації та обробки.

Підсистема Remesh subsystem відповідає за алгоритмічну обробку моделі. Основним її компонентом є Remesh engine, який отримує геометричні дані моделі, параметри обробки та виконує перебудову топологічної сітки відповідно до обраного способу. Результатом роботи цієї підсистеми є оновлена геометрична структура моделі, яка надалі передається для відображення та збереження.

Таким чином, діаграма компонентів показує розподіл програмної системи на функціональні частини та демонструє, як саме ці частини взаємодіють між

собою. Такий поділ дозволяє відокремити інтерфейс користувача, візуалізацію, роботу з геометричними даними та алгоритмічну обробку. Завдяки цьому система має більш зрозумілу структуру та може бути розширена в майбутньому, наприклад шляхом додавання нових алгоритмів перебудови топологічної сітки або підтримки нових форматів 3D-моделей.

2.5 Діаграма розгортання

Діаграма розгортання використовується для відображення фізичного розміщення програмної системи та її основних артефактів у середовищі виконання. Вона показує, на якому апаратному вузлі розгортається застосунок, які файли входять до складу системи та з якими зовнішніми або допоміжними компонентами взаємодіє виконуваний файл програми. На рисунку 11 зображено діаграму розгортання системи.

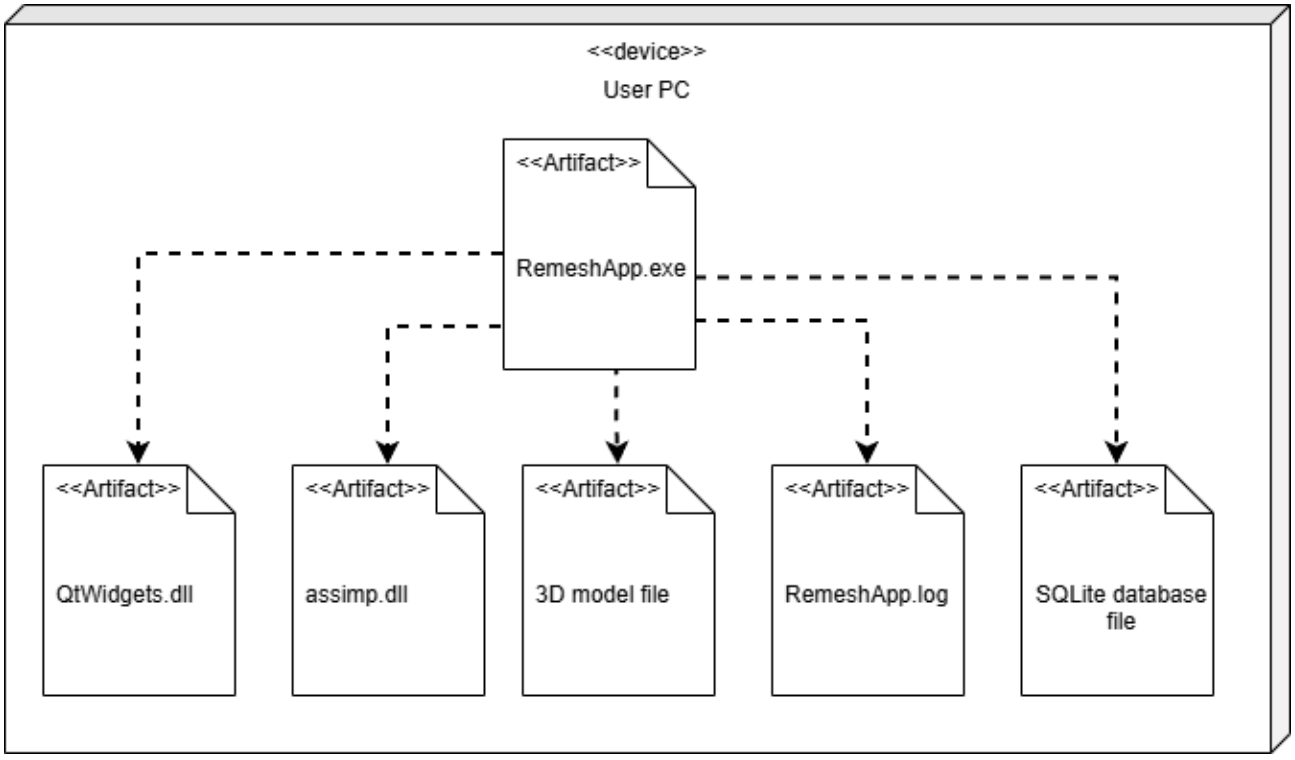


Рис. 11 Діаграма розгортання програмної системи

У розроблюваній системі розгортання виконується на одному фізичному вузлі — персональному комп’ютері користувача. Це відповідає архітектурі desktop-додатку, оскільки програма не потребує підключення до віддаленого сервера та виконує всі основні операції локально. На комп’ютері користувача

розміщується основний виконуваний файл RemeshApp.exe, який запускає програмну систему та координує роботу інтерфейсу, імпорту моделей, візуалізації, обробки сітки, збереження службових даних і формування журналу роботи.

На діаграмі показано вузол User PC, який позначає комп’ютер користувача. Усередині цього вузла розміщено основний артефакт RemeshApp.exe. Він є центральним елементом розгортання, оскільки саме через нього користувач взаємодіє з програмною системою. Від нього показано залежності до інших артефактів, необхідних для повноцінної роботи додатку.

Пунктирні стрілки на діаграмі показують залежності між основним виконуваним файлом RemeshApp.exe та іншими артефактами. Це означає, що для коректної роботи програма використовує відповідні бібліотеки, файли моделей, журнал подій і локальну базу даних. Усі ці компоненти розміщуються або використовуються на одному комп’ютері користувача, що підтверджує локальний характер розгортання системи.

Таким чином, діаграма розгортання демонструє, що програмна система має просту локальну архітектуру. Усі основні компоненти розміщуються на персональному комп’ютері користувача, а робота програми не залежить від мережевого з’єднання або серверної інфраструктури. Такий підхід є доцільним для розроблюваного desktop-додатку, оскільки забезпечує автономність, простоту встановлення та зручність експлуатації.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Під час розробки програмного забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі виникає потреба у збереженні інформації, яка описує стан роботи користувача з моделлю. Така інформація не обмежується лише самим файлом 3D-моделі, оскільки користувач також працює з проєктом, параметрами обробки, шляхами до файлів, характеристиками моделі та результатами виконаних дій. Для організації цих даних у системі передбачається використання інформаційної бази.

Інформаційна база в додатку виконує допоміжну, але важливу роль. Вона не замінює файл тривимірної моделі, а зберігає службову та проєктну інформацію, необхідну для відновлення стану роботи. До таких даних належать назва проєкту, шлях до файлу моделі, тип файлу, кількість вершин і граней, дата створення та оновлення проєкту, а також параметри перебудови топологічної сітки. Завдяки цьому користувач може не лише імпортувати й експортувати окрему модель, а й працювати з проєктом як з цілісною одиницею.

Важливою особливістю інформаційної бази є те, що збереження даних має виконуватися не постійно, а за дією користувача. Тобто інформація про проєкт, модель і параметри обробки записується після натискання відповідної кнопки збереження. Такий підхід є зручним для desktop-додатку, оскільки користувач сам контролює момент фіксації змін. При цьому база даних може використовуватися разом із файлом проєкту, який зберігає посилання на необхідні дані та дозволяє відкрити проєкт повторно.

У таблиці 3.1 показано основні вимоги, що висуваються до системи управління інформаційною базою для розроблюваного додатку.

Таблиця 3.1

Вимоги до системи управління інформаційною базою

№	Вимога до СУБД	Пояснення
1	Локальне зберігання даних	Додаток повинен працювати без необхідності підключення до віддаленого сервера
2	Простота інтеграції	СУБД має легко інтегруватися в desktop-застосунок
3	Невеликий обсяг службових даних	База використовується для збереження метаданих проєкту, а не великих 3D-файлів
4	Надійність збереження	Дані про проєкт і параметри повинні зберігатися у структурованому вигляді
5	Підтримка реляційної моделі	Необхідно забезпечити зв’язки між проєктом, моделлю, файлом і параметрами
6	Відсутність складного адміністрування	Користувач не повинен налаштовувати сервер або окрему службу бази даних
7	Портативність	Дані мають легко переноситися разом із проєктом або файлом бази
8	Достатня продуктивність	Операції збереження й читання невеликих обсягів даних мають виконуватися швидко

З урахуванням цих вимог для реалізації інформаційної бази було обрано SQLite. SQLite є вбудованою реляційною системою управління базами даних, яка зберігає всю базу в одному локальному файлі. Це добре відповідає характеру розроблюваного програмного забезпечення, оскільки додаток є настільним і не потребує багатокористувацького серверного доступу.

Однією з головних переваг SQLite є відсутність необхідності встановлення та налаштування окремого сервера бази даних. На відміну від серверних СУБД, таких як MySQL або PostgreSQL, SQLite працює без окремої служби, а база даних зберігається безпосередньо у файлі. Це спрощує розгортання програми на комп’ютері користувача та робить систему більш автономною.

SQLite також добре підходить для збереження невеликих і середніх обсягів структурованих даних. У межах розроблюваного додатку база даних не призначена для зберігання великих геометричних масивів самої 3D-моделі. Основні 3D-дані залишаються у файлі моделі, а в базі зберігаються відомості

про проєкт, шляхи до файлів, характеристики моделі та параметри обробки. Такий обсяг даних є невеликим, тому SQLite є достатнім і практичним рішенням.

Вибір SQLite також обґрунтований портативністю. Оскільки база даних зберігається в одному файлі, її можна легко переміщувати, копіювати або зберігати разом із файлом проєкту. Це зручно для користувача, який працює з 3D-моделями локально та може переносити проєкти між різними комп'ютерами.

Для даного програмного забезпечення використання потужних серверних СУБД є недоцільним. Такі системи краще підходять для веб-додатків, багатокористувацьких інформаційних систем або великих корпоративних баз даних. У цьому випадку додаток працює локально, з одним користувачем і невеликим обсягом службової інформації. Тому SQLite забезпечує потрібну функціональність без зайвої складності.

Таким чином, SQLite є оптимальним вибором для реалізації інформаційної бази розроблюваного desktop-додатку. Вона забезпечує локальне збереження структурованих даних, не потребує окремого сервера, підтримує реляційну модель, легко інтегрується в програму та відповідає вимогам до збереження інформації про проєкт, 3D-модель і параметри перебудови топологічної сітки.

3.2 Розробка інформаційної бази

Розробка інформаційної бази виконується на основі логічної ER-моделі, побудованої на етапі проєктування інформаційного забезпечення. Якщо логічна модель описує основні сутності, їхні атрибути та зв'язки на концептуальному рівні, то фізична модель визначає безпосередню структуру таблиць бази даних, типи даних, первинні та зовнішні ключі. У межах розроблюваного програмного забезпечення інформаційна база призначена для збереження даних про проєкт, файл 3D-моделі, характеристики моделі та параметри перебудови топологічної сітки.

Для реалізації інформаційної бази обрано SQLite, тому фізична модель орієнтована на використання типів даних, які підтримуються цією СУБД. У базі даних передбачено чотири основні таблиці: Project_file, 3D_model, File_3D_model та Remesh_parameters. Така структура відповідає логічній моделі даних і дозволяє зберігати службову інформацію про роботу користувача з моделлю. Зображення фізичної моделі бази даних показано на рисунку 12.

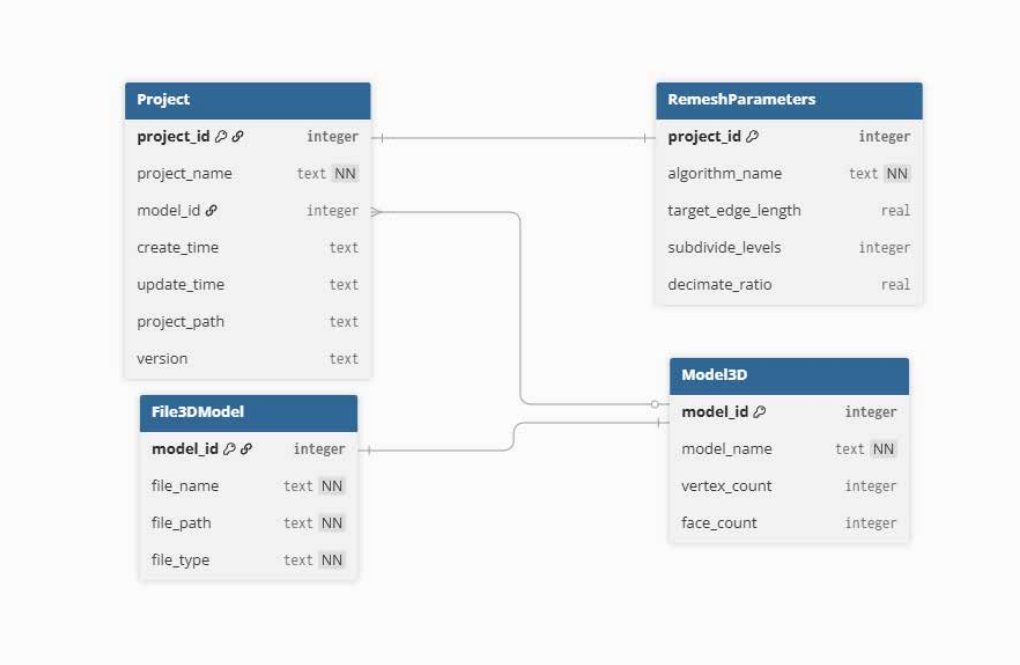


Рис. 12 Фізична модель бази даних програмної системи

На фізичній моделі бази даних відображено таблиці, їхні поля, ключові атрибути та зв’язки між таблицями. Центральною таблицею є Project_file, оскільки саме вона описує файл проєкту та пов’язує між собою модель і параметри її обробки.

Важливо, що в базі даних не передбачається зберігання повного набору геометричних даних моделі у вигляді масивів вершин і граней. Такі дані залишаються у файлі 3D-моделі, а база зберігає лише службову інформацію та параметри, необхідні для відновлення проєкту. Це дозволяє зменшити обсяг бази даних і спростити її структуру.

Для створення фізичної структури бази даних можна використати SQL-запити показані на рисунку 13.

```
CREATE TABLE IF NOT EXISTS "3D_model" (
    model_id INTEGER PRIMARY KEY,
    model_name TEXT NOT NULL,
    format TEXT,
    vertex_count INTEGER,
    face_count INTEGER
);
CREATE TABLE IF NOT EXISTS File_3D_model (
    model_id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    path TEXT NOT NULL,
    type TEXT,
    FOREIGN KEY (model_id) REFERENCES "3D_model"(model_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS Project_file (
    project_id INTEGER PRIMARY KEY,
    project_name TEXT NOT NULL,
    model_id INTEGER UNIQUE,
    creation_time TEXT,
    update_time TEXT,
    project_path TEXT,
    version TEXT,
    FOREIGN KEY (model_id) REFERENCES "3D_model"(model_id)
        ON UPDATE CASCADE
        ON DELETE SET NULL
);
CREATE TABLE IF NOT EXISTS Remesh_parameters (
    project_id INTEGER PRIMARY KEY,
    target_edge_length REAL,
    subdivide_levels INTEGER,
    reduction_ratio REAL,
    algorithm_type TEXT NOT NULL,
    FOREIGN KEY (project_id) REFERENCES Project_file(project_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

Рис. 13 Фрагмент коду для створення таблиць бази даних

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Для розробки програмного забезпечення автоматизованої перебудови топологічної сітки тривимірної моделі необхідно обрати інструментарій, який забезпечить створення графічного інтерфейсу, роботу з 3D-моделями, візуалізацію тривимірної сцени, виконання геометричної обробки та збереження службових даних про проєкт. Оскільки розроблюваний додаток є настільним застосунком для Windows, основними критеріями вибору інструментів були продуктивність, підтримка C++, можливість роботи з графікою, наявність готових бібліотек для 3D-геометрії та зручність інтеграції між окремими компонентами.

Для реалізації прикладного програмного забезпечення було обрано мову програмування C++. Ця мова добре підходить для задач, пов’язаних із комп’ютерною графікою та обробкою геометричних даних, оскільки забезпечує високу продуктивність, контроль над пам’яттю та доступ до широкого набору бібліотек. У межах даної роботи C++ використовується для реалізації основної логіки програми, обробки 3D-моделі, роботи з параметрами алгоритмів та взаємодії між класами системи.

Для створення графічного інтерфейсу користувача було обрано фреймворк Qt. Він надає засоби для розробки desktop-додатків, створення головного вікна, меню, кнопок, полів введення, перемикачів теми та інших елементів інтерфейсу. Використання Qt є доцільним, оскільки фреймворк добре інтегрується з C++, підтримує механізм сигналів і слотів, дозволяє створювати інтерфейс у Qt Designer та забезпечує можливість подальшого розширення програми.

Окремою перевагою Qt є підтримка віджетів для роботи з графікою. Для відображення тривимірної моделі використовується віджет на основі OpenGL, що дає змогу поєднати графічний інтерфейс із візуалізацією 3D-сцени в одному вікні програми. Це важливо для розроблюваного застосунку, оскільки

користувач повинен не лише запускати алгоритми обробки, а й одразу бачити результат на екрані.

Для візуалізації тривимірної моделі було обрано OpenGL. Він використовується для відображення полігональної сітки, роботи з буферами вершин та індексів, налаштування шейдерів і відмальовування моделі у вікні перегляду. OpenGL є доречним вибором для даної роботи, оскільки дозволяє безпосередньо працювати з графічним представленням моделі, реалізувати суцільний і каркасний режими перегляду, а також забезпечити базове освітлення сцени.

Для імпорту 3D-моделей було обрано бібліотеку Assimp. Вона призначена для завантаження моделей із різних форматів і перетворення їх до структури, зручної для подальшої обробки. Assimp залишає можливість у майбутньому додати підтримку різних форматів 3D-моделей. Це робить застосунок більш гнучким і розширюваним.

Для збереження службової інформації про проєкт було обрано SQLite. Ця СУБД використовується для збереження інформації про файл проєкту, модель, шлях до 3D-файлу, характеристики сітки та параметри обробки. SQLite не потребує окремого сервера, зберігає базу даних в одному локальному файлі та добре підходить для настільного застосунку, який працює з одним користувачем і невеликим обсягом структурованих даних.

Середовище розробки Visual Studio було обрано як основний інструмент для написання, збирання та налагодження програмного коду. Visual Studio забезпечує зручну роботу з C++-проєктами, підтримує налагодження, перегляд помилок компіляції, керування конфігураціями збірки та інтеграцію з Qt. Це особливо важливо для desktop-додатку на Windows, оскільки середовище дозволяє ефективно працювати з підключеними бібліотеками та налаштуваннями проєкту.

Для керування зовнішніми бібліотеками може використовуватися vcpkg. Цей менеджер пакетів спрощує встановлення та підключення C++-бібліотек, таких як Assimp. Його використання дозволяє зменшити кількість ручних

налаштувань і полегшує повторне розгортання середовища розробки на іншому комп’ютері.

Таким чином, вибраний інструментарій відповідає вимогам розроблюваного програмного забезпечення. C++ забезпечує продуктивну реалізацію логіки програми, Qt — створення графічного інтерфейсу, OpenGL — відображення тривимірної моделі, Assimp — імпорт 3D-файлів, SQLite — збереження даних про проєкт, а Visual Studio та vcpkg — зручне середовище розробки та керування залежностями. Сукупність цих інструментів дозволяє реалізувати повноцінний настільний застосунок для автоматизованої перебудови топологічної сітки тривимірної моделі.

3.4 Алгоритмізація та програмування програмних модулів

Алгоритмізація та програмування програмних модулів є одним із ключових етапів розробки прикладного програмного забезпечення. На цьому етапі визначається послідовність виконання основних процесів системи, описується логіка обробки даних, будуються блок-схеми алгоритмів і демонструються фрагменти програмного коду, які відображають особливості реалізації власного рішення.

Розроблюваний програмний додаток призначений для роботи з полігональними тривимірними моделями. Загальна логіка його роботи полягає в тому, що користувач імпортує 3D-модель із файлу, переглядає її у вікні візуалізації, за потреби обирає один зі способів перебудови топологічної сітки, задає параметри обробки, запускає алгоритм і після отримання результату експортує оброблену модель у файл. Таким чином, програма реалізує повний базовий цикл роботи з 3D-моделлю: від завантаження початкових даних до отримання нового файлу з оновленою структурою сітки.

Основою роботи програми є полігональна сітка, яку можна подати як сукупність вершин, ребер і граней. Під час імпорту модель перетворюється у внутрішнє представлення, придатне для подальшої обробки. Під час візуалізації

ці дані перетворюються у формат, зручний для графічного відображення. Під час перебудови топологічної сітки змінюється структура вершин, ребер і граней відповідно до вибраного алгоритму та заданих параметрів. Після цього результат знову передається до підсистеми візуалізації та може бути експортований у файл.

У цьому підрозділі розглянуто алгоритми основних процесів програмної системи: імпорту та підготовки тривимірної моделі, її візуалізації, експорту результату та перебудови топологічної сітки. Для кожного процесу наведено опис логіки роботи, місце для блок-схеми алгоритму, а також фрагменти програмної реалізації. Оскільки центральною задачею роботи є саме обробка топологічної структури 3D-моделі, найбільшу увагу приділено математичним основам представлення сітки, обчисленню нормалей, роботі з трикутниками, зміні кількості граней, поділу елементів сітки та вирівнюванню довжин ребер.

Такий підхід дозволяє показати не лише зовнішній сценарій роботи користувача, а й внутрішню логіку програмної системи. Блок-схеми демонструють послідовність виконання операцій, а фрагменти коду підтверджують практичну реалізацію відповідних алгоритмів у програмному додатку.

3.4.1 Алгоритм імпорту та підготовки тривимірної моделі. Алгоритм імпорту та підготовки тривимірної моделі призначений для перетворення зовнішнього файлу моделі у внутрішнє представлення, з яким надалі можуть працювати підсистеми візуалізації та обробки сітки. На цьому етапі система не лише зчитує файл, а й перевіряє коректність отриманих геометричних даних, формує буфер сітки, перераховує нормалі за потреби, створює внутрішню полігональну структуру та виконує базову підготовку моделі.

У додатку А подано блок-схему алгоритму імпорту та підготовки тривимірної моделі.

Алгоритм починається з отримання запиту на імпорт моделі та шляху до файлу. Якщо шлях до файлу є коректним, система очищає попередні дані моделі та виконує завантаження нового файлу. У програмній реалізації для цього

використовується метод loadModel(), у якому модель зчитується з файлу, після чого перевіряється наявність геометричних даних. Якщо файл не вдалося завантажити або він не містить сітки, система формує повідомлення про помилку та припиняє виконання імпорту. У коді це реалізовано через перевірку результату зчитування сцени та наявності сіток у ній.

Після успішного завантаження система формує первинний буфер геометричних даних. У цей буфер записуються координати вершин, нормалі та індекси граней. Такий підхід дозволяє відокремити початкове зчитування даних із файлу від подальшого внутрішнього представлення сітки. У структурі CpuMeshBuffer для цього передбачено три основні масиви: positions, normals та indices, які зображені на рисунку 14.

```
class CpuMeshBuffer
{
public:
    std::vector<QVector3D> positions;
    std::vector<QVector3D> normals;
    std::vector<uint32_t>  indices;

    void clear();
    bool isEmpty() const;
    void recalculateNormals();
};
```

Рис. 14 Фрагмент коду класу CpuMeshBuffer

Наступним етапом є перевірка коректності буфера сітки. Буфер вважається некоректним, якщо він не містить вершин або індексів граней. У такому випадку подальша побудова внутрішньої сітки неможлива, тому система формує повідомлення про помилку. У класі CpuMeshBuffer для цього передбачено метод isEmpty(), який повертає ознаку відсутності необхідних даних.

Окрему роль під час імпорту відіграє перевірка нормалей. Нормалі потрібні для коректного освітлення поверхні під час візуалізації. Якщо у файлі моделі нормалі відсутні або їхня кількість не відповідає кількості вершин, система виконує їх перерахунок. У програмі це реалізовано шляхом проходу по

всіх трикутниках сітки, обчислення нормалі трикутника через векторний добуток і накопичення цієї нормалі для відповідних вершин.

Математично нормаль трикутника з вершинами p_0, p_1, p_2 можна обчислити за формулою:

$$\vec{\mathbf{n}} = (\mathbf{p_1} - \mathbf{p_0}) \times (\mathbf{p_2} - \mathbf{p_0})$$

де × — векторний добуток. Після обчислення нормаль нормалізується:

$$\vec{\mathbf{n}}_{\mathbf{norm}} = \frac{\vec{\mathbf{n}}}{|\vec{\mathbf{n}}|}$$

Це дозволяє отримати одиничний вектор напряму поверхні, який надалі використовується під час освітлення моделі. На рисунку 15 показана реалізація в коді.

```
const QVector3D& p0 = positions[i0];
const QVector3D& p1 = positions[i1];
const QVector3D& p2 = positions[i2];

const QVector3D e1 = p1 - p0;
const QVector3D e2 = p2 - p0;
const QVector3D n = QVector3D::crossProduct(e1, e2);

normals[i0] += n;
normals[i1] += n;
normals[i2] += n;
```

Рис. 15 Фрагмент коду, який підраховує вектор нормалі

Після формування буфера система перетворює його у внутрішнє представлення полігональної сітки. Для цього створюються вершини внутрішньої сітки, а потім за індексами формуються трикутні грані. Перед виконанням цього етапу перевіряється, чи не є буфер порожнім, а також чи кількість індексів кратна трьом, оскільки кожна грань у підготовленій сітці розглядається як трикутник.

Після створення внутрішньої сітки виконується її підготовка до подальшого використання. Цей етап є важливим, оскільки імпортована модель може містити некоректні елементи, які заважають візуалізації або подальшій обробці. Підготовка включає видалення ізольованих вершин, видалення

вироджених граней, триангуляцію поверхні та оновлення структури сітки. У програмі ці дії об’єднані в методі sanitizeAndTriangulate() класу CoreMesh. Метод sanitizeAndTriangulate показано на рисунку 16.

```
bool CoreMesh::sanitizeAndTriangulate()
{
    if (sm.is_empty()) return false;

    PMP::remove_isolated_vertices(sm);
    PMP::remove_degenerate_faces(sm);
    PMP::triangulate_faces(sm);
    PMP::remove_isolated_vertices(sm);

    return !sm.is_empty() && sm.number_of_faces() > 0;
}
```

Рис. 16 Фрагмент коду, який відповідає за попередню обробку

З математичної точки зору виродженою можна вважати грань, площа якої наближається до нуля. Для трикутника площа обчислюється через векторний добуток двох його сторін:

$$\mathbf{S} = \frac{\mathbf{1}}{\mathbf{2}}|(\mathbf{p_1} - \mathbf{p_0}) \times (\mathbf{p_2} - \mathbf{p_0})|$$

Якщо значення S дуже мале, це означає, що вершини трикутника розташовані майже на одній прямій або збігаються, тому така грань не утворює коректної поверхні. Видалення таких елементів підвищує стабільність подальшої роботи з моделлю.

Після очищення та триангуляції система перевіряє, чи підготовлена сітка залишилася коректною. Якщо після підготовки сітка порожня або не містить граней, формується повідомлення про помилку підготовки. Якщо сітка коректна, вона перетворюється назад у буфер, придатний для візуалізації. Метод toCpuMeshBuffer() проходить по вершинах і гранях внутрішньої сітки, формує масив позицій та індексів, а за потреби повторно обчислює нормалі.

Після цього підготовлена модель зберігається як поточна модель програми. У класі OpenGLWidget для цього використовується метод setCoreMesh(), який записує нову сітку в поле m_core, оновлює графічні дані та викликає оновлення

відображення. Таким чином, після завершення імпорту модель стає доступною для перегляду та подальшої перебудови топологічної сітки. На рисунку 17 представлено метод оновлення моделі.

```
void OpenGLWidget::setCoreMesh(CoreMesh core)
{
    m_core = std::move(core);

    if (isValid()) {
        makeCurrent();
        refreshGPUFromCore(true);
        doneCurrent();
    }

    update();
}
```

Рис. 17 Фрагмент коду оновлення поточної моделі (CoreMesh)

Отже, алгоритм імпорту та підготовки тривимірної моделі складається не лише із завантаження файлу, а й із послідовності перевірок і перетворень. Система перевіряє шлях до файлу, зчитує геометричні дані, формує буфер сітки, перераховує нормалі за потреби, створює внутрішнє представлення моделі, очищує та триангулює сітку, після чого передає підготовлену модель до підсистем візуалізації та обробки. Такий підхід дозволяє зменшити ймовірність помилок під час подальшої роботи з моделлю та забезпечує коректну підготовку даних для наступних етапів програми.

3.4.2 Алгоритм візуалізації тривимірної моделі. Алгоритм візуалізації тривимірної моделі призначений для відображення підготовленої полігональної сітки у вікні програми. На цьому етапі система перетворює внутрішнє представлення моделі у набір даних, придатний для графічного відображення, передає ці дані до графічних буферів, налаштовує параметри сцени, формує матриці перетворення та виконує відмальовування моделі. Блок-схему цього алгоритму продемонстровано на рисунку 18.

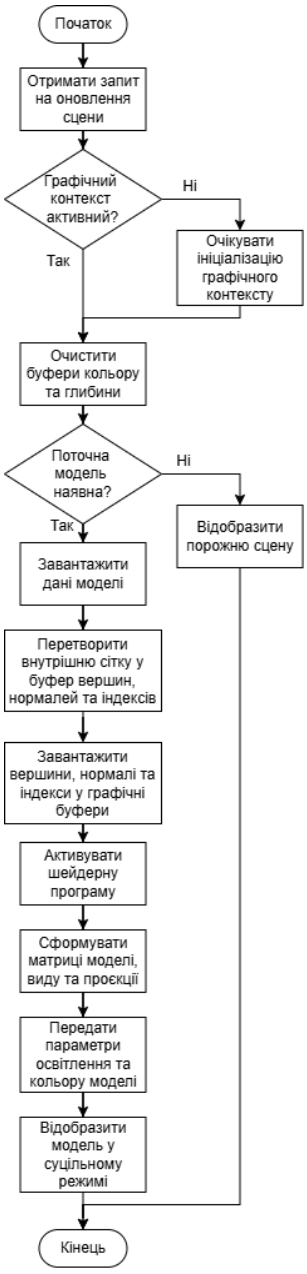


Рис. 18 Блок-схема алгоритму візуалізації тривимірної моделі

Відповідно до блок-схеми, алгоритм починається з отримання запиту на оновлення сцени. Такий запит може виникати після імпорту моделі, зміни її структури, оновлення параметрів перегляду або необхідності перемалювати вікно. Після цього система перевіряє, чи активний графічний контекст. Якщо графічний контекст ще не ініціалізовано, система очікує його створення і не виконує відображення моделі.

Після успішної перевірки графічного контексту система очищує буфери кольору та глибини. Це необхідно для того, щоб перед побудовою нового кадру прибрати дані попереднього відображення. У програмній реалізації для цього

використовується очищення кольорового буфера та буфера глибини, що дозволяє коректно відображати тривимірні об’єкти з урахуванням їхнього просторового положення. У методі paintGL() спочатку задається колір фону сцени, після чого виконується очищення відповідних буферів.

Далі система перевіряє, чи наявна поточна модель. Якщо модель відсутня або графічне представлення порожнє, відображається порожня сцена, а виконання алгоритму завершується. Якщо модель присутня, система переходить до завантаження даних моделі та підготовки їх до відображення.

У структурі класу OpenGLWidget передбачено зберігання внутрішньої сітки, проміжного буфера та графічного представлення моделі. Для цього використовуються об’єкти CoreMesh, CpuMeshBuffer і MeshGL, що дозволяє розділити внутрішнє геометричне представлення, дані на стороні процесора та дані, які передаються для відображення.

Після перевірки наявності моделі система перетворює внутрішню сітку у буфер вершин, нормалей та індексів. Це необхідно, оскільки внутрішнє представлення сітки зручне для геометричної обробки, але для візуалізації потрібен послідовний набір вершин і індексів. У методі refreshGPUFromCore() виконується підготовка сітки, перетворення її у CpuMeshBuffer, передавання в графічну підсистему та оновлення сцени. Код цього методу продемонстровано на рисунку 19.

```
void OpenGLWidget::refreshGPUFromCore(bool recomputeNormals)
{
    if (!isValid())
        return;

    if (m_core.empty()) {
        m_cpuCache.clear();
        update();
        return;
    }

    m_core.sanitizeAndTriangulate();
    m_cpuCache = m_core.toCpuMeshBuffer(recomputeNormals);

    uploadToGPU();
    update();
}
```

Рис. 19 Перетворення внутрішньої сітки у буфер для візуалізації

Наступним кроком є передавання даних у графічні буфери. На цьому етапі система завантажує координати вершин, нормалі та індекси граней у відповідні OpenGL-буфери. У класі MeshGL для цього використовуються VAO, VBO та EBO. Метод upload() перевіряє коректність даних, формує масив вершин із позиціями та нормалями, після чого передає ці дані у графічну пам'ять. Процес завантаження показано у фрагменті коду представленому на рисунку 20.

```
m_vbo.bind();
m_vbo.allocate(verts.data(), int(verts.size() * sizeof(Vertex)));

m_ebo.bind();
m_ebo.allocate(cpu.indices.data(), int(cpu.indices.size() * sizeof(uint32_t)));

f->glEnableVertexAttribArray(0);
f->glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, px));

f->glEnableVertexAttribArray(1);
f->glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, nx));
```

Рис. 20 Завантаження вершин, нормалей та індексів у графічні буфери

Після завантаження даних система активує шейдерну програму. Шейдери відповідають за обробку вершин і визначення кольору фрагментів моделі. У вершинному шейдері координати вершини перетворюються з локального простору моделі у простір відображення. Для цього використовуються три основні матриці: матриця моделі, матриця виду та матриця проєкції.

Математично перетворення координати вершини можна подати так:

$$\mathbf{p_{clip}} = \mathbf{P} \cdot \mathbf{V} \cdot \mathbf{M} \cdot \mathbf{p_{model}}$$

де p_{model} — координата вершини в локальному просторі моделі,

M — матриця моделі,

V — матриця виду,

P — матриця проєкції,

p_{clip} — координата після перетворення для подальшого відображення.

У програмі матриця виду формується через зміщення камери по осі Z, а матриця моделі — через обертання навколо осей X та Y. Далі ці матриці передаються у шейдерну програму як uniform-змінні. Фрагмент коду, що відповідає за роботу з цими матрицями, наведений на рисунку 21.

```
QMatrix4x4 view, model;
view.setToIdentity();
view.translate(0, 0, -cameraDistance);

model.setToIdentity();
model.rotate(rotationX, 1.0f, 0.0f, 0.0f);
model.rotate(rotationY, 0.0f, 1.0f, 0.0f);

shader.setUniformValue("projection", projection);
shader.setUniformValue("view", view);
shader.setUniformValue("model", model);
```

Рис. 21 Формування та передавання матриць моделі, виду та проєкції

Матриця проєкції формується при зміні розміру області відображення. У програмі використовується перспективна проєкція з кутом огляду 45 градусів, співвідношенням сторін вікна та ближньою і дальньою площинами відсікання. Це дозволяє отримати реалістичне сприйняття глибини сцени, коли віддалені частини моделі виглядають меншими. На рисунку 22 зображено код методу resizeGL().

```
void OpenGLWidget::resizeGL(int w, int h)
{
    projection.setToIdentity();
    projection.perspective(45.0f, float(w) / float(std::max(h, 1)), 0.1f, 100.0f);
}
```

Рис. 22 Формування матриці перспективної проєкції

Окремим етапом алгоритму є передавання параметрів освітлення та кольору моделі. Для базового освітлення використовується напрям джерела світла, колір світла та колір об’єкта. У фрагментному шейдері обчислюється дифузна складова освітлення на основі скалярного добутку нормалі поверхні та напряму до світла. Математично це можна подати так:

$$\mathbf{I} = \mathbf{max}(\vec{N} \cdot \vec{L}, \mathbf{0})$$

де N — нормалізована нормаль поверхні,

L — нормалізований напрям до джерела світла. Якщо поверхня повернута до світла, значення скалярного добутку більше нуля, і ділянка моделі освітлюється сильніше. Якщо поверхня відвернута від світла, значення обмежується нулем. Фрагмент шейдеру наведено на рисунку 23.

```
vec3 norm = normalize(Normal);
vec3 L = normalize(-lightDir);
float diff = max(dot(norm, L), 0.0);

vec3 ambient = 0.2 * lightColor;
vec3 diffuse = diff * lightColor;

vec3 result = (ambient + diffuse) * objectColor;
FragColor = vec4(result, 1.0);
```

Рис. 23 Обчислення освітлення у фрагментному шейдері

Після налаштування сцени система виконує відображення моделі у суцільному режимі. Для цього встановлюється режим заповнення полігонів, у шейдер передається ознака звичайного відображення, після чого викликається метод draw() об’єкта MeshGL. У цьому методі виконується прив’язка VAO та виклик glDrawElements(), який відображає модель за індексами граней.

Таким чином, алгоритм візуалізації тривимірної моделі складається з кількох послідовних етапів: перевірки графічного контексту, очищення сцени, перевірки наявності моделі, формування буфера візуалізації, завантаження даних у графічну пам’ять, активації шейдерної програми, формування матриць перетворення, передавання параметрів освітлення та відображення моделі. Такий підхід дозволяє відокремити геометричну обробку від графічного представлення та забезпечує коректне оновлення моделі після імпорту або перебудови топологічної сітки.

3.4.3 Алгоритм експорту обробленої моделі. Алгоритм експорту обробленої моделі призначений для збереження поточного стану тривимірної моделі у зовнішній файл. Після імпорту або перебудови топологічної сітки модель зберігається у внутрішньому представленні програми, тому перед експортом система повинна отримати актуальну геометрію, перевірити її наявність, сформувати шлях до файлу результату та виконати запис у форматі OBJ. Блок-схему алгоритму зображено на рисунку 24.

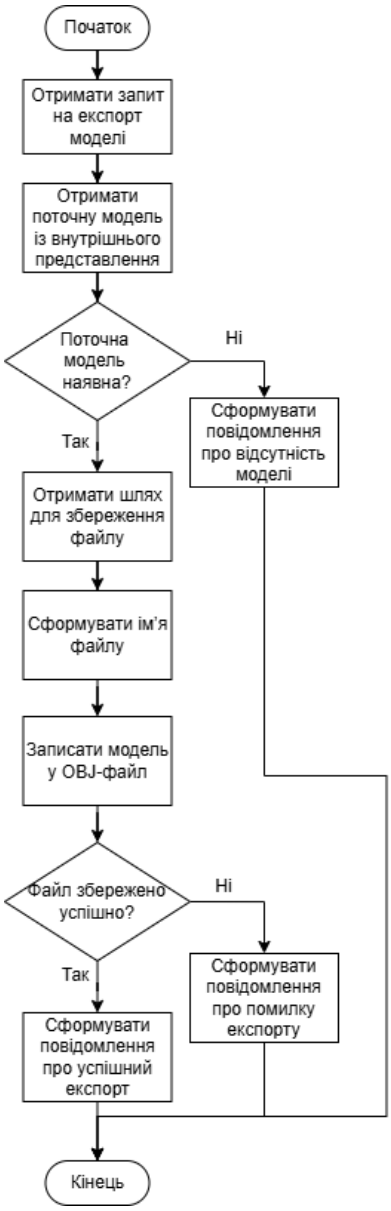


Рис. 24 Блок-схема алгоритму експорту обробленої тривимірної моделі

Відповідно до блок-схеми, алгоритм починається з отримання запиту на експорт моделі. Після цього система звертається до внутрішнього представлення та отримує поточну модель, яка зберігається у вигляді об'єкта CoreMesh. Якщо модель відсутня, тобто внутрішня сітка порожня, система формує повідомлення про відсутність моделі та завершує виконання алгоритму.

3.4.4 Алгоритм перебудови топологічної сітки. Алгоритм перебудови топологічної сітки є центральною частиною розроблюваної програмної системи, оскільки саме він забезпечує зміну структури полігональної моделі після її імпорту та підготовки. На цьому етапі система працює вже не з початковим файлом моделі, а з внутрішнім представленням сітки CoreMesh, яке

містить вершини, ребра та грані моделі. У загальному вигляді полігональну сітку можна подати як:

$$\mathbf{M} = (\mathbf{V}, \mathbf{E}, \mathbf{F})$$

де V — множина вершин,

E — множина ребер,

F — множина граней. Під час перебудови топологічної сітки змінюється один або кілька елементів цієї структури: кількість вершин, кількість граней, довжини ребер або спосіб з’єднання елементів між собою.

У програмі реалізовано три способи обробки сітки: ізотропну перебудову, поділ сітки та спрощення сітки. Для вибору конкретного алгоритму використовується підхід Strategy: головне вікно зчитує параметри, визначає вибраний алгоритм і запускає відповідний клас-стратегію. У коді це реалізовано в методі onRemeshButtonClicked(), де після перевірки наявності моделі створюється об’єкт RemeshParams, виконується підготовка сітки, а потім через оператор switch запускається одна зі стратегій: DecimateStrategy, SubdivideStrategy або IsotropicRemeshStrategy. Загальна блок-схема для перебудови показана на рисунку 25.

На загальній блок-схемі показано спільну логіку запуску процесу обробки. Алгоритм починається з отримання запиту на перебудову топологічної сітки. Далі система отримує поточну модель і перевіряє, чи вона наявна. Якщо модель відсутня, формується повідомлення про помилку, а виконання алгоритму завершується. Якщо модель наявна, система зчитує параметри обробки, очищує та триангулює сітку, після чого визначає обраний алгоритм.

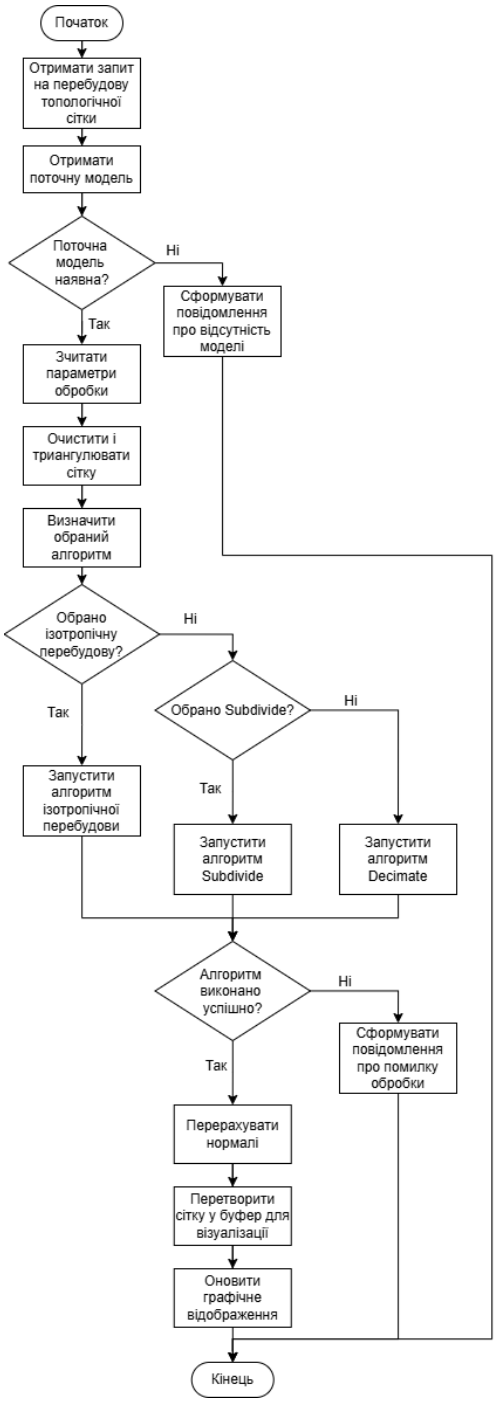


Рис. 25 Загальна блок-схема алгоритму перебудови топологічної сітки

Після вибору алгоритму запускається відповідна процедура. Якщо обрано ізотропну перебудову, система переходить до алгоритму вирівнювання довжин ребер. Якщо обрано Subdivide, виконується алгоритм поділу трикутників. Якщо не обрано жоден із цих двох варіантів, запускається алгоритм Decimate, тобто спрощення сітки. Після завершення будь-якого з трьох алгоритмів система перевіряє, чи обробка виконана успішно. У разі помилки формується повідомлення про помилку обробки. Якщо обробка успішна, система

перераховує нормалі, перетворює сітку у буфер для візуалізації та оновлює графічне відображення. На рисунку 26 зображено код, який відповідає за вибір і запуск необхідного алгоритму.

```
CoreMesh& core = oglWidget->coreMesh();
if (core.empty()) {
    qWarning() << "No mesh loaded (core is empty).";
    return;
}

RemeshParams params;
params.targetEdgeLength = ui->SpinBoxRes->value();
params.subdivLevels = ui->SpinBox_Subd_level->value();
params.decimateRatio = ui->SpinBox_decim_ratio->value();

core.sanitizeAndTriangulate();

switch (m_selectedAlgo)
{
case RemeshAlgo::Decimate: {
    DecimateStrategy s;
    ok = s.run(core, params, &err);
    break;
}
case RemeshAlgo::Subdivide: {
    SubdivideStrategy s;
    ok = s.run(core, params, &err);
    break;
}
case RemeshAlgo::Isotropic:
default: {
    IsotropicRemeshStrategy s;
    ok = s.run(core, params, &err);
    break;
}
```

Рис. 26 Вибір і запуск алгоритму перебудови топологічної сітки

Після виконання алгоритму система оновлює графічне представлення моделі. Для цього викликається метод refreshGPUFromCore(true), який перетворює оновлену сітку у буфер для візуалізації, перераховує нормалі та передає нові дані до графічної підсистеми.

Алгоритм Decimate призначений для зменшення кількості граней моделі. Його доцільно використовувати тоді, коли модель має надмірну деталізацію або містить занадто велику кількість полігонів. Блок-схему алгоритму наведено на рисунку 27.

Рис. 27 Блок-схема алгоритму спрощення топологічної сітки Decimate

Відповідно до блок-схеми, алгоритм починається з отримання поточної сітки CoreMesh. Далі система визначає поточну кількість граней — F. Після цього зчитується коефіцієнт спрощення r, на основі якого обчислюється цільова кількість граней:

$$\mathbf{F_{target} = F_{current} \cdot r}$$

де r — коефіцієнт збереження частини граней. Наприклад, якщо r=0.5, то система прагне залишити приблизно половину від початкової кількості граней.

У програмній реалізації цільова кількість граней обчислюється у функції computeTargetFaces(). Якщо задано коефіцієнт decimateRatio, він обмежується діапазоном від 0 до 1, після чого поточна кількість граней множиться на цей коефіцієнт. Також результат додатково обмежується, щоб не отримати некоректне або надто мале значення. Фрагмент коду, що відповідає за визначення цільової кількості граней наведено на рисунку 28.

```
static std::size_t computeTargetFaces(std::size_t curF, const RemeshParams& p)
{
    std::size_t targetF = 0;

    if (p.decimateRatio > 0.0) {
        const double r = std::clamp(p.decimateRatio, 0.0, 1.0);
        targetF = static_cast<std::size_t>(std::floor(double(curF) * r));
    }
    else {
        targetF = (p.decimateTargetFaces > 0) ? p.decimateTargetFaces : curF;
    }

    targetF = std::max<std::size_t>(4, std::min<std::size_t>(targetF, curF));
    return targetF;
}
```

Рис. 28 Обчислення цільової кількості граней для спрощення сітки

Після визначення цільової кількості граней система задає функцію вартості згортання ребер і правило розміщення нової вершини. Основна операція спрощення — це згортання ребра:

$$\mathbf{(v_i, v_j) \rightarrow v_{new}}$$

У простому випадку нова вершина може бути розміщена в середині ребра:

$$\mathbf{v_{new} = \frac{v_i + v_j}{2}}$$

Під час згортання дві вершини ребра замінюються однією новою вершиною, а суміжні грані та ребра перебудовуються. Цей процес повторюється доти, доки кількість граней не досягне значення F_{target}. У коді для цього використовується edge_collapse() із критерієм зупинки Face_count_stop_predicate. Після завершення спрощення система очищує та триангулює нову сітку, а також обчислює статистику після обробки. У

результаті модель отримує меншу кількість граней, але зберігає загальну форму початкового об’єкта.

Алгоритм Subdivide виконує протилежну задачу порівняно з Decimate: він збільшує деталізацію сітки шляхом поділу трикутних граней. Основна ідея полягає в тому, що кожний трикутник поділяється на чотири менші трикутники. Блок-схема представлена на рисунку 29.

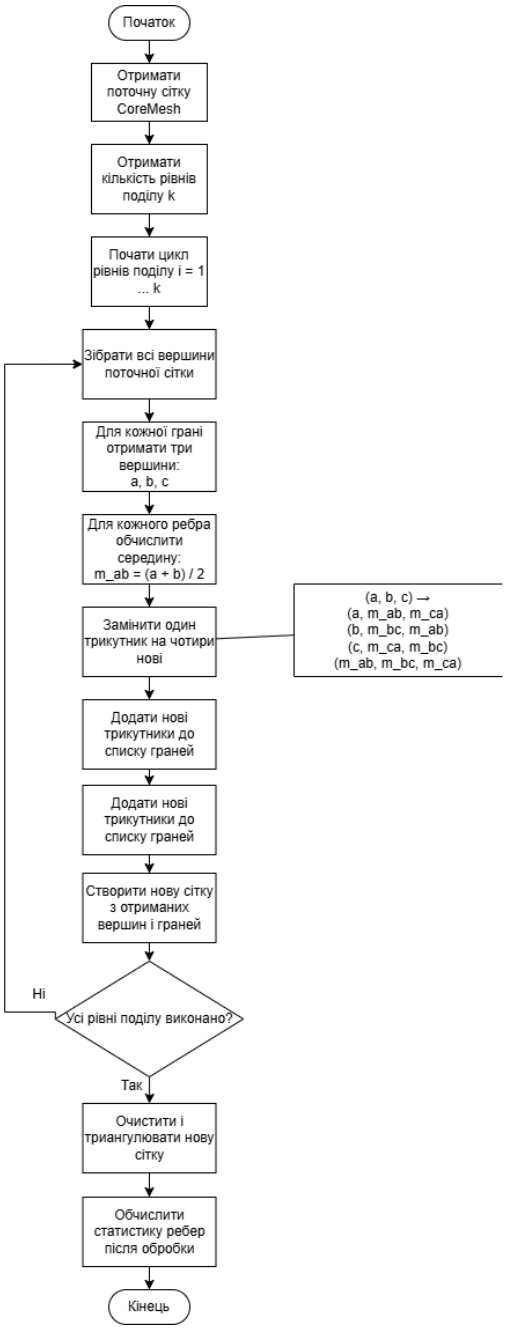


Рис. 29 Блок-схема алгоритму поділу топологічної сітки Subdivide

Відповідно до блок-схеми, алгоритм починається з отримання поточної сітки CoreMesh і кількості рівнів поділу k. Далі запускається цикл рівнів поділу

від 1 до k. На кожному рівні система збирає всі вершини поточної сітки, а потім для кожної грані отримує три вершини: a, b, c. Для кожного ребра обчислюється середина:

$$\mathbf{m_{ab}} = \frac{\mathbf{a} + \mathbf{b}}{\mathbf{2}}$$

Аналогічно обчислюються середини інших ребер. Після цього один початковий трикутник замінюється на чотири нові:

$$(\mathbf{a}, \mathbf{b}, \mathbf{c}) \rightarrow \begin{cases} (\mathbf{a}, \mathbf{m_{ab}}, \mathbf{m_{ca}}) \\ (\mathbf{b}, \mathbf{m_{bc}}, \mathbf{m_{ab}}) \\ (\mathbf{c}, \mathbf{m_{ca}}, \mathbf{m_{bc}}) \\ (\mathbf{m_{ab}}, \mathbf{m_{bc}}, \mathbf{m_{ca}}) \end{cases}$$

Саме ця формула показана на блок-схемі як математична суть алгоритму. Після одного рівня поділу кількість граней збільшується приблизно у чотири рази. У програмній реалізації для кожного ребра створюється ключ EdgeKey, щоб середина спільного ребра не створювалася кілька разів для сусідніх трикутників. Якщо середина ребра вже існує, система повторно використовує її індекс. Якщо ні — створює нову вершину в середині ребра. На рисунку 30 наведено метод, який обраховує середню точку.

```
static CoreMesh::Point midpoint(const CoreMesh::Point& p0, const CoreMesh::Point& p1)
{
    return CoreMesh::Point(
        (p0.x() + p1.x()) * 0.5,
        (p0.y() + p1.y()) * 0.5,
        (p0.z() + p1.z()) * 0.5
    );
}
```

Рис. 30 Обчислення середини ребра під час поділу сітки

Далі для кожної грані отримуються три вершини, обчислюються середини ребер і формуються чотири нові трикутники. У коді це реалізовано через додавання чотирьох елементів до списку tris. Фрагмент коду, який показує додавання трикутників, продемонстровано на рисунку 31.

```
const std::size_t m01 = getMid(i0, i1);
const std::size_t m12 = getMid(i1, i2);
const std::size_t m20 = getMid(i2, i0);

tris.push_back({ i0,  m01, m20 });
tris.push_back({ i1,  m12, m01 });
tris.push_back({ i2,  m20, m12 });
tris.push_back({ m01, m12, m20 });
```

Рис. 31 Заміна одного трикутника на чотири нові

Після обробки всіх граней система створює нову сітку з отриманих вершин і граней, замінює стару сітку новою, очищує та триангулює її. Якщо задано кілька рівнів поділу, процес повторюється потрібну кількість разів. У результаті модель отримує більшу щільність полігональної сітки та може виглядати більш деталізованою.

Алгоритм ізотропної перебудови топологічної сітки спрямований не просто на збільшення або зменшення кількості граней, а на вирівнювання структури сітки. Його основна мета — зробити довжини ребер більш рівномірними та наблизити їх до заданого цільового значення. Блок-схему цього алгоритму можна переглянути в додатку Б.

У цьому алгоритмі для кожного ребра e з кінцевими вершинами p_i та p_j обчислюється довжина:

$$\mathbf{l(e) = \|p_i - p_j\|}$$

Мета алгоритму полягає в наближенні довжин ребер до цільового значення. Відповідно до блок-схеми, алгоритм починається з отримання поточної сітки CoreMesh, параметра targetEdgeLength і кількості ітерацій. Далі система обчислює довжини всіх ребер і статистику.

Ці значення дозволяють оцінити стан сітки до обробки та після неї. У коді для цього використовується функція edgeLength(), яка обчислює відстань між двома вершинами ребра, а також функція computeEdgeStats(), яка визначає мінімальну, максимальну та середню довжину ребер. Метод edgeLength() можна переглянути на рисунку 32.

```
static double edgeLength(const CoreMesh::SMesh& sm, CoreMesh::SMesh::Edge_index e)
{
    auto h = sm.halfedge(e);
    auto v0 = sm.source(h);
    auto v1 = sm.target(h);

    const auto& p0 = sm.point(v0);
    const auto& p1 = sm.point(v1);

    const double dx = p0.x() - p1.x();
    const double dy = p0.y() - p1.y();
    const double dz = p0.z() - p1.z();
    return std::sqrt(dx * dx + dy * dy + dz * dz);
}
```

Рис. 32 Обчислення довжини ребра в алгоритмі ізотропної перебудови

Після обчислення статистики система позначає межові ребра як обмежені. Це потрібно для того, щоб контролювати поведінку алгоритму на межах моделі. Далі обчислюється допустима довжина межового ребра. Система ділить занадто довгі межові ребра. Це робиться для того, щоб під час подальшої обробки уникнути некоректної поведінки на межах сітки. У програмній реалізації межові ребра позначаються через карту властивостей, після чого за потреби викликається процедура split_long_edges(). Після цього система збирає список граней для обробки та запускає ітерації ізотропної перебудови.

Окремим кроком на блок-схемі показано релаксацію вершин. Її можна подати як зміщення вершини до середнього положення її сусідів:

$$\mathbf{p_i'} = \frac{\mathbf{1}}{\mathbf{n}} \sum \mathbf{p_j}$$

де p_i' — нове положення вершини,

n — кількість сусідніх вершин,

p_j — координати сусідніх вершин. Така операція допомагає зробити сітку більш рівномірною та зменшити локальні нерівності.

Після завершення ітерацій система очищує та триангулює сітку, а також повторно обчислює статистику ребер після обробки. Це дозволяє порівняти стан сітки до та після виконання алгоритму. У результаті ізотропна перебудова не просто змінює кількість граней, а прагне зробити структуру сітки більш рівномірною з погляду довжин ребер і форми трикутників.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмної системи є важливим етапом перевірки працездатності розробленого програмного забезпечення. Його основна мета полягає у виявленні помилок, перевірці відповідності системи поставленим вимогам, а також підтвердженні того, що основні функції додатку працюють коректно в типових і помилкових сценаріях.

У межах даної роботи тестування виконувалося для перевірки основного циклу роботи програмної системи: запуску додатку, імпорту тривимірної моделі, її візуалізації, виконання алгоритмів перебудови топологічної сітки, експорту результату та реакції системи на некоректні дії. Оскільки розроблений додаток орієнтований на роботу користувача з графічним інтерфейсом, основним видом перевірки було обрано функціональне тестування методом «чорної скриньки».

Функціональне тестування передбачає перевірку системи з точки зору її зовнішньої поведінки. У цьому випадку не аналізується внутрішня структура програмного коду під час виконання тесту, а перевіряється відповідність між вхідними діями, очікуваним результатом і фактичною реакцією системи. Такий підхід є доцільним для розроблюваного додатку, оскільки основні функції мають чітко визначені вхідні дані та очікувані результати.

Процес тестування системи можна подати у вигляді такої послідовності: спочатку визначається функція, яку необхідно перевірити, далі обираються вхідні дані або дія, після цього виконується тестовий сценарій, фіксується фактичний результат і порівнюється з очікуваним. Якщо фактичний результат відповідає очікуваному, тест вважається пройденим. Якщо результат

відрізняється, необхідно визначити причину помилки та внести відповідні зміни до програмної системи.

Оскільки основною функцією системи є перебудова топологічної сітки, окрему увагу було приділено перевірці результатів роботи алгоритмів. Для алгоритму Decimate основним критерієм є зменшення кількості граней моделі. Для алгоритму Subdivide перевіряється збільшення кількості граней після поділу трикутників. Для алгоритму Isotropic перевіряється оновлення структури сітки та наближення довжин ребер до заданого цільового значення.

У таблиці в додатку В наведено приклади функціональних тестових сценаріїв, які використовувалися для перевірки основних можливостей програмної системи.

Окремо було виконано перевірку алгоритмів обробки топологічної сітки. Для цього використовувалася кілька тестових моделей у форматі OBJ. Перед запуском алгоритму фіксувалися початкові характеристики моделі: кількість вершин, кількість граней і візуальний стан сітки. Після виконання алгоритму ці показники порівнювалися з очікуваним результатом.

Для алгоритму Decimate очікуваним результатом є зменшення кількості граней. Це означає, що після виконання алгоритму модель повинна містити менше полігонів, ніж до обробки. Такий результат підтверджує, що операція спрощення виконана правильно.

Для алгоритму Subdivide очікуваним результатом є збільшення кількості граней. Оскільки в реалізації використовується поділ одного трикутника на чотири нові, після одного рівня поділу кількість граней повинна збільшитися приблизно у чотири рази. Така перевірка дозволяє підтвердити математичну правильність роботи алгоритму.

Для алгоритму Isotropic основним критерієм є не лише зміна кількості елементів сітки, а й вирівнювання довжин ребер. У цьому випадку перевіряється, чи наближаються довжини ребер до заданого значення. Також оцінюється візуальна рівномірність сітки після обробки.

У межах тестування також перевірялася реакція системи на помилкові ситуації. До таких ситуацій належать відсутність завантаженої моделі під час запуску обробки, скасування вибору файлу під час імпорту, скасування вибору папки під час експорту та спроба завантаження некоректного файлу. У кожному з таких випадків програма не повинна завершувати роботу аварійно, а має безпечно припинити виконання операції або сформувати повідомлення про помилку.

У результаті проведення тестування було перевірено основні функції програмної системи: запуск додатку, імпорт тривимірної моделі, відображення моделі, зміну режиму перегляду, виконання алгоритмів перебудови топологічної сітки та експорт результату у файл. Функціональні тестові сценарії підтвердили, що система виконує основні дії відповідно до очікуваної логіки.

4.2 Вимоги до апаратного та програмного забезпечення

Для впровадження та експлуатації розробленої програмної системи необхідно визначити вимоги до апаратного та програмного забезпечення. Оскільки система є настільним застосунком для автоматизованої перебудови топологічної сітки тривимірної моделі, вона розгортається на локальному комп’ютері користувача та не потребує підключення до віддаленого сервера. Усі основні операції, пов’язані з імпортом, візуалізацією, обробкою та експортом моделі, виконуються локально.

Програмна система взаємодіє з файловою системою для відкриття та збереження 3D-моделей. Також передбачено використання локальної бази даних SQLite для збереження службової інформації про проєкт, параметри обробки та характеристики моделі. Відображення тривимірної моделі здійснюється засобами OpenGL, а графічний інтерфейс користувача реалізується з використанням Qt.

4.2.1 Апаратні вимоги. Для використання програмної системи необхідний персональний комп’ютер або ноутбук, який забезпечує стабільну роботу

настільних застосунків із графічним інтерфейсом та апаратно прискореною 3D-візуалізацією. Оскільки програма працює з полігональними моделями, важливими є продуктивність процесора, обсяг оперативної пам’яті та підтримка сучасної версії OpenGL графічним адаптером. Загальний список вимог до апаратного забезпечення наведений у таблиці 4.1.

Таблиця 4.1

Вимоги до апаратного забезпечення

Компонент	**Мінімальні вимоги**
Процесор	Intel Core i3 / AMD Ryzen 3 або вище
Оперативна пам’ять	Від 8 ГБ RAM
Відеоадаптер	Відеокарта або вбудоване графічне ядро з підтримкою OpenGL 3.3 або вище
Вільне місце на диску	Від 500 МБ для встановлення програми та додатковий простір для моделей і проєктних файлів
Пристрої введення	Миша або тачпад для керування оглядом моделі
Монітор	Роздільна здатність не нижче 1366×768

Вимога щодо підтримки OpenGL 3.3 є важливою, оскільки візуалізація моделі виконується у графічному вікні. Процесор використовується для зчитування, підготовки та обробки полігональної сітки, а оперативна пам’ять — для зберігання вершин, граней, нормалей та інших геометричних даних під час роботи програми.

Для моделей невеликої та середньої складності достатньо мінімальної конфігурації. Проте для комфортної роботи з більш деталізованими моделями бажано використовувати продуктивніший процесор, більший обсяг оперативної пам’яті та дискретний графічний адаптер. Це особливо актуально під час виконання алгоритмів перебудови топологічної сітки, оскільки вони можуть потребувати додаткових обчислень.

4.2.2 Програмні вимоги. Для функціонування системи необхідна операційна система Windows 10 або новіша. Також мають бути встановлені драйвери відеоадаптера з підтримкою OpenGL, оскільки без цього коректна

робота 3D-візуалізації може бути неможливою. Також для роботи користувача потрібна можливість запуску настільних застосунків, створених за допомогою Qt, а також доступ до файлової системи для відкриття та збереження моделей. Переглянути усі вимоги до програмного забезпечення можна у таблиці 4.2.

Таблиця 4.2

Вимоги до програмного забезпечення

Компонент	**Вимога**
Операційна система	Windows 10 або новіша
Графічні драйвери	Драйвери відеоадаптера з підтримкою OpenGL 3.3+
Графічний інтерфейс	Підтримка запуску застосунку, зібраного на базі Qt
База даних	SQLite як локальна база даних
Файлова система	Доступ до відкриття та збереження файлів моделей

Оскільки система є локальним настільним застосунком, користувачу не потрібно встановлювати або налаштовувати окремий сервер бази даних. SQLite використовується як локальна база даних, що зберігає службові дані про проєкт, параметри обробки та характеристики моделі. Такий підхід спрощує впровадження системи, оскільки для її роботи достатньо встановити програму.

Для коректної роботи програми також необхідно забезпечити доступ до файлової системи. Це потрібно для імпорту початкової 3D-моделі, збереження результату після обробки та роботи з файлами проєкту. Користувач повинен мати права на читання файлів моделей і запис у папку, куди виконується експорт результату.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмної системи призначений для розгортання готового настільного застосунку на комп’ютері користувача. Оскільки розроблена система є локальним desktop-додатком, користувачу не потрібно встановлювати середовище розробки, Qt, Assimp, SQLite або інші бібліотеки

окремо. Усі необхідні компоненти мають бути включені до інсталяційного пакету та розміщені у відповідних каталогах під час встановлення програми.

З боку користувача необхідною умовою є наявність операційної системи Windows 10 або новішої, драйверів відеоадаптера з підтримкою OpenGL 3.3 або вище, а також достатнього обсягу оперативної пам’яті та вільного місця на диску. У попередньо сформованих вимогах до системи зазначалося, що програма працює локально, використовує файлову систему, OpenGL для візуалізації, Qt для інтерфейсу та SQLite для службових даних проєкту. Користувачу не потрібно вручну встановлювати програмні бібліотеки, однак на його комп’ютері мають бути базові умови для запуску застосунку. Таким чином, користувач встановлює лише саму програму через інсталятор. Усі інші компоненти, які потрібні для роботи застосунку, мають входити до складу інсталяційного пакету.

Інсталятор повинен автоматично розмістити у каталозі програми виконуваний файл, динамічні бібліотеки, плагіни, ресурси інтерфейсу, файл бази даних, а також допоміжні каталоги для журналів. Всі компоненти інсталяційного пакету показані у таблиці 4.3.

Таблиця 4.3

Компоненти, що входять до інсталяційного пакету

Компонент	**Призначення**
RemeshApp.exe	Основний виконуваний файл програмної системи
Qt DLL-бібліотеки	Забезпечують роботу графічного інтерфейсу користувача
assimp.dll	Бібліотека для імпорту 3D-моделей
Файл бази даних	Початкова структура для збереження службових даних про проєкт
Ресурси інтерфейсу	Іконки, теми оформлення, стилі, службові файли
Папка logs	Каталог для збереження журналів роботи програми
README.txt	Коротка інструкція зі встановлення та запуску

Окремо варто зазначити, що користувач не повинен самостійно копіювати бібліотеки Qt або Assimp. Якщо інсталяційний пакет сформовано правильно, ці

файли автоматично встановлюються разом із програмою. Це зменшує ризик помилок під час першого запуску та робить програму зручнішою для використання.

Після встановлення користувач повинен мати можливість одразу запустити програму без додаткового налаштування середовища. Якщо встановлення виконано коректно, програма відкриває головне вікно, дозволяє імпортувати модель, відобразити її у вікні перегляду, виконати перебудову топологічної сітки та експортувати результат.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено програмне забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі. Розроблена система є настільним застосунком, який дозволяє користувачу імпортувати 3D-модель, переглядати її у графічному вікні, обирати один із доступних способів обробки сітки, задавати параметри алгоритму, виконувати обробку та експортувати отриманий результат у файл.

У першому розділі було проведено системний аналіз предметної області. Було розглянуто особливості подання тривимірних моделей у вигляді полігональної сітки, визначено основні проблеми таких моделей, зокрема надмірну кількість полігонів, нерівномірну структуру сітки, вироджені грані та інші дефекти. Також було визначено функціональні та нефункціональні вимоги до програмної системи, побудовано моделі предметної області та виконано огляд існуючих програмних рішень. На основі цього було обґрунтовано доцільність створення власного програмного засобу, орієнтованого саме на простий і зрозумілий процес обробки топологічної сітки.

У другому розділі було виконано проєктування інформаційного та програмного забезпечення. Було побудовано логічну модель даних у вигляді ER-діаграми, яка описує сутності проєкту, 3D-моделі, файлу моделі та параметрів обробки. Також було розроблено діаграму класів програмної системи, прості кооперації, діаграму пакетів і діаграму компонентів. Це дозволило визначити загальну структуру програми, розподіл відповідальності між її частинами та взаємодію між модулями інтерфейсу, візуалізації, геометричного ядра й алгоритмів обробки.

У третьому розділі було розглянуто розробку інформаційного та програмного забезпечення. Для збереження службових даних про проєкт було обрано SQLite, оскільки ця система управління базами даних не потребує

окремого сервера та добре підходить для локального desktop-додатку. Було розроблено фізичну структуру бази даних, яка передбачає збереження інформації про проєкт, файл моделі, характеристики 3D-моделі та параметри перебудови топологічної сітки.

Також було обґрунтовано вибір інструментарію розробки. Основною мовою програмування було обрано C++, для створення графічного інтерфейсу використано Qt, для візуалізації тривимірної сцени — OpenGL, для імпорту моделей — Assimp, а для збереження службових даних — SQLite. Такий набір інструментів дозволив реалізувати програму, яка поєднує роботу з 3D-геометрією, графічне відображення моделі та збереження інформації про проєкт.

Окрему увагу було приділено алгоритмізації та програмуванню основних модулів системи. Було описано алгоритми імпорту та підготовки тривимірної моделі, її візуалізації, експорту результату та перебудови топологічної сітки. Для обробки сітки реалізовано три підходи: Decimate для зменшення кількості граней, Subdivide для збільшення деталізації моделі та Isotropic для вирівнювання структури сітки за довжинами ребер. У роботі було розглянуто не лише загальну логіку цих алгоритмів, а й математичні принципи їхньої роботи: обчислення нормалей, поділ трикутника на чотири частини, згортання ребер, визначення цільової кількості граней і наближення довжин ребер до заданого значення.

У четвертому розділі було наведено рекомендації щодо впровадження та експлуатації системи. Було проведено функціональне тестування методом «чорної скриньки», під час якого перевірялися запуск програми, імпорт моделі, візуалізація, обертання та масштабування моделі, каркасний режим, виконання алгоритмів обробки, експорт результату та реакція системи на помилкові ситуації. Результати тестування показали, що основні функції програми працюють відповідно до очікуваної логіки, а система коректно реагує на типові помилки.

Поставлену мету роботи можна вважати досягнутою. Було створено програмне забезпечення, яке забезпечує базовий цикл роботи з тривимірною моделлю: імпорт, перегляд, вибір алгоритму, налаштування параметрів, перебудову топологічної сітки, оновлення відображення та експорт результату. Розроблена система дозволяє автоматизувати зміну структури полігональної сітки без ручного редагування окремих вершин, ребер і граней, що відповідає основній ідеї роботи.

Практична цінність розробки полягає в тому, що створений застосунок може використовуватися як навчально-практичний інструмент для роботи з полігональними 3D-моделями. Він дозволяє наочно продемонструвати різні підходи до обробки сітки, оцінити результат візуально та отримати файл моделі для подальшого використання в інших програмних середовищах. Крім того, побудована структура програми дає змогу в майбутньому розширювати функціональність системи, наприклад додавати нові алгоритми обробки, підтримку інших форматів 3D-файлів, розширену статистику сітки або можливість порівняння стану моделі до та після обробки.

Отже, у межах бакалаврської кваліфікаційної роботи було виконано аналіз предметної області, сформовано вимоги до системи, спроєктовано інформаційне та програмне забезпечення, розроблено основні програмні модулі, реалізовано алгоритми обробки топологічної сітки та проведено тестування системи. Результатом роботи є працездатний настільний програмний застосунок, який відповідає поставленій меті та може бути використаний для автоматизованої перебудови топологічної сітки тривимірної моделі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Botsch M. Polygon Mesh Processing / M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, B. Lévy. — Boca Raton : A K Peters/CRC Press, 2010. — 250 p.
2. Marschner S. Fundamentals of Computer Graphics / S. Marschner, P. Shirley. — 4th ed. — Boca Raton : CRC Press, 2016. — 748 p.
3. Qt Documentation [Електронний ресурс]. — The Qt Company. — Режим доступу: https://doc.qt.io/ — Дата звернення: 15.05.2026.
4. Khronos OpenGL Reference Pages [Електронний ресурс]. — Khronos Group. — Режим доступу: https://registry.khronos.org/OpenGL-Refpages/ — Дата звернення: 15.05.2026.
5. OpenGL 3.3 Core Profile Specification [Електронний ресурс]. — Khronos Group, 2010. — Режим доступу: https://registry.khronos.org/OpenGL/specs/gl/glspec33.core.pdf — Дата звернення: 15.05.2026.
6. Assimp — Open Asset Import Library [Електронний ресурс]. — The Open Asset Import Library. — Режим доступу: https://www.assimp.org/ — Дата звернення: 15.05.2026.
7. SQLite Documentation [Електронний ресурс]. — SQLite. — Режим доступу: https://sqlite.org/docs.html — Дата звернення: 15.05.2026.
8. About SQLite [Електронний ресурс]. — SQLite. — Режим доступу: https://sqlite.org/about.html — Дата звернення: 15.05.2026.
9. QOpenGLWidget Class [Електронний ресурс]. — Qt Documentation. — Режим доступу: https://doc.qt.io/qt-6/qopenglwidget.html — Дата звернення: 15.05.2026.
10. Qt OpenGL [Електронний ресурс]. — Qt Documentation. — Режим доступу: https://doc.qt.io/qt-6/qtopengl-index.html — Дата звернення: 15.05.2026.

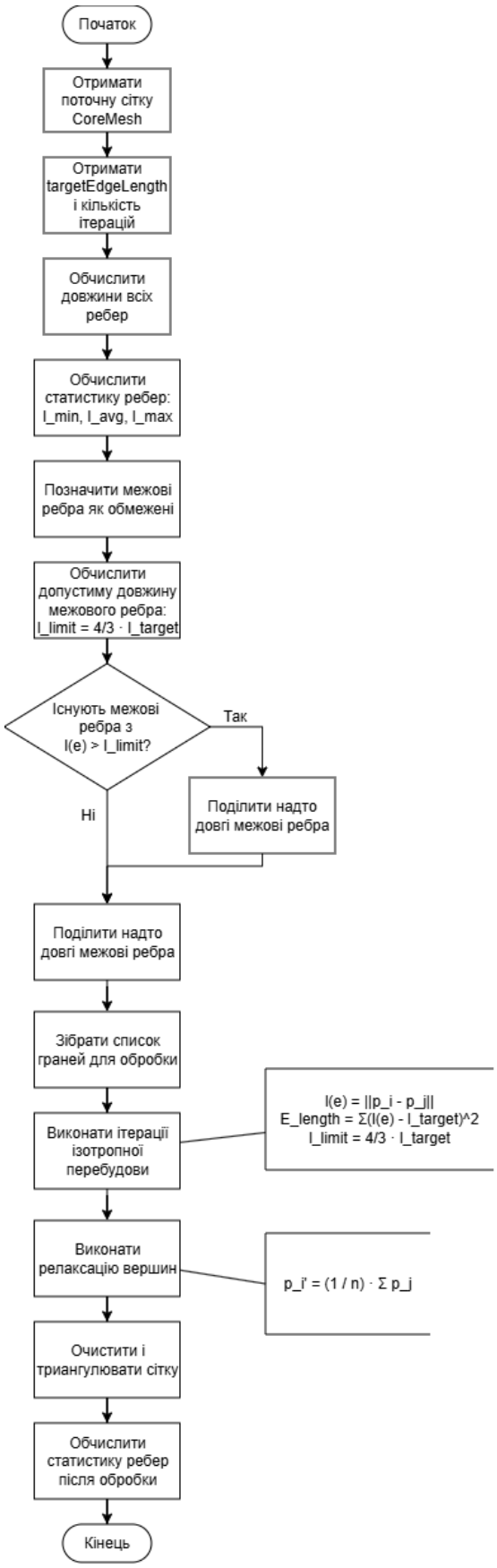
11. Wavefront OBJ File Format [Електронний ресурс]. — Library of Congress. — Режим доступу: https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml — Дата звернення: 15.05.2026.
12. vcpkg documentation [Електронний ресурс]. — Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/vcpkg/ — Дата звернення: 15.05.2026.
13. Microsoft C++, C, and Assembler documentation [Електронний ресурс]. — Microsoft Learn. — Режим доступу: https://learn.microsoft.com/en-us/cpp/ — Дата звернення: 15.05.2026.
14. Decimate Modifier — Blender Manual [Електронний ресурс]. — Blender Documentation. — Режим доступу: https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/decimate.html — Дата звернення: 15.05.2026.
15. MeshLab [Електронний ресурс]. — Visual Computing Lab, ISTI-CNR. — Режим доступу: https://www.meshlab.net/ — Дата звернення: 15.05.2026.

ДОДАТОК А

Блок-схема для алгоритму імпорту та підготовки тривимірної моделі

ДОДАТОК Б

Блок-схема алгоритму ізотропної перебудови



ДОДАТОК В

Таблиця функціональних тестових сценаріїв, які використовувалися для перевірки основних можливостей програмної системи

№	**Назва тесту**	**Вхідні дані / дія**	**Очікуваний результат**	**Фактичний результат**
TC-01	Запуск програми	Запустити виконуваний файл додатку	Відкривається головне вікно програми	Головне вікно відкрито
TC-02	Імпорт OBJ-моделі	Обрати коректний OBJ-файл	Модель завантажується та відображається у вікні перегляду	Модель відображена
TC-03	Скасування імпорту	Закрити вікно вибору файлу без вибору моделі	Програма не завершується аварійно, імпорт не виконується	Додаток продовжує роботу
TC-04	Візуалізація моделі	Завантажити модель у програму	Модель відображається у вікні перегляду	Модель відображена
TC-05	Обертання моделі	Виконати переміщення миші у вікні перегляду	Модель змінює ракурс	Обертання працює
TC-06	Масштабування моделі	Використати колесо миші	Відстань камери до моделі змінюється	Масштабування працює

Таблиця функціональних тестових сценаріїв, які використовувалися для перевірки основних можливостей програмної системи(продовження)

TC-07	Каркасний режим	Увімкнути режим відображення сітки	Поверх моделі відображається каркасна структура	Каркас відображено
TC-08	Запуск обробки без моделі	Натиснути кнопку обробки без імпорту моделі	Обробка не виконується, програма не завершується аварійно	Помилка оброблена
TC-09	Виконання Decimate	Обрати Decimate і задати коефіцієнт спрощення	Кількість граней моделі зменшується	Кількість граней зменшена
TC-10	Виконання Subdivide	Обрати Subdivide і задати рівень поділу	Кількість граней моделі збільшується	Кількість граней збільшена
TC-11	Виконання Isotropic	Обрати Isotropic і задати цільову довжину ребра	Структура сітки оновлюється відповідно до параметрів	Сітку оновлено
TC-12	Експорт моделі	Обрати папку для експорту	Створюється OBJ-файл із моделлю	Файл створено
TC-13	Перевірка експортованого файлу	Відкрити OBJ-файл у сторонньому 3D-редакторі	Модель відкривається коректно	Модель відкрита

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Троян Олександр Володимирович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення для автоматизованої перебудови топологічної сітки тривимірної моделі» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
    - [ ] інструменти генеративного ШІ не використовувалися.
    - [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [ ] стилістичного редагування та перевірки граматики;
        - [ ] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ **Олександр Троян**

(підпис)