Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей

Виконала: студентка 2-го курсу, гр. 2КІТС-22м
спеціальності 125 – Кібербезпека
Освітня програма – Кібербезпека інформаційних технологій та систем
Мовчанюк М.Т.

Керівник: д.ф. (PhD), доц. каф. МБІС
Салієва О.В.
«04» грудня 2023 р.

Опонент: к.т.н., доцент, доцент каф. ОТ
Войцеховська О.В.
«04» грудня 2023 р.

Допущено до захисту
Голова секції УБ кафедри МБІС

__________ Юрій ЯРЕМЧУК
“04” грудня 2023 р.

Вінниця ВНТУ – 2023 рік

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

______________________**Юрій ЯРЕМЧУК**
"20" вересня 2023 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
Мовчанюк Мар'яні Тимофіївні
(прізвище, ім`я, по-батькові)

1. Тема роботи «Підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей»

Керівник роботи Салієва О. В., д.ф. (PhD), доц. каф. МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "18" вересня 2023 року № 247

2. Строк подання студентом роботи за тиждень до захисту.

3. Вихідні дані до роботи: наукові роботи, монографії, книги, статті та інші публікації за темою роботи, онлайн-ресурси, стандарти, висновки із практичного застосування аналогічних програм, експертні відгуки.

4. Зміст текстової частини: в першому розділі проаналізувати стеганографічний захист інформації на основі цифрового водяного знаку; в другому розділі здійснити підвищення стійкості невидимого водяного знаку від геометричних операцій шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин; у третьому розділі здійснити програмну реалізацію удосконаленого алгоритму та здійснити його тестування; у четвертому розділі здійснити комплексне дослідження економічного потенціалу розробки.

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

– у першому розділі наведено рис. 9, табл. 1;
– у другому розділі наведено рис. 12;
– у третьому розділі наведено рис. 5, табл. 5;

– у четвертому розділі наведено табл. 7.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Салієва О. В., д.ф. (PhD), доц. каф. МБІС		
2	Салієва О. В., д.ф. (PhD), доц. каф. МБІС		
3	Салієва О. В., д.ф. (PhD), доц. каф. МБІС		
4	Причепа І. В., доцент кафедри ЕПВМ, к.е.н.		

7.Дата видачі завдання <u>20 вересня 2023 р.</u>

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1	Визначення напрямку магістерської кваліфікаційної роботи, формулювання теми	20.09.2023	24.09.2023	
2	Аналіз предметної області обраної теми	25.09.2023	07.10.2023	
3	Формулювання теоретичної бази для дослідження та визначення ключових концепцій.	08.10.2023	17.10.2023	
4	Розробка алгоритму роботи	18.10.2023	25.10.2023	
5	Написання магістерської кваліфікаційної роботи на основі розробленої теми	26.10.2023	14.11.2023	
6	Апробація отриманих результатів	15.11.2023	18.11.2023	
7	Розробка економічної частини	19.11.2023	23.11.2023	
8	Попередній захист магістерської кваліфікаційної роботи	24.11.2023	25.11.2023	
9	Виправлення, уточнення, коригування роботи	25.11.2023	03.12.2023	
10	Захист магістерської кваліфікаційної роботи	11.12.2023	15.12.2023	

Студент ____________ Мовчанюк М.Т.
(підпис)

Керівник роботи ____________ Салієва О.В.
(підпис)

АНОТАЦІЯ

УДК 004.932.052.2

Мовчанюк М. Т. Підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей. Магістерська кваліфікаційна робота зі спеціальності 125 – «Кібербезпека», освітня програма «Кібербезпека інформаційних технологій та систем». Вінниця: ВНТУ, 2023. 92 с.

На укр. мові. Бібліогр.: 53 назв; рис.: 26; табл.: 13.

У магістерській кваліфікаційній роботі здійснено підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей.

У роботі досліджено використання цифрових водяних знаків для тривимірних моделей, визначено їхню ефективність та можливі області застосування, приділяючи особливу увагу можливим атакам на цифрові водяні знаки у тривимірних моделях. Удосконалення методу вбудовування цифрових водяних знаків включає в себе впровадження алгоритму 3D-розпізнавання реберних вершин для знаходження більш стійких опорних точок, покращуючи таким чином стійкість невидимого водяного знаку до геометричних операцій перетворення у тривимірних моделях. Практично реалізовано програмний додаток, який був протестований шляхом проведення різноманітних атак на різні моделі. Результати тестування, що включають порівняння з початковим методом та визначення непомітності за допомогою PSNR, свідчать про те, що удосконалений алгоритм проявляє більшу стійкість до геометричних атак порівняно з початковим методом. Розробка має економічну обґрунтованість, що підтверджується результатами проведеного аналізу.

Ключові слова: цифровий водяний знак, тривимірна модель, стійкість, реберні вершини, геометричні операції перетворення, невидимість.

ABSTRACT

Mariana Movchanyuk. Increasing the invisible watermark robustness from geometric transformation operations by improving the embedding method based on 3D edge vertex recognition for three-dimensional models. Master's qualification work in the specialty 125 - "Cybersecurity", educational program "Cybersecurity of information technologies and systems". Vinnytsia: VNTU, 2023. 92 p.

In Ukrainian language. Bibliography: 53 titles; Figures: 26; Table 13.

In the master's qualification work, the invisible watermark robustness against geometric transformation operations was increased by improving the embedding method based on 3D edge vertex recognition for three-dimensional models.

The paper investigates the use of digital watermarks for three-dimensional models, determines their effectiveness and possible applications, paying special attention to possible attacks on digital watermarks in three-dimensional models.

Improvements to the watermark embedding method include the implementation of a 3D edge vertex recognition algorithm to find more stable anchor points, thus improving the invisible watermark's resistance to geometric transformation operations in three-dimensional models.

A software application has been practically implemented and tested by conducting various attacks on different models. The test results, which include a comparison with the original method and the determination of invisibility using PSNR, indicate that the improved algorithm is more resistant to geometric attacks than the original method.

The development has economic feasibility, which is confirmed by the results of the analysis.

Keywords: digital watermark, three-dimensional model, stability, edge vertices, geometric transformation operations, invisibility.

ЗМІСТ

ВСТУП.. 7

1 ЗАГАЛЬНИЙ АНАЛІЗ СТЕГАНОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ НА ОСНОВІ ЦИФРОВОГО ВОДЯНОГО ЗНАКУ.. 10

1.1 Аналіз основних видів цифрових водяних знаків та їх роль у сучасному цифровому середовищі 10

1.2 Аналіз найпоширеніших алгоритмів вбудовування ЦВЗ 16

1.3 Дослідження застосування ЦВЗ для тривимірних моделей 20

1.4 Аналіз атак на ЦВЗ тривимірних моделей 26

1.5 Висновок до розділу 1 та постановка задач.. 29

2 ПІДВИЩЕННЯ СТІЙКОСТІ НЕВИДИМОГО ВОДЯНОГО ЗНАКУ ВІД ГЕОМЕТРИЧНИХ ОПЕРАЦІЙ ВДОСКОНАЛЕННЯМ МЕТОДУ ВБУДОВУВАННЯ НА ОСНОВІ 3D-РОЗПІЗНАВАННЯ РЕБЕРНИХ ВЕРШИН.. 31

2.1 Алгоритм вбудовування та розпізнавання водяних знаків............................. 31

2.2 Вдосконалення алгоритму 40

2.3 Розробка алгоритму роботи програмного додатку.. 42

2.4 Висновок до розділу 2.. 48

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВДОСКОНАЛЕНОГО АЛГОРИТМУ ВБУДОВУВАННЯ ТА РОЗПІЗНАВАННЯ ВОДЯНИХ ЗНАКІВ ... 49

3.1 Практична реалізація програми вдосконаленого алгоритму вбудовування та розпізнавання водяних знаків 49

3.2 Приклад реалізації вдосконаленого алгоритму... 59

3.3 Аналіз стійкості вдосконаленого методу до геометричних атак 62

3.4 Висновки до розділу 3 65

4 ЕКОНОМІЧНА ЧАСТИНА... 67

4.1 Оцінка комерційного потенціалу розробки програмного забезпечення........ 67

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів.. 72

4.3 Розрахунок економічної ефективності наукової роботи за її можливої комерціалізації потенційним інвестором 78

4.4 Висновок до розділу 4.. 83

ВИСНОВОК.. 85

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 87

ДОДАТКИ 93

Додаток А. Технічне завдання.. 94

Додаток Б. Лістинг програми 98

Додаток В. Ілюстративний матеріал... 122

Додаток Г. Протокол перевірки на антиплагіат 128

ВСТУП

Актуальність. Стрімкий ріст популярності використання тривимірних моделей зумовлений швидким технологічним прогресом. Розвиток обчислювальних потужностей і методів візуалізації даних не лише сприяв широкому використанню тривимірних моделей, але й підвищив попит на них у різних галузях. Тривимірні моделі використовують у різних галузях – від розваг і дизайну до інженерії та наукової візуалізації, що підкреслює їхню універсальну роль у творчій, технічній та аналітичній сферах.

Зростання популярності тривимірних моделей призвело до виникнення таких проблем, як порушення авторських прав і суперечок щодо автентичності власника, поширених через легкість піратського використання 3D-моделей. Цифрові водяні знаки вважаються ефективним рішенням для забезпечення автентифікації та захисту авторських прав на тривимірні моделі.

Останніми роками проводиться численне дослідження цифрових водяних знаків для 3D-сітчастих моделей. Проте, актуальною залишається необхідність в алгоритмі, що буде стійким до геометричних операцій перетворення для ефективного захисту цих моделей.

Дослідженням методів вбудовування цифрових водяних знаків у тривимірні моделі займалися Джонатан Су, Бернд Гірод, Мохсен Ашурян, Чжоу Зуде, Сагаріка Бора, Сінью Ван та інші [1-5].

Актуальність роботи визначається тим, що геометричні перетворення суттєво впливають на ефективність застосування цифрових водяних знаків у захисті тривимірних моделей, що вимагає подальшого удосконалення алгоритмів вбудовування для забезпечення їх стійкості.

У магістерській кваліфікаційній роботі досягнуте підвищення стійкості невидимого водяного знаку до геометричних операцій перетворення полягає у вдосконаленні методу вбудовування водяних знаків за допомогою 3D-розпізнавання реберних вершин для тривимірних моделей.

Для вдосконалення методу вбудовування цифрового водяного знаку внесено зміни до алгоритму пошуку опорних точок, який використовується для вбудовування водяного знаку. Застосовано алгоритм 3D-розпізнавання реберних вершин для ефективного визначення цих опорних точок. Враховуючи стабільність цих вершин, їх використання для вбудовування водяного знаку стало більш надійним порівняно з початковим методом.

Під час практичного тестування покращеного методу на складній тривимірній моделі зроблено висновок, що удосконалений метод є більш захищеним та ефективнішим, ніж початковий.

Мета і задачі дослідження. Метою роботи є підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей та програмна реалізація удосконаленого методу.

Задачами дослідження є:

– здійснити аналіз існуючих методів вбудовування ЦВЗ та визначення їх слабких сторін;

– удосконалити метод вбудовування невидимого ЦВЗ для тривимірних моделей за рахунок підвищення його стійкості від геометричних операцій перетворення;

– розробити алгоритм роботи програмного засобу, використовуючи вдосконалений метод вбудовування невидимого ЦВЗ;

– здійснити його програмну реалізацію;

– розробити приклад реалізації удосконаленого алгоритму;

– здійснити аналіз стійкості вдосконаленого методу;

– економічно обґрунтувати доцільність впровадження розробки на основі вдосконаленого методу вбудовування ЦВЗ для тривимірних моделей.

Об’єкт дослідження – невидимий цифровий водяний знак у тривимірних моделях.

Предмет дослідження – процес вдосконалення методу вбудовування водяного знаку на основі 3D-розпізнавання реберних вершин для тривимірних

моделей з метою підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення.

Наукова новизна – вдосконалення методу вбудовування ЦВЗ на основі 3D-розпізнавання реберних вершин для тривимірних моделей.

Практична цінність – розроблено програмний додаток, що реалізує вдосконалений метод вбудовування ЦВЗ у тривимірну модель для збереження цілісності та авторських прав.

Апробація – тези доповідей у даній галузі представленні на міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» [6].

1 ЗАГАЛЬНИЙ АНАЛІЗ СТЕГАНОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ НА ОСНОВІ ЦИФРОВОГО ВОДЯНОГО ЗНАКУ

Даний розділ магістерської кваліфікаційної роботи присвячений загальному аналізу стеганографічного захисту інформації, зосереджуючи увагу на використанні цифрового водяного знаку (ЦВЗ). У розділі розглядаються види та особливості ЦВЗ, досліджується їхня роль у сучасному цифровому середовищі. Далі аналізуються найпоширеніші алгоритми вбудовування ЦВЗ та їхні характеристики. Також досліджується застосування ЦВЗ для тривимірних моделей, розкриваючи ефективність водяних знаків та можливі області застосування. Проведений аналіз атак на ЦВЗ тривимірних моделей допомагає визначити потенційні вразливості для можливості вдосконалення методів захисту від них.

1.1 Аналіз основних видів цифрових водяних знаків та їх роль у сучасному цифровому середовищі

Цифровий водяний знак – це метод захисту авторських прав на медіа-контент, який полягає у вбудовуванні унікальної ідентифікаційної інформації безпосередньо в нього. Цей метод дозволяє встановити походження та власника контенту, а також уникнути незаконного копіювання та використання файлів. Він вбудовується безпосередньо в цифровий файл і представляє собою непомітну, але унікальну марку, яка надає інформацію про автора, дату створення та походження файлу.

ЦВЗ дозволяють вбудовувати або приховувати дані у функціональних компонентах іншого цифрового контенту, таких як зображення, аудіо- чи відеокліпи, 3D-сітки тощо. Це забезпечує конфіденційність, ідентифікацію авторства і дозволяє керувати використанням цифрового контенту в різних мультимедійних середовищах.

Необхідно пояснити ключову різницю між стеганографією та ЦВЗ. Головна мета стеганографії полягає в збереженні таємного повідомлення, не викликаючи підозр про його наявність. У випадку з ЦВЗ – основна мета полягає в тому, щоб забезпечити водяний знак високою надійністю, зробити його стійким до будь-яких

змін чи маніпуляцій [7]. Алгоритми приховування ЦВЗ часто розробляються як стійкі алгоритми вбудовування, які можуть виявити вбудовані дані навіть після спотворення цифрового об'єкта [8].

Найпоширеніший спосіб класифікації ЦВЗ полягає в розділенні їх на два основних типи: видимі та невидимі. Видимі водяні знаки виглядають як графічні чи текстові елементи, які наносяться на зображення чи відео і використовуються для ідентифікації авторства та захисту від незаконного використання (рис. 1.1). Їх можуть складати логотипи компаній, авторські підписи чи будь-яка текстова чи графічна інформація [9].

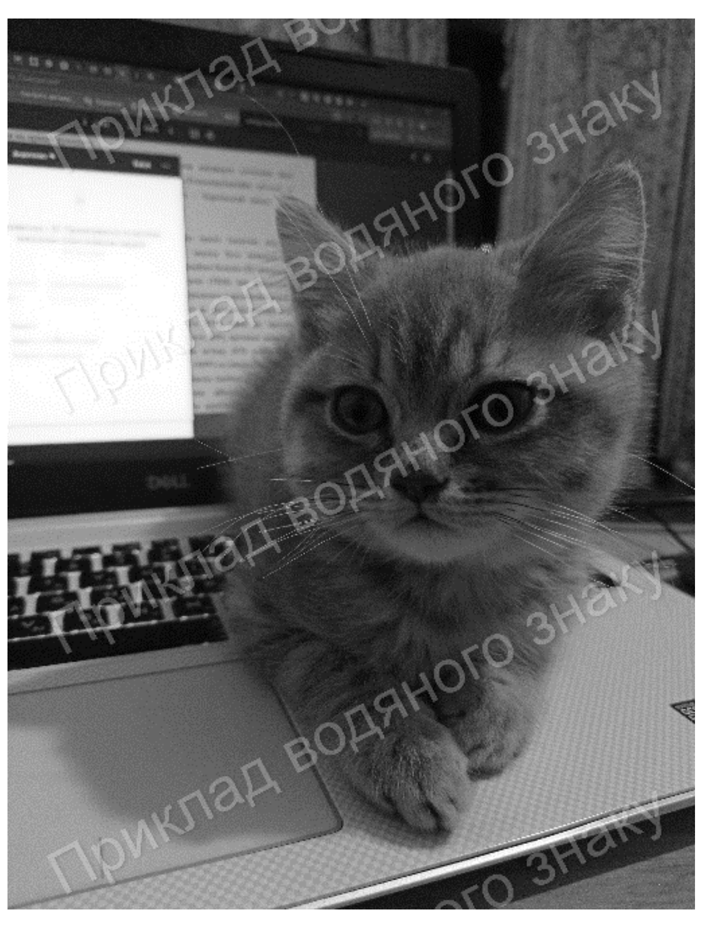


Рисунок 1.1 – Приклад видимого ЦВЗ

Невидимі водяні знаки, навпаки, вбудовуються безпосередньо у структуру цифрового контенту, таку як пікселі зображення чи біти аудіофайлу. Невидимі водяні знаки зазвичай використовуються для відстеження джерела контенту та авторства, а також для захисту від піратства та незаконного розповсюдження.

Обидва типи водяних знаків відіграють важливу роль у захисті авторських прав та ідентифікації походження цифрового контенту.

Також, схеми водяних знаків можна класифікувати за їхньою стійкістю до модифікацій у вихідному середовищі. Ці зміни можуть виникати через звичайні операції обробки файлу з ЦВЗ, такі як стиснення аудіо чи втрати якості зображень. Крім того, вони можуть бути спричинені спеціально розробленими атаками, спрямованими на виявлення водяного знака чи його пошкодження. Ступінь стійкості методу нанесення водяних знаків до модифікацій носія зазвичай називають робастністю. В залежності від вимог до надійності можна виділити наступні класи ЦВЗ:

– надійні (стійкі);

– крихкі;

– напівкрихкі.

Надійні водяні знаки мають високу стійкість до будь-яких впливів та різноманітних операцій обробки [10]. Даний клас ЦВЗ використовуються для надійного захисту авторських прав та їх відстеження у різних середовищах. Важливо враховувати, що жодна схема водяних знаків не може бути абсолютно стійкою до всіх можливих видів модифікацій.

Крихкий водяний знак надзвичайно чутливий і розроблений для виявлення навіть найменших змін у зазначеному зображенні. Цей клас водяних знаків ідеально підходить для перевірки цілісності даних і служить альтернативою стандартним цифровим підписам. Крихкий водяний знак забезпечує виявлення зловмисних спотворень, таких як додавання чи видалення елементів [11].

Більшість мультимедійних програм можуть допускати невеликі зміни в цифровому контенті, які відображаються як відхилення у даних, але не порушують автентичності змісту. Тому виділяють третій клас ЦВЗ – напівкрихкий водяний знак, який дозволяє прийнятні маніпуляції зі збереженням вмісту, такі як стиснення чи покращення. Цей клас водяних знаків має обмежену стійкість до певних модифікацій, але залишається вразливим до інших видів змін. Такі водяні знаки також можуть використовуватися для автентифікації замість більш крихких

варіантів. На практиці, усі стійкі водяні знаки можуть вважатися напівкрихкими, оскільки вибіркова стійкість не є вимогою розробників систем, а скоріше необхідністю, яка виникає внаслідок обмежень у виявленні всіх можливих змін.

ЦВЗ мають ряд ключових властивостей, які роблять їх ефективними і надійними (табл. 1.1) [12].

Таблиця 1.1 – Властивості ЦВЗ

Властивість	Опис
Надійність та стійкість	Після внесення будь-яких змін або маніпуляцій у маркований документ, цифровий водяний знак повинен бути виявлений та розпізнаний, а також залишатись стійким до підробки та змін.
Непомітність	Водяний знак повинен бути непомітним, не спотворюючи оригінальний документ і залишаючись нерозрізненним з оригіналом.
Безпека	Тільки уповноважені особи повинні мати право доступу до водяного знаку. Ключі водяних знаків гарантують, що тільки авторизовані користувачі зможуть виявити/модифікувати водяний знак.
Швидкість вбудовування та вилучення	Водяний знак повинен вбудовуватися та вилучатися швидко, що важливо для додатків зі значним обсягом даних.
Однозначність	Водяний знак повинен передавати чітку інформацію про законного власника авторських прав та місце розповсюдження.

У ЦВЗ може міститися різна інформація, включаючи ліцензійні умови, авторські права, контроль копіювання, дані для автентифікації контенту та інформація для відстеження. Ця інформація використовується для захисту від незаконного копіювання, ідентифікації документів, визначення прав власності та для відстеження передачі творів ліцензованим користувачам та від них. Незалежно від типу вмісту, який вбудовано в медіа-контент, існує баланс між корисним навантаженням водяного знаку (кількістю інформації, яку можна вбудувати в

зображення) та його стійкістю до маніпуляцій. Зазвичай, водяні знаки містять обмежену кількість інформації та частково розраховані на надмірність знаку для витримки атак, таких як обрізання чи інші зміни [13].

ЦВЗ відіграють важливу роль у захисті авторських прав, запобіганні плагіату, відстеженні розповсюдження контенту в мережі Інтернет та їх використанні у комерційних цілях.

Цифрове водяне маркування – це комплексний процес, який включає в себе такі етапи, як: вбудовування ідентифікаційної інформації або водяного знаку у цифровий контент, виявлення цього знаку для ідентифікації та вилучення його за потреби.

Загалом, для вбудовування водяного знаку потрібні вихідні дані (цифровий контент), водяний знак і додатковий відкритий або секретний ключ, що дозволяєте підвищити рівень секретності вбудовування інформації. Результатом є дані з водяним знаком, тобто уже маркований цифровий контент (рис. 1.2).

Рисунок 1.2 – Загальна схема вбудовування ЦВЗ

Нанесення ЦВЗ може відбуватися різними способами, які зазвичай розробляються для конкретних потреб. Ці методи можуть бути неуніверсальними і не підходити для всіх випадків використання.

Для виявлення водяних знаків використовують конкретний ідентифікатор водяного знаку та ключ, які порівнюються із маркованими даними, щоб визначити, чи є водяний знак коректним та вірним (рис. 1.3). Цей процес забезпечує надійність та достовірність водяних знаків у цифровому контенті.

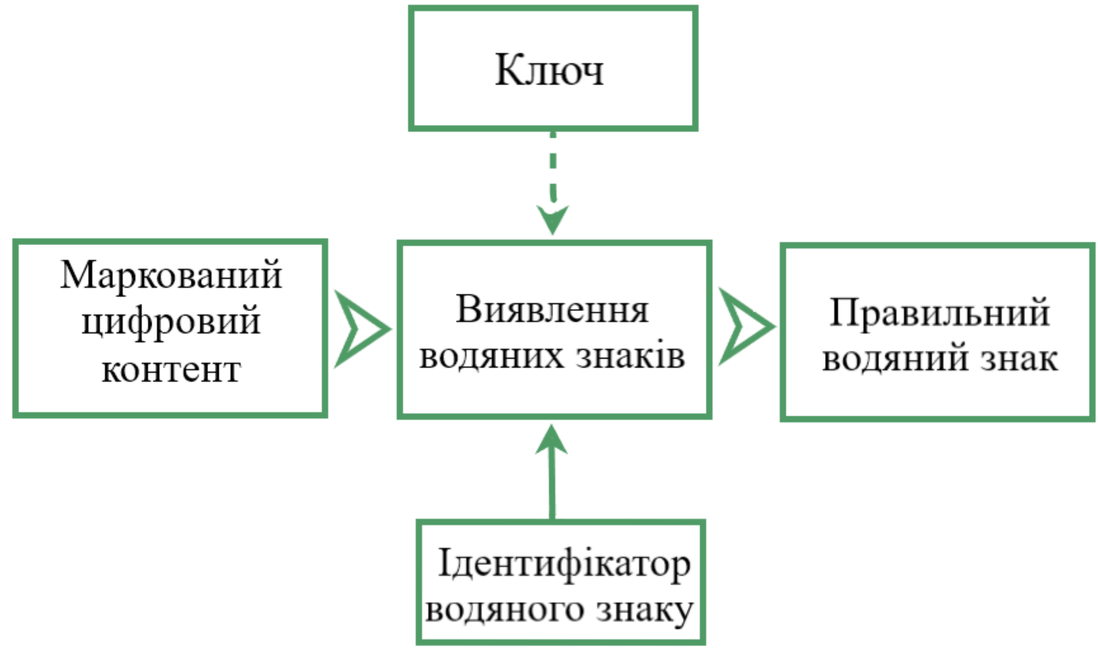


Рисунок 1.3 – Загальна схема виявлення ЦВЗ

Для вилучення водяного знаку використовується ключ разом з маркованим цифровим контентом для отримання вбудованого водяного знаку (рис. 1.4).

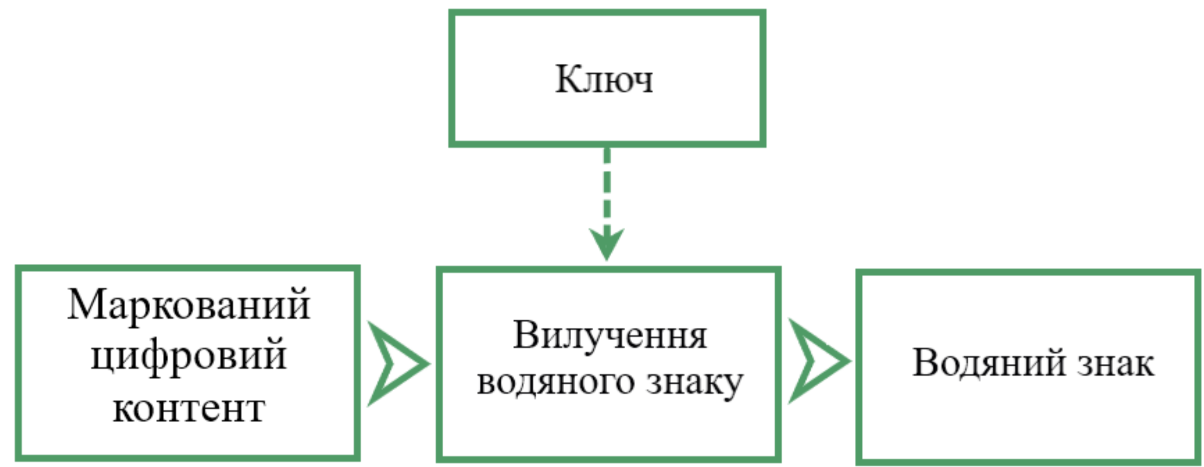


Рисунок 1.4 – Загальна схема вилучення ЦВЗ

Система цифрового водяного маркування складається з компонентів вбудовування та зчитування. Компонент вбудовування, відомий також як кодер,

вбудовує водяний знак у медіа-контент, використовуючи спеціальні алгоритми для зміни даних так, щоб водяний знак залишався непомітним для спостерігача. Компонент зчитування, або декодер, аналізує медіа-контент для виявлення вбудованого водяного знаку. Крім того, компонент зчитування можна поділити на дві частини: зчитування та вилучення водяного знаку. Функція зчитування визначає наявність водяного знаку, а функція вилучення водяного знаку вилучає його, якщо він був вбудований, відновлюючи початковий медіа-контент. Ці компоненти важливі для забезпечення безпеки та непомітності водяних знаків у цифровому середовищі, а також для визначення авторства інформації та контролю за її цілісністю [14].

Таким чином, розглянуто особливості ЦВЗ та виділено їхню важливу роль у сучасному цифровому середовищі для захисту авторських прав, відстеження походження та забезпечення цілісності інформації. Надалі буде проведений аналіз алгоритмів цифрового водяного маркування, які використовуються для вбудовування і вилучення водяних знаків з різних типів медіа-контенту. Особливий акцент буде зроблено на дослідження використання ЦВЗ для тривимірних моделей.

1.2 Аналіз найпоширеніших алгоритмів вбудовування ЦВЗ

Методи цифрового нанесення водяних знаків класифікуються з точки зору алгоритму, який використовується для вбудовування, оскільки вони використовують різні підходи до вставлення інформації в оригінальний цифровий контент. Ці алгоритми повинні бути досконало адаптовані до конкретного типу контенту, забезпечуючи оптимальний баланс між невидимістю водяного знаку та стійкістю до видалення чи модифікації.

Алгоритм вбудовування водяних знаків – це процес вбудовування даних в оригінальний контент для підтвердження інформації про право власності або авторські права. Загалом, методи нанесення водяних знаків класифікуються на основі різної області вбудовування водяного знаку на два класи: просторово-трансформаційні та частотні [15].

Методи нанесення водяних знаків у просторовій області зазвичай менш стійкі до таких атак, як стиснення і додатковий шум. Однак вони мають набагато меншу обчислювальну складність і зазвичай можуть витримати атаку обрізання, яку часто не вдається здійснити в частотній області. Також, можлива комбінація просторових водяних знаків і частотних водяних знаків для підвищення стійкості та зменшення складності реалізації [16].

Просторово-трансформаційні методи ЦВЗ вбудовують інформацію безпосередньо в просторовий домен даних, такий як пікселі або біти зображення. У цій категорії методів нанесення водяних знаків вбудовування ЦВЗ відбувається шляхом безпосередньої модифікації вихідного вмісту, тобто значень пікселів основного зображення або відео, оригінального контенту. Це може включати зміни в найменш значущих бітах RGB-каналів пікселів, а також використання різних атрибутів чорно-білих зображень для вбудовування водяних знаків. Обираючи просторово-трансформаційні методи, важливо враховувати, як зміни впливають на якість та видимість оригінального контенту, а також стійкість водяного знака до обробки чи атак [17].

Спочатку розглянемо метод найменш значущих бітів (LSB). У ньому водяний знак додається в найменш значущий біт кожного пікселя. Коли потрібно витягти інформацію, зчитується LSB кожного пікселя. Це дозволяє відновити вбудовану інформацію, оскільки зміни в найменш значущих бітах відображають приховану інформацію. Цей метод може бути застосований як до зображень, так і до відео, і забезпечує високу швидкість вбудовування, але він не дуже стійкий до поширених атак на обробку сигналів, що робить його вразливим до змін [18].

Адитивне нанесення водяних знаків у просторовій області передбачає вбудовування водяного знаку в зображення шляхом додавання псевдовипадкового шумового сигналу до значень пікселів. Ця техніка спрямована на зміну інтенсивності пікселів таким чином, щоб зміни були малопомітними для людського ока, але при цьому вбудований водяний знак можна було витягти за допомогою секретного ключа або спеціального алгоритму [19].

З метою підвищення непомітності, велике значення вбудовується не в один піксель, а в спеціальний блок пікселів. На рисунку 1.5. показано приклад блочного адитивного нанесення водяних знаків.

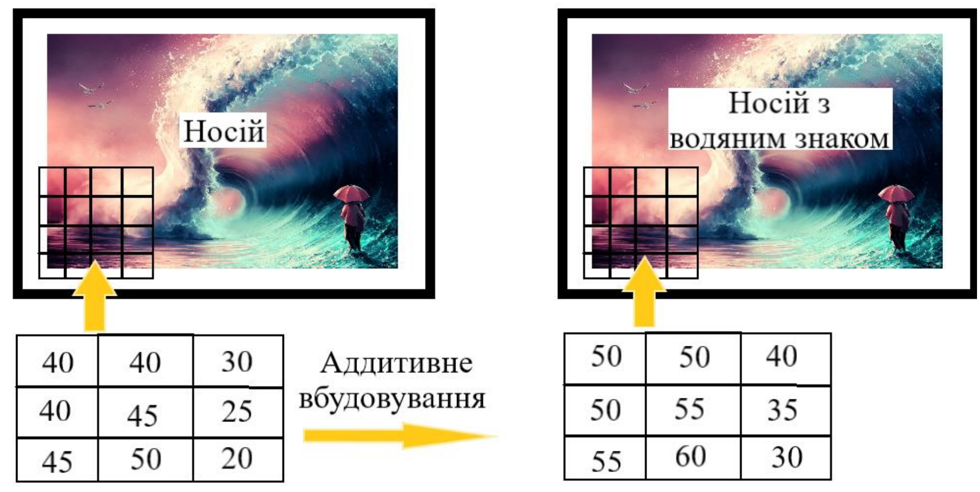


Рисунок 1.5 – Метод блочного адитивного водяного знаку в просторовій області [20]

Під час вбудовування водяного знаку обирається блок розміром 3 × 3, і застосовується коефіцієнт масштабування, який дорівнює 90. Потім величина цього коефіцієнта ділиться на 9 (розмір матриці). Навіть при великому значенні коефіцієнта масштабування зміни в значеннях пікселів залишаються помірною, оскільки розділова здатність пікселів не змінюється значно. Таким чином, отримавши зображення з водяним знаком, спостерігачу важко визначити вбудоване повідомлення, оскільки він не знає точного місцезнаходження блоку для вбудовування. Цей підхід сприяє підвищенню надійності даних з водяними знаками і збільшує їх стійкість до різних видів атак. Проте важливо правильно підібрати значення коефіцієнта масштабування відповідно до характеристик носія і вимог застосування адитивного водяного знаку, оскільки це також впливає на ємність водяного знаку [20].

Методи модуляції з розширеним спектром навмисно розподіляють енергію між різними конкретними частотами в часі. Такий підхід підвищує безпеку зв'язку, збільшує стійкість до перешкод і глушіння, а також запобігає легкому виявленню. У контексті нанесення водяних знаків на зображення алгоритм модуляції

розширеного спектра (SSM) вбудовує інформацію, поєднуючи оригінальне зображення з тонким псевдошумовим сигналом, модульованим доданим водяним знаком. Цей метод забезпечує безпечне вбудовування повідомлень у зображення, що ускладнює виявлення або втручання в процес нанесення водяних знаків [21].

Частотні методи ЦВЗ використовують перетворення сигналу для вбудовування інформації у частотний домен, такий як спектральні коефіцієнти зображення. Коли інформація вбудовується у частотний домен, це може бути менш помітно для людського сприйняття, оскільки зміни в спектрі можуть бути дрібними та розподіленими. Найпоширенішими перетвореннями є дискретне косинусне перетворення (DCT), дискретне перетворення Фур’є (DFT) і дискретне вейвлетне перетворення (DWT) [22].

Водяні знаки, які використовують DCT, подібні до водяних знаків просторової області, за винятком того, що замість зміни LSB пікселя бітової площини зображення змінюються частотні коефіцієнти. Такі водяні знаки стійкі до таких атак, як шум, стиснення, підвищення різкості та фільтрація. DCT дозволяє розкласти зображення на різні діапазони частот, що полегшує вбудовування інформації водяних знаків у діапазони частот зображення. Для вбудовування переважні смуги середніх частот. Це підвищує надійність проти атак, які в багатьох випадках можуть спотворювати вищі частоти. Крім того, вибираючи середні частоти, можна уникнути найбільш візуально важливих частин зображення, зосереджених на нижчих частотах [23].

Дискретне перетворення Фур'є (DFT) дозволяє перетворити зображення у набір його частотних складових. Воно має високу стійкість до геометричних атак, таких як обертання, масштабування, обрізання та зсуви.

DFT демонструє властивість інваріантності до зсувів. Зміщення об'єкта на зображенні впливає на фазовий опис зображення, але не впливає на його амплітудний опис. Це означає, що зміни в положенні об'єкта на зображенні не впливають на загальну інформацію про його структуру в частотному домені [24].

Дискретне вейвлет-перетворення (DWT) є потужним інструментом для вбудовування водяних знаків в цифрові зображення. Відзначається високою

стійкістю до геометричних та сигнальних атак. Цей метод дозволяє розбити зображення на різні частотні складові, що полегшує вбудовування інформації в різні частотні діапазони. Крім того, він враховує особливості людського зору, забезпечуючи точне відображення анізотропних особливостей зображення. DWT широко використовується для захисту від підробки та незаконного використання в цифрових зображеннях через вбудовування невидимих водяних знаків [25].

Опираючись на здійснений аналіз, можна зробити висновок, що водяні знаки у просторовій області мають нижчу складність алгоритму та вищу ефективність роботи, тоді як водяні знаки в частотній області мають вищу надійність. У галузі ЦВЗ важливо знайти хороший баланс між невидимістю, надійністю, роботою в режимі реального часу та ємністю.

Таким чином, враховуючи усі наведені особливості ЦВЗ та обрану тему роботи, далі буде здійснено дослідження використання ЦВЗ для тривимірних моделей та їх актуальність. Буде досліджено, як ці методи можуть бути застосовані для тривимірних моделей та як вони можуть вирішити конкретні виклики, пов'язані з темою роботи.

1.3 Дослідження застосування ЦВЗ для тривимірних моделей

Оскільки дана робота присвячена підвищенню стійкості водяного знаку, що вбудовується у тривимірну модель, спочатку буде розглянуто визначення та головні поняття тривимірних моделей для більш глибокого розуміння та аналізу особливостей вбудовування ЦВЗ.

Тривимірна модель, або 3D-модель – це математично згенероване або комп'ютерне представлення тривимірного об'єкта, яке описує його форму, розмір та структуру у тривимірному просторі. Ця модель може використовувати різні геометричні елементи, включаючи полігони, криві, поверхні та інші геометричні об'єкти [26].

Подібно до зображень, тексту, аудіо та відео, тривимірні моделі також стикаються з проблемами, такими як захист авторських прав, виявлення порушень тощо. Особливо це стає актуальним у зв'язку з розвитком спільного проектування

та віртуальних продуктів у мережевому середовищі. З швидким розвитком технологій мережевого мультимедіа, дослідження технологій забезпечення інформаційної безпеки на основі методів приховування інформації та технологій цифрового водяного знаку є актуальною темою досліджень. У порівнянні із зображеннями, текстом і аудіо, захист 3D-моделей, зокрема полігональних, стає важливішим та вимагає вивчення та впровадження нових методів захисту від незаконного використання, копіювання та модифікації [27].

Полігональна модель – це конкретний тип тривимірних моделей, який використовує полігони (геометричні фігури з трьох або більше вершин), з'єднані ребрами, для утворення поверхні об'єкта (рис.1.6). Полігональні моделі відзначаються гнучкістю та ефективністю рендерингу, забезпечуючи легку обробку та модифікацію у багатьох галузях, включаючи ігрову індустрію, кіно, дизайн, та архітектуру.

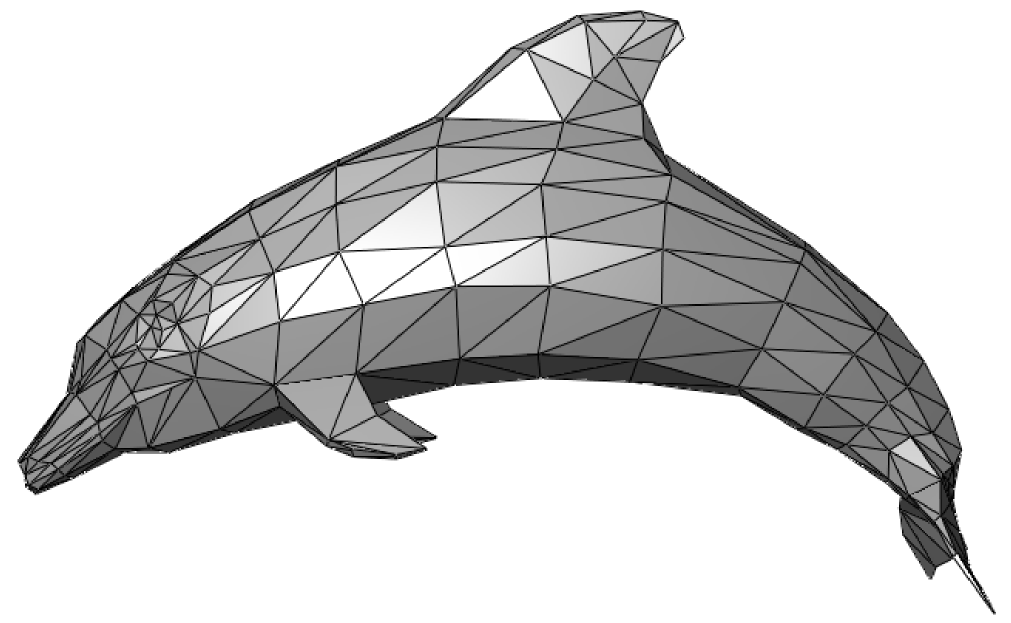


Рисунок 1.6 – Приклад полігональної моделі [28]

Полігон – це геометричний об'єкт у тривимірному просторі, який складається з трьох або більше вершин, які з'єднані ребрами. Полігони використовуються для створення поверхонь 3D-моделей.

Полігональна сітка – це набір вершин, ребер і граней, які визначають форму багатогранного об’єкта в тривимірному просторі. Вершини (точки) вказують на

конкретні точки у просторі, ребра з'єднують вершини, а грані утворюють поверхні об'єкта, обмежуючи його об'єм та форму [29].

Збільшення кількості полігонів у тривимірних моделях може покращити їхню деталізацію та гладкість поверхні (рис. 1.7).

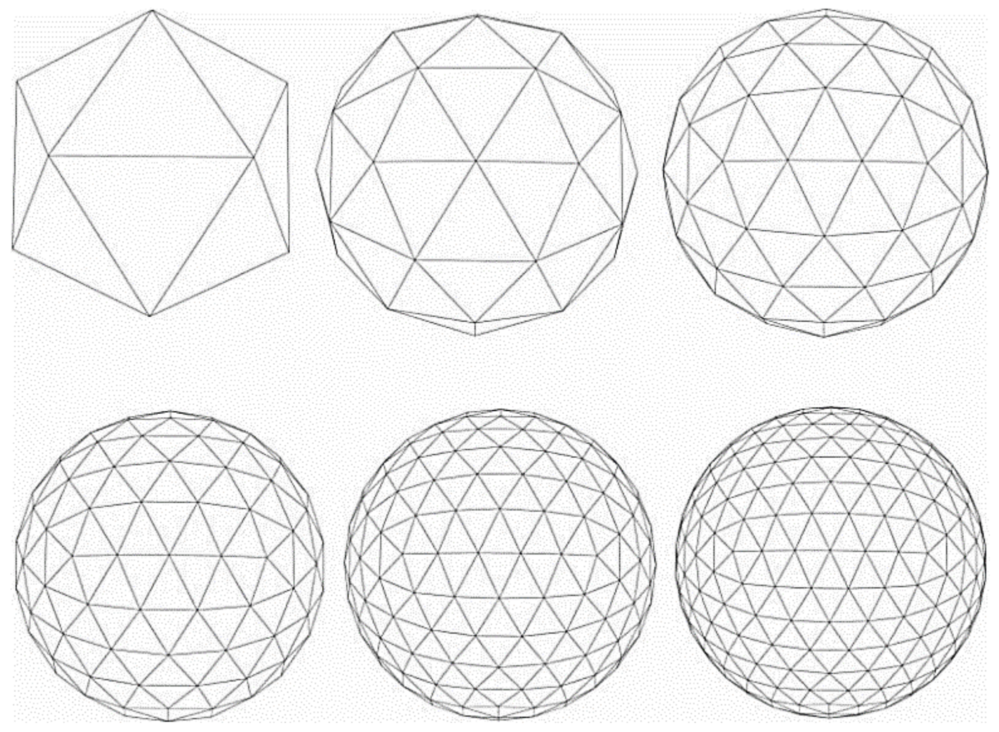


Рисунок 1.7 – Демонстрація впливу кількості полігонів на вигляд тривимірних моделей [30]

Полігони розбиваються на більшу кількість менших полігонів, зазвичай за допомогою спеціальних алгоритмів. Це збільшує кількість деталей, які можуть бути відображені на поверхні моделі, дозволяючи отримати більш реалістичний вигляд об'єкта.

Опираючись на особливості побудови тривимірних моделей, розглянемо методи вбудовування ЦВЗ у них.

Один з найперших та найбільш відомих – метод вбудовування видимого водяного знаку даних про авторське право в 3D полігональні моделі, або алгоритм Обучі. Згідно з описаною раніше класифікацією типів ЦВЗ, алгоритм нанесення водяних знаків доктора Обучі можна віднести до методу просторової області. Він включає в себе модифікацію або координат вершин, або їх зв'язності, або обидвох цих параметрів для внесення водяного знаку в дані полігону. Автори також

розглядали різні схеми впорядкування геометричних або топологічних елементів для вбудовування та вилучення водяного знаку. На основі цих досліджень було розроблено кілька алгоритмів вбудовування, таких як алгоритм квадратичної подібності трикутників, алгоритм вбудовування тетраедричного об'ємного відношення та алгоритм послідовності символів, які відшаровуються від трикутника. Вбудований водяний знак стійкий до 3D афінного перетворення та обрізання [31-32].

Проте, метод має свої слабкі сторони. Незважаючи на стійкість до певних афінних перетворень, цей метод може бути вразливим у випадках комплексних атак, які використовують спеціалізовані алгоритми для вилучення водяних знаків. Вбудовування водяного знаку може збільшити об'єм даних та ускладнити обробку та передачу інформації, особливо в великих тривимірних моделях. Модифікація вершин чи їх зв'язності може призвести до втрати якості самої моделі, зокрема у випадку великої кількості водяних знаків або при використанні агресивних методів вбудовування [33].

Метод нанесення водяних знаків на сітку з різною роздільною здатністю, спеціально розроблений для роботи з напіврегулярними сітками зі зв'язністю розбиття. Алгоритм вбудовує водяний знак, модифікуючи вейвлет-коефіцієнти 3D-моделей, отриманих шляхом декомпозиції хост-сітки за допомогою алгоритму, запропонованого Ленсбері (рис. 1.8) [34].

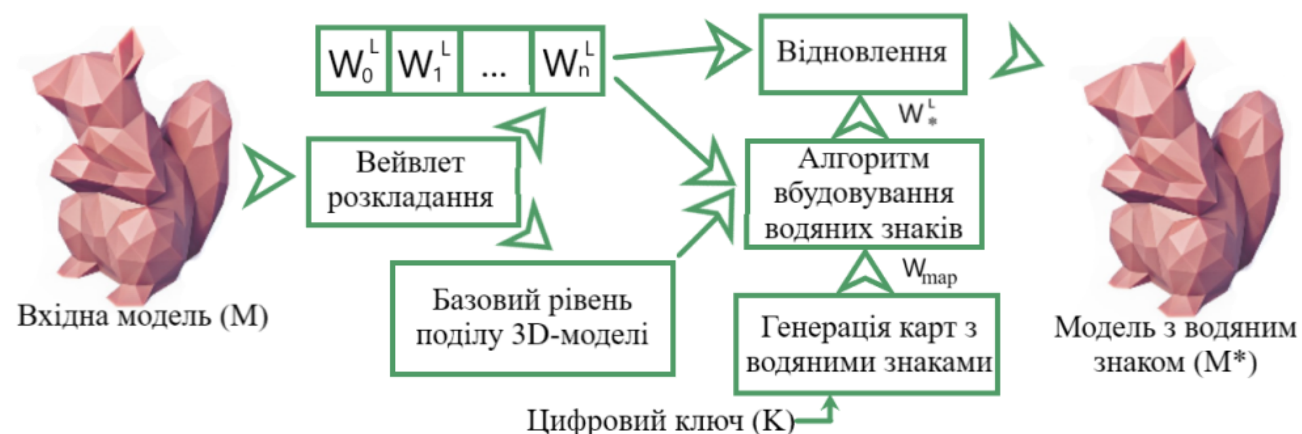


Рисунок 1.8 – Схема алгоритму вбудовування водяного знака [34]

Особлива увага приділяється тому, щоб алгоритм вбудовування зберігав візуальну цілісність моделей. Відновлення водяного знаку відбувається за допомогою кореляційного детектора, розробленого відповідно до теорії статистичного виявлення. Їх система не вимагає, щоб вихідна немаркована сітка була доступна на детекторі, що робить систему набагато гнучкішою і легко адаптованою до практичних застосувань [35]. Алгоритм вбудовує водяний знак, змінюючи положення вершин сітки в перетвореній області, тобто вейвлет-області. Представлена система 3D-водного маркування вбудовує цифровий код у напіврегулярну сітку зі зв'язністю підрозділів, використовуючи фреймворк з мультироздільною роздільною здатністю. Даний алгоритм, належить до категорії водяних знаків, які можна виявити. Алгоритм вбудовування водяних знаків працює за трьома параметрами (секретний ключ, рівень роздільної здатності та коефіцієнт, що визначає стійкість водяного знаку) [36].

Важливою проблемою в нанесенні водяних знаків на полігональні сітки є можливість різноманітних атак на ці сітки. Варіант водяного маркування, описаний вище є досить непродуманим для напіврегулярних сіток зі зв'язністю розбиття та не враховує атаки, які можуть змінювати саму структуру сітки. Замість цього, його стійкість показує гарний результат щодо конкретних видів атак, таких як адитивний шум, низькочастотна фільтрація, геометричні зміни, обрізання та їх комбінація [37].

Ще один метод був розроблений для оптимізації 3D нанесення водяних знаків для мінімізації спотворень поверхні за допомогою методу оптимізації Левенберга-Марквардта для вершин, представлених у сферичних координатах. Запропонований метод є статистичним, сліпим і робастним методом нанесення 3D водяних знаків. Водяний знак вбудовується в гістограми відстаней від центру об'єкта до вершин на його поверхні шляхом незначного зміщення розташування вершин [38].

Цей метод розглядає функцію похибки, що складається з трьох компонент, які вимірюють спотворення відносно вихідної поверхні, поверхні з водяним знаком (оновленої поверхні) та евклідової відстані від вихідного розташування вершин. Ця функція похибки забезпечує як мінімальне спотворення відносно вихідної

поверхні, так і забезпечує гладкість поверхні об'єкта, отриманої після нанесення водяного знаку [38].

Алгоритм нанесення водяних знаків трикутною сіткою на основі 3D-сегментації – це доволі новий підхід до нанесення водяних знаків на 3D-зображення, який поєднує в собі три класи вставки. Ця комбінація дозволяє отримати вигоду з численних переваг кожного класу. Це досягається шляхом сегментації 3D-зображення на нерафіновані області з різними значеннями кривизни та адаптації кожної області до класу вставки для отримання максимальної невидимості знаку [39].

Методи використовують дві можливі області вставки: просторову (геометричну та топологічну) та спектральну області.

Цей алгоритм розбиває тривимірну сітку на зв'язані групи трикутників, які мають майже постійну кривину та точно визначені межі. Спочатку для кожної вершини обчислюється кривина, що вказує на ступінь зігнутості поверхні. Потім вершини поділяються на групи відповідно до їх кривини[39].

Далі застосовується алгоритм, який об'єднує трикутники в групи, враховуючи їхні групи вершин. Він заповнює порожні місця між цими групами, дотримуючись певних критеріїв. Наприклад, граничні трикутники мають бути добре визначені та з'єднані [40].

На завершення, використовується граф суміжності груп, який скорочується для об'єднання схожих груп за різними критеріями, такими як схожість кривини, розмір чи спільний периметр. Цей метод дозволяє використовувати переваги різних типів водяних знаків і забезпечує надійність та ефективність у визначенні регіонів для вбудовування водяних знаків [40].

Слабкі сторони цього алгоритму включають його чутливість до шуму, витратність обчислень, залежність від параметрів, відсутність універсальності, потребу в великій кількості даних та обмежену стійкість до складних геометричних структур [41].

Кожен з розглянутих методів вбудовування ЦВЗ у тривимірну модель має свої особливості та слабкі сторони. Так як розробка таких алгоритмів є доволі новим

напрямком захисту цифрової власності в тривимірних моделях, подальше дослідження вбудовування ЦВЗ на основі реберних вершин є актуальним та доречим.

1.4 Аналіз атак на ЦВЗ тривимірних моделей

Для тривимірних цифрових медіа використання водяних знаків стало важливим інструментом захисту прав власності та автентичності. Незважаючи на те, що водяні знаки є надійною технологією з різноманітним застосуванням, вони є вразливими до різноманітних атак, як на самі водяні знаки, так і на цифровий контент, який вони захищають. Розглянемо основні типи атак на водяні знаки тривимірних моделей.

Атака на водяні знаки тривимірних моделей – це навмисна спроба модифікувати або порушити цілісність, видимість або можливість виявлення водяного знака, вбудованого в 3D-модель. Зловмисник намагається або повністю вилучити водяний знак, або змінити його так, що виявити його стає вкрай складно або навіть неможливо. Метою є унеможливлення ідентифікації законного власника та незаконне використання моделі [42].

Загалом можна виділити декілька типів атак на ЦВЗ тривимірних моделей (рис. 1.9).

Рисунок 1.9 – Типи атак на ЦВЗ тривимірних моделей

Геометричні атаки – це категорія атак, які спрямовані на зміну геометричних аспектів тривимірної моделі, в першу чергу через збурення в положеннях вершин. Ці атаки можуть включати різні перетворення, такі як обертання, трансляція,

масштабування та інші геометричні модифікації, такі як зсув або нерівномірне масштабування. Ці перетворення часто називають перетвореннями подібності. Різноманітність можливих атак ускладнює процес вбудовування водяних знаків у тривимірні моделі порівняно із застосуванням їх до зображень та відео [43].

Геометричні атаки становлять значний виклик у сфері захисту цифрових активів, оскільки вони можуть маніпулювати просторовими властивостями 3D-моделі, не обов'язково погіршуючи її загальну візуальну якість. Це означає, що зловмисники можуть змінювати геометрію моделі, намагаючись зберегти її візуальну привабливість для кінцевого користувача, що ускладнює виявлення водяного знаку.

Геометричні атаки в системах цифрового водяного маркування тривимірних моделей можна розділити на глобальні геометричні атаки та локальні геометричні атаки. Глобальні геометричні атаки охоплюють такі перетворення, як обертання, масштабування, трансляція, спотворення розтягування (зсув), а також інші афінні перетворення та проекційні перетворення. Ці перетворення можуть суттєво вплинути на цілісність вбудованих водяних знаків. На відміну від локальних геометричних атак, які націлені на конкретні області, глобальні геометричні атаки змінюють загальну форму і структуру всієї 3D-моделі [44].

Також, до глобальних геометричних атак віднесемо додавання випадкового шуму, згладжування, покращення та стиснення (часто за допомогою квантування). Ці методи атак впливають на всю тривимірну структуру моделі, змінюючи її геометричні та просторові характеристики. Хоча вони зазвичай застосовуються в анімації та спецефектах, вони можуть серйозно ушкодити вбудовані водяні знаки. Ці атаки спираються на те, що тривимірні моделі розглядаються як сигнали у тривимірному просторі, і можуть впливати на їхню геометричну цілісність та безпеку вбудованих водяних знаків [45].

Для 3D-моделей локальні геометричні атаки можуть включати більш складні маніпуляції, такі як деформація вершин, нерівномірне масштабування певних частин, зміна текстури або цілеспрямоване спрощення сітки, коли певні області моделі навмисно спрощуються або модифікуються. Ці атаки спеціально

пристосовані до унікальних геометричних властивостей тривимірних моделей, впливаючи на їхню просторову цілісність і потенційно компрометуючи вбудовані водяні знаки [46].

Геометричні атаки зазвичай спрямовані не на пряме видалення ЦВЗ, а на зміну його розташування шляхом внесення просторових часових спотворень.

Розглянемо наступну категорію атак. Атаки на зв'язок або Connectivity Attacks – це атаки, спрямовані на порушення зв'язків, структури чи цілісності тривимірних моделей. Ця назва вказує на те, що атаки спрямовані на руйнування або зміну зв'язків між компонентами моделі, включаючи їхню структуру та форму. Ці атаки включають в себе обрізання, повторення, поділ та спрощення геометричних компонентів моделі, таких як вершини, ребра та поверхні [47].

Атака обрізання в тривимірних моделях відбувається, коли зловмисники намагаються вирізати або приховати водяний знак, видаляючи або змінюючи частини моделі, де він розміщений. Вони можуть видалити певні точки, лінії або поверхні, які містять водяний знак, або змінити їхню форму, щоб зробити його менш помітним. Цей тип атаки є унікальним і може розглядатися як геометрична атака, оскільки має схожі ефекти з локальними спотвореннями [48].

Атаки повторення включають намагання створити дублікати окремих частин моделі, що може призвести до неправильних з'єднань між цими дубльованими об'єктами. Атаки поділу полягають у розділенні моделі на менші частини. Це може спричинити втрату зв'язку між розділеними компонентами, особливо якщо не враховані правила з'єднаності. Атаки спрощення використовуються для скорочення кількості деталей у моделі, знижуючи кількість вершин, ребер та поверхонь. Це може вплинути на точність форми та розміщення компонентів. Ці атаки можуть серйозно пошкодити водяні знаки в тривимірних моделях. Повторення може призвести до неправильних з'єднань, порушуючи цілісність водяного знака. Поділ може спричинити втрату зв'язку між розділеними частинами водяного знака, зробивши його менш впізнаваним. Спрощення може призвести до втрати деталей у водяному знаку, зменшуючи його надійність та видимість [49].

Остання виділена категорія атак на ЦВЗ тривимірних моделей – топологічні атаки. Ці атаки включають в себе різноманітні методи втручання у топологічну структуру моделі, такі як видалення вершин, зміни у з'єднаннях між вершинами, або перерозподіл поверхонь. Видалення або модифікація вершин може порушити природні зв'язки між елементами моделі, створюючи неправильно з'єднані області. Зміни у з'єднаннях можуть призвести до втрати зв'язку між компонентами, ускладнюючи виявлення водяного знака. Такі атаки можуть також включати розділення поверхонь чи зміни границь моделі, створюючи деформовані або роз'єднані частини [50].

Таким чином, у даному підрозділі було проаналізовано головні типи атак. Можна зробити висновок, що геометричні атаки – це один з найбільш різноманітних типів атак, які несуть значні ризики для ЦВЗ. Тому розробка та удосконалення методів, стійких до даного типу атак є надзвичайно актуальним завданням.

1.5 Висновок до розділу 1 та постановка задач

Отже, у даному розділі здійснено загальний аналіз стеганографічного захисту інформації на основі ЦВЗ.

У ньому розглядаються види та особливості ЦВЗ, а також визначено їхню роль у сучасному цифровому середовищі. Далі проводиться детальний аналіз найпоширеніших алгоритмів вбудовування ЦВЗ та їх характеристики. Також досліджується використання ЦВЗ для тривимірних моделей, виявляючи ефективність ЦВЗ та потенційні області їх використання. Окрема увага приділяється аналізу можливих атак на ЦВЗ тривимірних моделей.

Після ретельного аналізу теоретичного матеріалу та беручи до уваги мету та актуальність теми роботи, поставлено задачі для подальшого наукового дослідження та розробки:

– удосконалити метод вбудовування невидимого ЦВЗ на основі 3D-розпізнавання реберних вершин для тривимірних моделей за рахунок підвищення його стійкості від геометричних операцій перетворення;

– розробити алгоритм роботи програмного засобу, використовуючи вдосконалений метод вбудовування невидимого ЦВЗ;

– здійснити його програмну реалізацію;

– розробити приклад реалізації удосконаленого алгоритму;

– здійснити аналіз стійкості удосконаленого алгоритму та провести аналіз отриманих результатів;

– економічно обґрунтувати доцільність впровадження розробки на основі вдосконаленого методу вбудовування ЦВЗ для тривимірних моделей.

Реалізація поставлених завдань є ключовим кроком для досягнення головної мети цієї роботи – здійснення підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей.

2 ПІДВИЩЕННЯ СТІЙКОСТІ НЕВИДИМОГО ВОДЯНОГО ЗНАКУ ВІД ГЕОМЕТРИЧНИХ ОПЕРАЦІЙ ВДОСКОНАЛЕННЯМ МЕТОДУ ВБУДОВУВАННЯ НА ОСНОВІ 3D-РОЗПІЗНАВАННЯ РЕБЕРНИХ ВЕРШИН

В даному розділі проведено аналіз алгоритму вбудовування та виявлення водяного знаку за допомогою дискретного косинусного перетворення (DCT) [51]. Здійснено його удосконалення за допомогою методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей. Крім того, розроблено алгоритм роботи програмного продукту та представлено висновки отриманих результатів.

2.1 Алгоритм вбудовування та розпізнавання водяних знаків

Досліджуваний алгоритм [51] заснований на пошуку опорних точок для вбудовування водяного знаку, які будуть сприяти поліпшенню збереження інформації після різних видів атак на сітчасту модель. Даний алгоритм використовуватиметься як основа алгоритму для вбудовування водяного знаку у 3D модель. Розглянемо його більш детально.

Перший етап алгоритму – це пошук відповідних точок для вбудовування водяного знаку. Спочатку потрібно побудувати окіл навколо вершин. Нехай сіткова модель M є сукупністю вершин. Вершина сітчастої моделі представлена у вигляді $v_i \in M(i = 1,2,\dots,N)$, а її координати множна визначати за допомогою вектора $\vec{v_i}(i = 1,2,\dots,N)$. В такому випадку околицю вершини можна визначити за формулою (2.1):

$$\mathrm{N}(\mathrm{v_i}) = \{\mathrm{v_j} | |\mathrm{v_i v_j}| > 0, \mathrm{i} = 1,2,\dots,\mathrm{N}; \mathrm{j} = 1,2,\dots,\mathrm{N}\} \tag{2.1}$$

де $|v_i v_j| > 0$ – відношення, що показує зв’язок між вершиною v_iта вершиною v_j, N – загальна кількість вершин у сітковій моделі.

Для позначення кільцевої околиці визначається α-кільцеву ($\alpha \neq 1$) околицю вершини v_i. Дане значення регулює масштаб округу навколо вершини v_i. В даному

випадку для уникнення дублювання вбудовування інформації масштаб буде обрано за одиницю. Приклад вершини з околом від однокільцевої до трьохкільцевої показаний на рисунку 2.1.

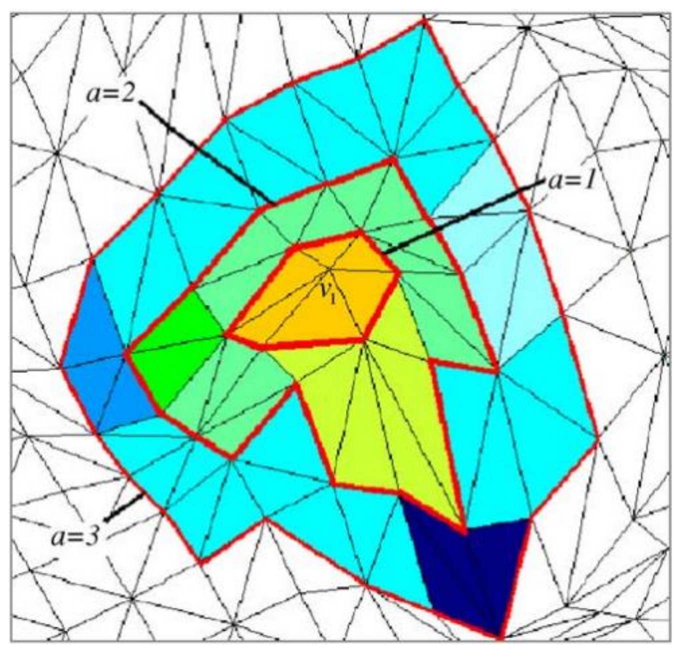


Рисунок 2.1 – Вершина v_i з околом від однокільцевої до трьохкільцевої [51]

Після утворення околу потрібно розрахувати нормаль напрямку для обраної вершини. Ділянки, які підходять для вбудовування інформації – це ділянки, для яких характерні різкі особливості. Для знаходження таких ділянок потрібно визначити напрямок нормаль для кожної вершини. Для визначення напрямку нормалі до вершини v_i потрібно наближено обчислити середньозважені значення її однокільцевих сусідніх вершин. Дане значення можна обчислити за допомогою формули (2.2):

$$\overrightarrow{n_{v_i}} = \frac{\sum_{j=1}^{N_i} v_j \in N(\mathrm{v_i})(\overrightarrow{v_j} - \overrightarrow{v_m})}{N_i} \tag{2.2}$$

де N_i – загальна кількість вершин що дотичні до вершини $\mathrm{v_i}$,

$\overrightarrow{v_m}$ – середній вектор від усіх вершин до центру тривимірної сітчастої моделі,

$\overrightarrow{n_{v_i}}$ – це нормальний вектор $\mathrm{v_i}$.

Для кращого розумінні, напрямки нормалей $\mathrm{v_i}$ та її однокільцевої вершини v_j показані на рисунку 2.2.

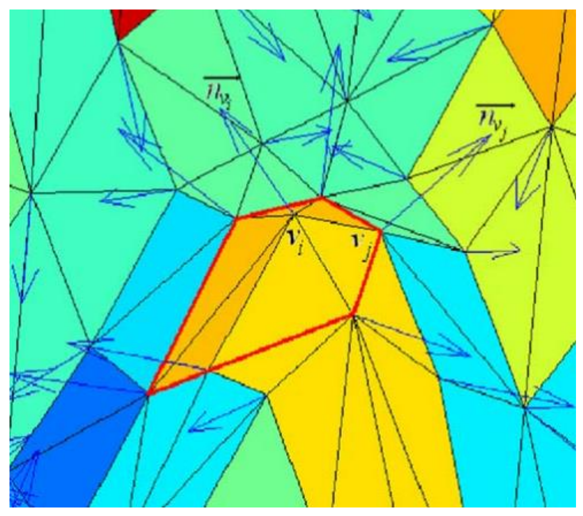


Рисунок 2.2 – Напрямки нормалей v_i та v_j [51]

Після визначення напрямку нормалі до кожної вершини, області 3D-сіткової моделі можна оцінити за наступною формулою (2.3):

$$D(v_i) = \sum_{j=1}^{N_i} v_j \in N(v_i) \cos^{-1}(\overrightarrow{n_{v_i}}, \overrightarrow{n_{v_j}}) \quad (2.3)$$

де $\cos^{-1}(\overrightarrow{n_{v_i}}, \overrightarrow{n_{v_j}})$ позначає кут між напрямками нормалей вершин v_i та v_j.

Визначаючи даний кут можна визначити оцінку різкої зміни площини. Даний процес показаний на рисунку 2.3, де однокільцева околиця вершини v_i використовується для оцінки різкої зміни площі, та кут між v_i та v_i.

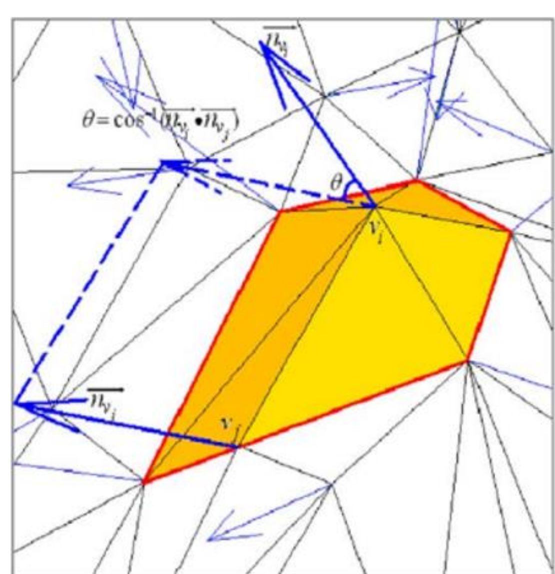


Рисунок 2.3 – Однокільцева околиця вершини v_i використовується для оцінки різкої зміни площини [51]

Значення $D(v_i)$ буде змінюватися в залежності від масштабування околу навколо опорної точки. Більше того, оскільки вплив зашумлених точок в околі може бути згладжений при збільшенні масштабу околу, вибір відповідного значення α є не тільки дуже важливим для оцінки областей різких змін, але й дуже важливим для покращення стійкості сіткової моделі до додавання шумових атак.

Але даний алгоритм пошуку $D(v_i)$ є досить примітивним та сильно залежить від значення α. До того ж чим більше це значення, тим менше опорних вершин буде знайдено, що є досить сильним недоліком даного алгоритму.

Після визначення усіх значень D відбувається їхнє впорядкування з якого перші N_f вершин з найбільшим значенням $D(v_i)$ обираються у якості особливих точок, що надалі будуть позначатися як $V_n(n = 1,2, \dots, N_f)$.

Далі відбувається процес розбиття фігури для тривимірних моделей. Кожна точка, що була обрана як характерна точка використовується як центроїд для ділянки Вороного. Надалі вимірюється геодезична відстань між кожною вершиною і кожною характерною точкою, усі вершини, що мають мінімальну геодезичну відстань до центроїда Вороного будуть належати до його ділянки. Геодезична відстань обчислюється наступною формулою (2.4):

$$\mathrm{P}(v_i)_n = Min\{d_g(v_i, V_n), n = 1,2, \dots, N_f\}, i = 1,2, \dots, N \tag{2.4}$$

де $d_g(v_i, V_n)$ позначає функцію вимірювання для обчислення геодизичної відстані між вершиною v_i та характерною або опорною точкою V_n.

Функція $Min\{\}$ виконує пошук мінімального значення, яке використовується для знаходження опорної точки, що має мінімальну геодезичну відстань з v_i. $\mathrm{P}(v_i)_n$ означає, що вершина v_i належить до ділянки Вороного $P_n(n = 1,2, \dots, N_f)$. Оскільки було вибрано N_f характерних точок, сіткову модель можна розбити на N_f ділянок Вороного.

Останнім етапом підготовки до вбудовування водяного знаку є генерування зображення діапазону для кожної ділянки Вороного.

Для кожної ділянки Вороного задається опорна квадратна площина, що є перпендикулярною до напрямку нормалі до центроїда V_n. Даний напрямок позначається як $\overrightarrow{n_{v_n}}$ і визначається формулою (2.2). для визначення площини, кожна вершина ділянки Вороного проектується на опорну площину вздовж напрямку $\overrightarrow{n_{v_n}}$. Надалі задається опорна квадратна площина яка покриває усі точки проекцій вершин ділянки Ворогого. Після створення площини, на ній потрібно задати сітку x-y.

Для того щоб задати сітку x-y потрібно знайти центроїд V_l який буде мати максимальну геодезичну відстань з центроїдом V_n. Це розраховується за формулою (2.5):

$$V_l = Max\{d_g(V_n, V_k), k = 1,2, \dots, N_f\} \quad (2.5)$$

де $Max\{\}$ – функція пошуку максимального значення.

Далі потрібно з'єднати центроїди V_n і V_k, та спроектувати вектор $\overrightarrow{V_n V_l}$ на опорну площину вздовж напрямку нормалі $\overrightarrow{n_{v_n}}$ для отримання вектору $\overrightarrow{V`_n V`_l}$.

Таким чином напрямок осі координат можна визначити вздовж напрямку вектору $\overrightarrow{V`_n V`_l}$, а перпендикуляр до напрямку осі x як напрямок осі y.

Останнім кроком буде розбиття координатної осі x на рівномірні інтервали N_x, а координатну вісь y на рівномірні інтервали N_y.

Такий чином буде утворена система координат x-y у квадратній площині ділянки Вороного.

Кожна точка перетину сітки x-y приймається за опорну точку, яку можна позначити за формулою (2.6):

$$r(x_n, y_n)((x_n, y_n) \in \{(0,0), (0,1), \dots (N_x - 1, N_y - 1)\}) \quad (2.6)$$

Надалі обчислюється відстань від кожної опорної точки $r(x_n, y_n)$ до поверхні сітчастої моделі. Для цього використовується техніка розбиття простору.

Під час виконання даної техніки потрібно перетворення двовимірних координат x-y кожної опорної точки $r(x_n, y_n)$ у тривимірні координати простору сіткової моделі.

Далі проводиться промінь від кожної опорної точки до поверхні сітчастої моделі вздовж напрямку $\overrightarrow{n_{v_n}}$ для отримання точки перетину з поверхнею сітчастої моделі. Наділі обчислюється відстань між кожною опорною точкою та відповідною точкою перетину в координатах тривимірного простору.

Даний процес показаний на рисунку 2.4.

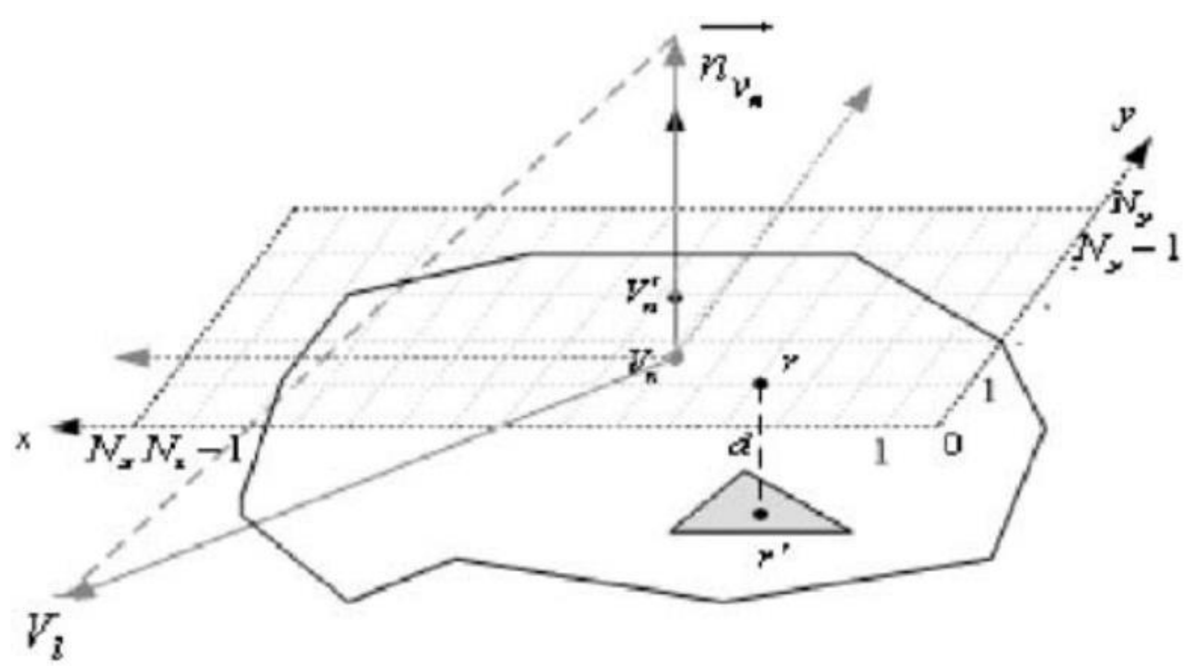


Рисунок 2.4 – Відстань між опорною точкою і точкою її перетину [45]

Після вимірювання відстані між кожною опорною точкою та поверхнею 3D-сіткової моделі, використовується метод лінійної нормалізації для перетворення цих відстаней у діапазон від [0 до 255]. Це дозволяє створити матрицю розміром $N_x \times N_y$, яку також називають зображенням діапазону. Оскільки сіткову модель розбито на N_f фрагментів, генерується N_f зображень діапазону.

Після отримання зображень діапазону виконується процес вбудовування водяного знаку.

Водяний знак представляє з себе масив з L випадкових чисел і позначається $W = \{w1, w2, \ldots, w_L\}^T$, що вибираються з гаусівського розподілу з нульовим

середнім значенням і дисперсією 1. Послідовність водяних знаків багаторазово вбудовується в кожну ділянку Вороного, це робиться для того, щоб уникнути втрат від атак обрізання. Даний алгоритм багаторазово вбудовує водяний знак у зображення діапазону кожної ділянки Вороного, а відповідні вершини кожної ділянки збурюються для генерації моделі з водяним знаком. Оригінальні зображення діапазону моделі зберігаються для виявлення водяних знаків.

Процес вбудовування водяного знаку включає в себе DCT. Спочатку зображення діапазону розбивається на 8 блоків, до кожного з блоків застосовується операція DCT. Надалі з кожного блоку DCT вибирається десять високочастотних коефіцієнтів для вбудовування водяного знаку за формулою (2.7):

$$D_{nw} = D_n + \beta \cdot W \tag{2.7}$$

де D_n – вибрані високочастотні коефіцієнти з оригінального зображення діапазону, β – сила вбудовування, а D_{nw} – високочастотні коефіцієнти з водяним знаком.

Далі відбувається обернене дискретне косинусне перетворення (IDCT) для кожного блоку, що застосовується для створення зображення діапазону з вбудованим водяним знаком.

Після внесення водяного знаку до зображення діапазону ділянки P_n, деякі значення пікселів цього зображення можуть зазнати змін. Це призводить до змін у відстані між опорними точками і відповідними точками їх перетину. Для відображення цих змін необхідно внести відповідні корекції до позицій вихідних вершин моделі, щоб вони відповідали змінам у зображенні та відстанях між ними.

На відстані між опорною точкою $r(x_n, y_n)$ та поверхнею сітчастої моделі впливають вершини однокільцевої околиці відповідної точки перетину r`. Проте, для зміни відстані, потрібно змінити лише одну вершину околиці. Щоб знайти цю вершину, кожна вершина ділянки P_n проектується на опорну площину, і вершина, яка знаходиться найближче до опорної точки $r(x_n, y_n)$, є тією вершиною, яку потрібно змінити, щоб скорегувати відстань між опорною точкою та поверхнею сітчастої моделі. Для коригування вершина моделі підіймається або опускається

вздовж напрямку $\overrightarrow{n_{v_n}}$, це реалізує зміну відстані відповідно до зображення діапазону з водяними знаками. Після завершення модифікації вершин кожної околиці буде отримано сітчасту модель з водяними знаками.

Загальний алгоритм вбудовування водяного знаку зображено на рисунку 2.5.

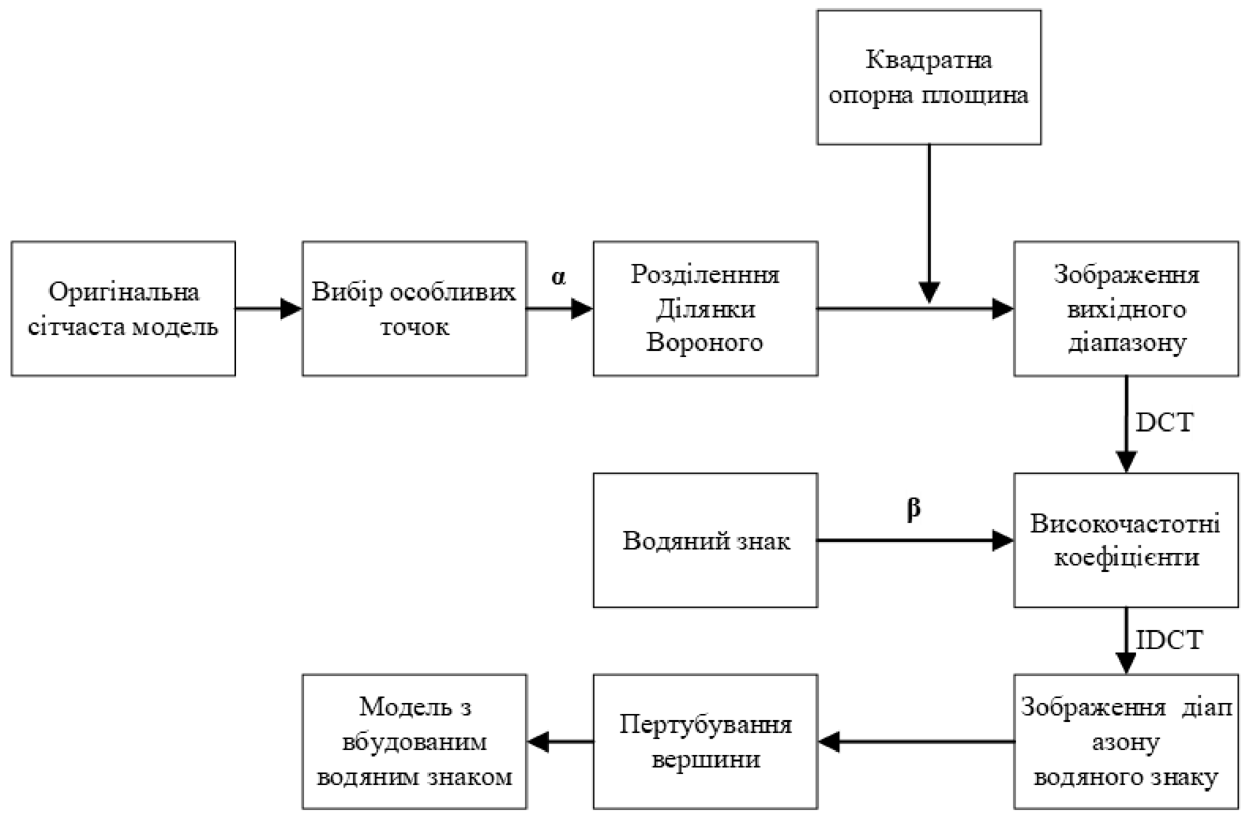


Рисунок 2.5 – Загальний алгоритм вбудовування водяного знаку в сітчасту модель

Процес виявлення водяного знаку є зворотнім до його вбудовування. Процес створення зображень діапазону з водяними знаками подібний до процесу вбудовування водяних знаків. Для виявлення водяних знаків використовуються вихідні зображення діапазону, які були отримані під час процесу вбудовування водяних знаків.

Операція DCT застосовується до оригінального зображення діапазону P_n, а також до зображення діапазону P_n з вбудованим водяним знаком. Це дозволяє отримати високочастотні коефіцієнти, які можуть бути використані для виділення або вилучення водяного знаку з зображення. Водяний знак з області P_n отримується формули (2.8).

$$W_n^* = \frac{D_{nw}^* - D_n}{\beta} \tag{2.8}$$

де D_{nw}^* – високочастотні коефіцієнти водяного знаку, отримані із зображення діапазону області P_n з водяним знаком,

D_n – високочастотні коефіцієнти, отримані із вихідного зображення діапазону області P_n,

W_n^* – водяний знак, витягнутий із області P_n.

Для отримання остаточного водяного знаку w^* використовується наступна формула (2.9):

$$w_i^* = \frac{\sum_{n=1}^{N_f} w_{in}^*}{N_f}, i = 1,2,\dots,L \tag{2.9}$$

де w_{in}^* – і-тий біт водяного знаку, витягнутий з області P_n,

w_i^* – і-тий біт кінцевого водяного знаку, L це довжина водяного знаку.

Для порівняння водяного знаку w та отриманого з сіткової моделі водяного знаку w^* потрібно проаналізувати, обчисливши їх коефіцієнт кореляції, що обчислюється за наступною формулою (2.10):

$$c(w, w^*) = \frac{\sum_{i=1}^{L}(w_i - \overline{w})(w_i^* - \overline{w^*})}{\sqrt{\sum_{i=1}^{L}(w_i - \overline{w})^2}\sqrt{\sum_{i=1}^{L}(w_i^* - \overline{w^*})^2}} \tag{2.10}$$

де $\overline{w}$ – середнє значення оригінального водяного знаку, а $\overline{w^*}$ – середнє значення вилученого водяного знаку.

Можливе значення $c(w, w^*)$ – число від -1 до 1. При порівнянні $c(w, w^*)$ із заданим порогом T, якщо $c(w, w^*) > T$, то виявляється існування водяного знаку.

Отже в даному підрозділі було описано досліджуваний алгоритм, що буде вдосконалений у наступному підрозділі. Даний алгоритм має недолік у вигляді першого етапу алгоритму, що виконує функцію пошуку підходящих вершин для вбудовування інформації. В наступному підрозділі буде виконано вдосконалення алгоритму таким чином, щоб краще протистояти геометричним атакам.

2.2 Вдосконалення алгоритму

Алгоритм, що був описаний у попередньому підрозділі, використовує алгоритм пошуку опорних точок для вбудовування водяного знаку, що засновується на знаходженні оцінки різкої зміни площі. Даний алгоритм має ряд недоліків, а саме опорні точки, що вираховуються для заданої сітчастої моделі не завжди є надійними для вбудовування інформації, а також їх пошук та впорядкування може бути ускладнене геометричними атаками на модель.

Тому пропонується використати для пошуку опорних точок алгоритм 3D-розпізнавання реберних вершин [52]. Дані вершини є більш стабільними для вбудовування водяного знаку, а їх пошук та визначення є надійнішими, ніж представлено у описаному методі.

Розглянемо даний алгоритм, він складається з декількох етапів.

Нехай сіткова модель буде позначена M є сукупністю ребер. Ребра представлені у вигляді $e_i \in M(i = 1,2,\dots,R)$, де R – множина всіх ребер моделі, а $\overrightarrow{e_i}$ – вектор, що представляє ребро. Кожне ребро складається з двох вершин, вершини позначені як в основному методі. Будується окіл навколо двох вершин ребра, з радіусом α.

Далі визначається площина параметрів, що перетинає середину вектору $\overrightarrow{e_i}$ та є перпендикулярною до нього, що зображено на рисунку 2.6(а). Унікальна площина визначається вимогою, щоб середина точки лежала на цій площині.

Точки обчислюються з перетину площини з набором сусідніх ребер, як показано на рисунку 2.6(б).

Надалі підбирається два поліноми: один для вершин, що лежать по один бік області параметрів, а інший для вершин, що лежать по інший бік, як показано на рисунку 2.6 (в).

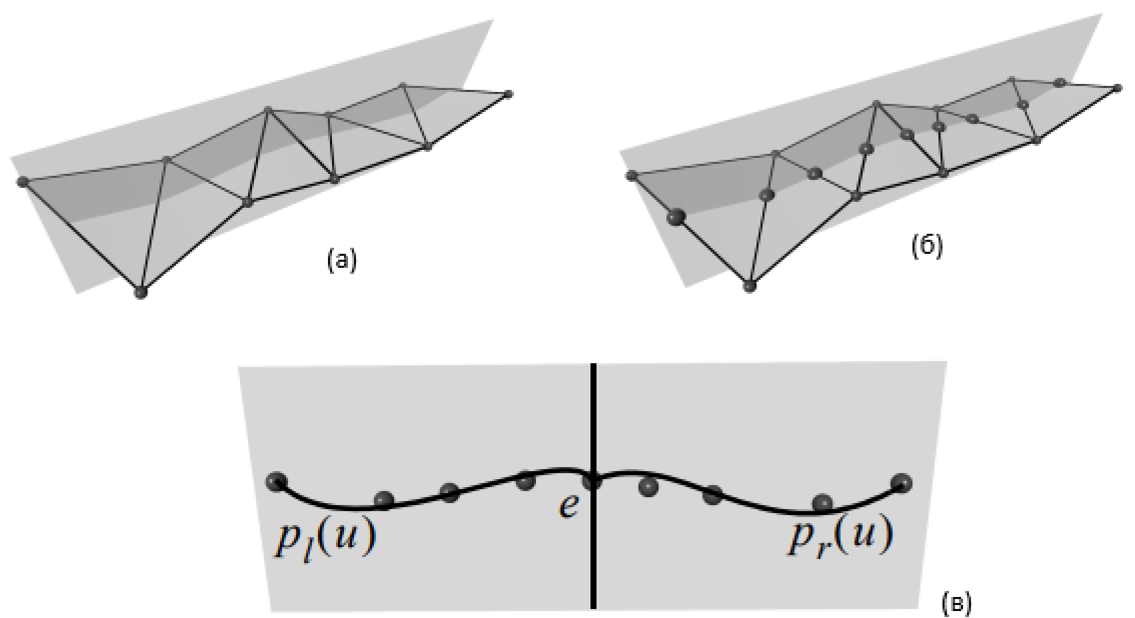


Рисунок 2.6 – Методи визначення реберної вершини: (а) – площина параметрів, (б) – перетин між площиною параметрів і сіткою, (в) –два многочлени і кут між їхніми дотичними в точці e [52]

Вага, що присвоюється поліномам, вибирається пропорційною куту між дотичними до двох кривих, оціненими в позиції параметра. Вага обчислюється рівнянням (2.11):

$$w(e_i) = \cos\left(\frac{(1, p_l'(e_i))}{\|(1, p_l'(e_i))\|} \cdot \frac{(1, p_r'(e_i))}{\|(1, p_r'(e_i))\|}\right)^{-1} \qquad (2.11)$$

де $p_l'(e_i)$ – похідна лівого поліному від ребра e_i,

$p_r'(e_i)$ – похідна правого поліному від ребра e_i,

$w(e_i)$ – вага ребра.

Додатковий ступінь свободи надається степенем підгонки полінома, який може бути адаптований до розміру підтримки оператора. Перевага такого підходу полягає в тому, що він є дуже гнучким, оскільки підтримка може бути адаптована як глобально, так і локально. Таким чином, на нього менше впливає шум, який відфільтровується під час процесу найкращого припасування. Крім того, вона може бути використана для будь-якого типу сітки, за умови вибору відповідного набору параметрів.

Отримані кутові ребра і містять кутові вершини моделі. Далі в ці вершини будуть використовуватися для вбудовування водяного знаку.

Вдосконалений алгоритм є більш стійким до геометричних атак завдяки використанню алгоритму 3D-розпізнавання реберних вершин, що є більш надійнішим ніж оригінальний алгоритм пошуку опорних точок для вбудовування водяного знаку.

2.3 Розробка алгоритму роботи програмного додатку

Алгоритм роботи програмного додатку буде складатися з блоків, це зроблено для кращого розуміння роботи програмного додатку та для спрощення реалізації в наступному розділі. Кожен блок буде відповідати за певний етап алгоритму.

Перший блок відповідає за обробку прийнятої сітчастої моделі та її підготовку для виконання вбудовування водяного знаку (рис. 2.7).

Рисунок 2.7 – Блок схема підготовки даних

Розглянемо кожен крок цієї блок схеми:

Крок 1. Отримання моделі.

Крок 2. Генерація водяного знаку з випадкових чисел, що вибираються з гаусівського розподілу з нульовим середнім значенням і дисперсією 1.

Крок 3. Розділення моделі на масив ребер.

Крок 4. Визначення середніх нормалей для всіх вершин моделі.

Таким чином відбувається підготування даних до подальшої обробки. Наступна блок схемі відображає визначення кутових ребер (рис. 2.8).

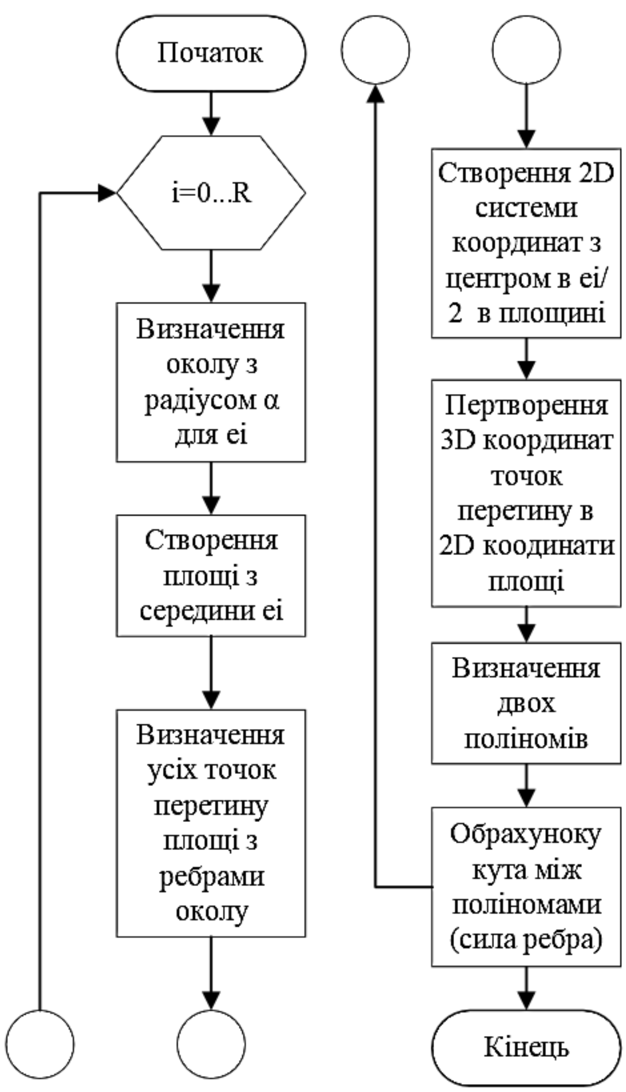


Рисунок 2.8 – Визначення сили ребер

Алгоритм проходить по всім ребрам моделі. Для кожного ребра відбувається визначення сили ребра. Розглянемо кожен крок для визначення сили для одного ребра:

Крок 1. Визначення околу з певним радіусом для обраного ребра.

Крок 2. Створення площі з середини обраного ребра.

Крок 3. Визначення усіх точок перетину площі з ребрами в околі.

Крок 4. Створення двовимірної системи координат з центром у вибраній точці в середині побудованої площини.

Крок 5. Перетворення визначених тривимірних координат точок перетину з площиною у двовимірні координати даної площі.

Крок 5. Визначення лівого та правого поліномів.

Крок 6. Обрахунок кута між двома поліномами.

Таким чином визначається сила кожного ребра.

Далі відбувається процес відсіювання, де обираються ребра з найбільшою силою а в них обираються опорні вершини (рис. 2.9).

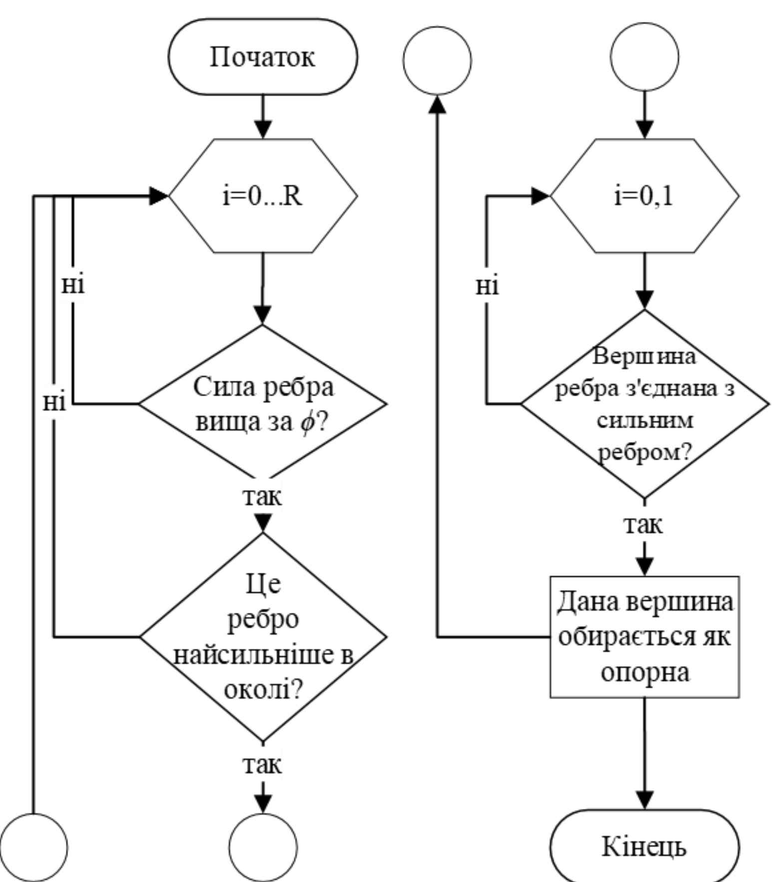


Рисунок 2.9 – Вибір опорних вершин

Усі ребра відсіюються за силою ребра ϕ. Дана сила може мати значення від 0 до 180 градусів. Після цього перевіряється чи в околі ребра є сильніше за нього

ребро. Таким чином вдається оминути можливість накладання даних. Далі з даного ребра обирається вершина, що з'єднана з сильнішим ребром. Таким чином обирається вершина, що позначається як опорна, і яка буде використовуватися для вбудовування в неї водяного знаку.

Далі відбувається процес підготовки ділянки Вороного (рис. 2.10).

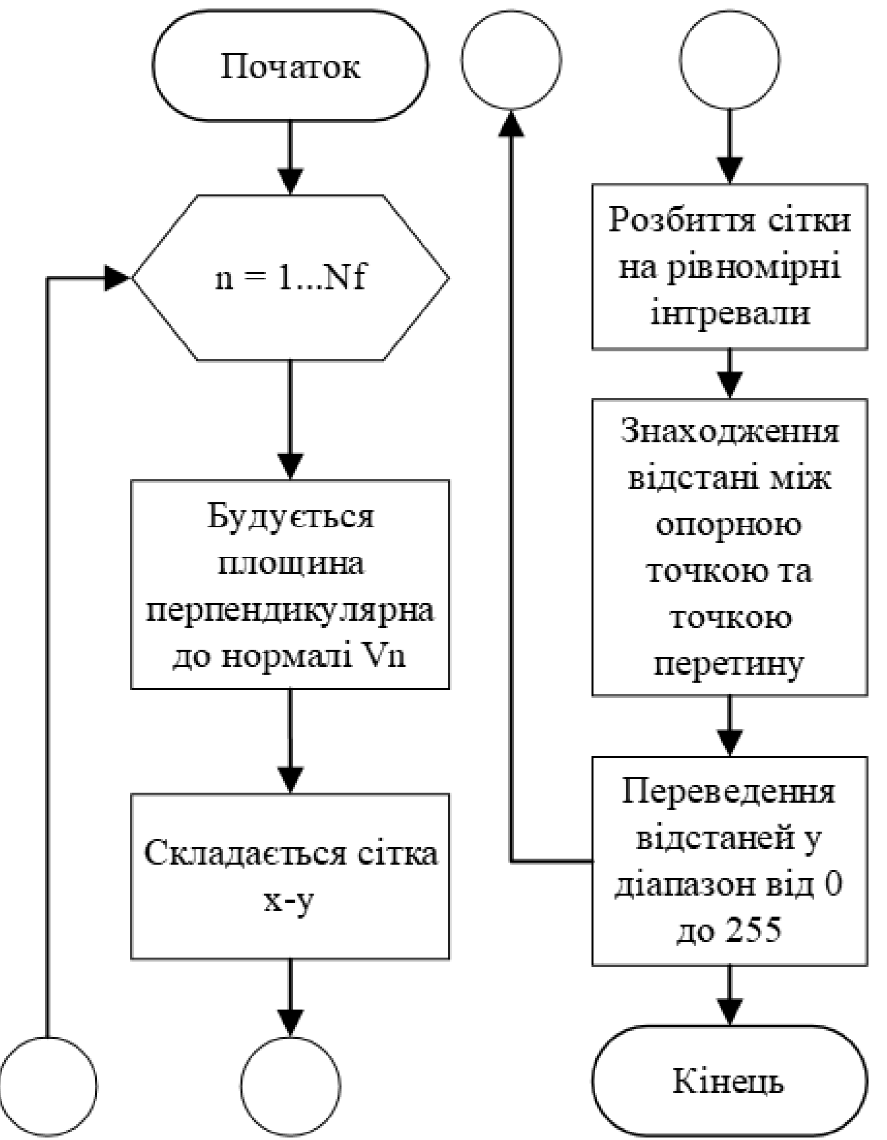


Рисунок 2.10 – Блок схема визначення зображення діапазону

Дана блок схема бере кожну опорну вершину та створює зображення діапазону з її околу. Надалі в дане зображення буде вбудовуватися водяний знак.

Розглянемо кожен крок блок схеми для окремої ділянки Вороного:

Крок 1. Будуємо площину перпендикулярну до нормалі Vn.

Крок 2. Складаємо сітку двовимірних координат на побудованій площині.

Крок 3. Розбиття отриманої двовимірної сітки координат на рівномірні проміжки, для утворення квадратних ділянок.

Крок 4. Визначення відстані між опоною точкою на моделі та точкою перетину в двовимірній сітці.

Крок 5. Перетворення відстаней у діапазон від 0 до 255.

Таким чином будується зображення діапазону.

Наступна блок схема відповідає за вбудовування водяного знаку (рис. 2.11).

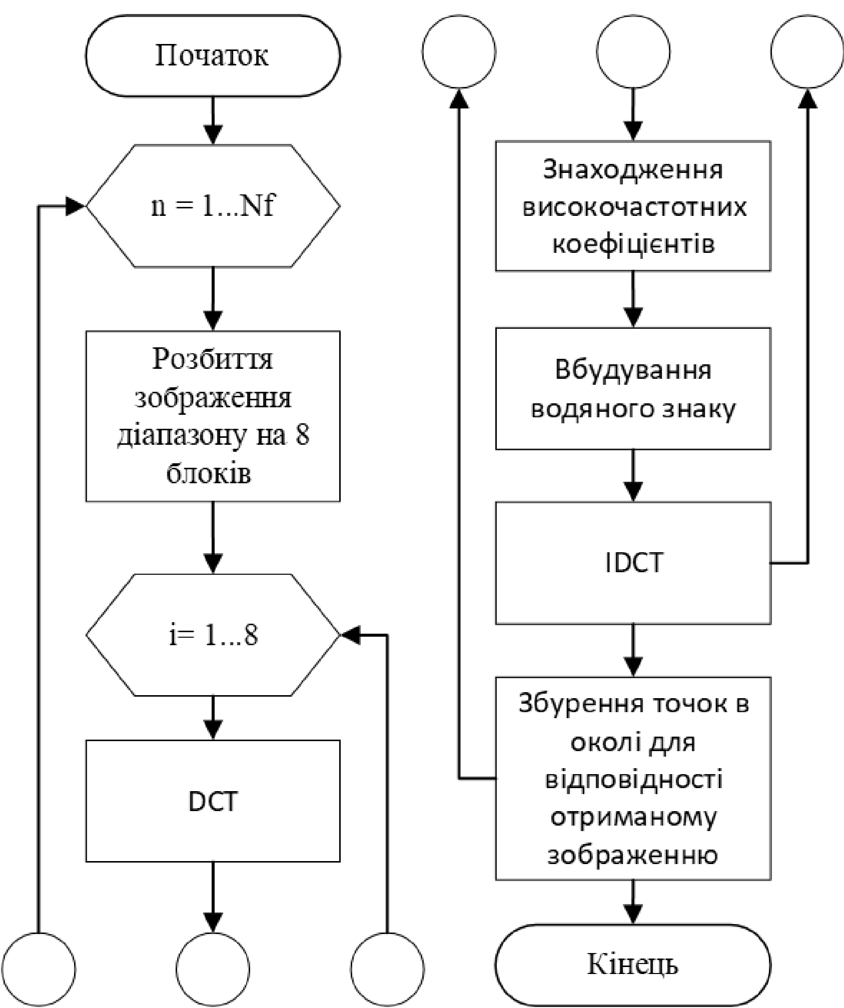


Рисунок 2.11 – Блок схема вбудовування водяного знаку

В даній блок схемі відбувається вбудовування водяного знаку в сітчасту модель. Розглянемо кожен крок блок схеми для окремої ділянки Вороного:

Крок 1. Розділення зображення діапазону на 8 блоків.

Крок 2. Застосування дискретного косинусного перетворення до кожного блоку.

Крок 3. В кожному блоці знаходяться високочастотні коефіцієнти, що будуть використовуватися для вбудовування інформації про водяний знак.

Крок 4. Використання оберненого дискретного косинусного перетворення (IDCT).

Крок 5. Збурення точок в околі, для внесення водяного знаку з отриманого зображення.

Остання блок схема відповідає за витягування та перевірку водяного знаку з моделі (рис. 2.12).

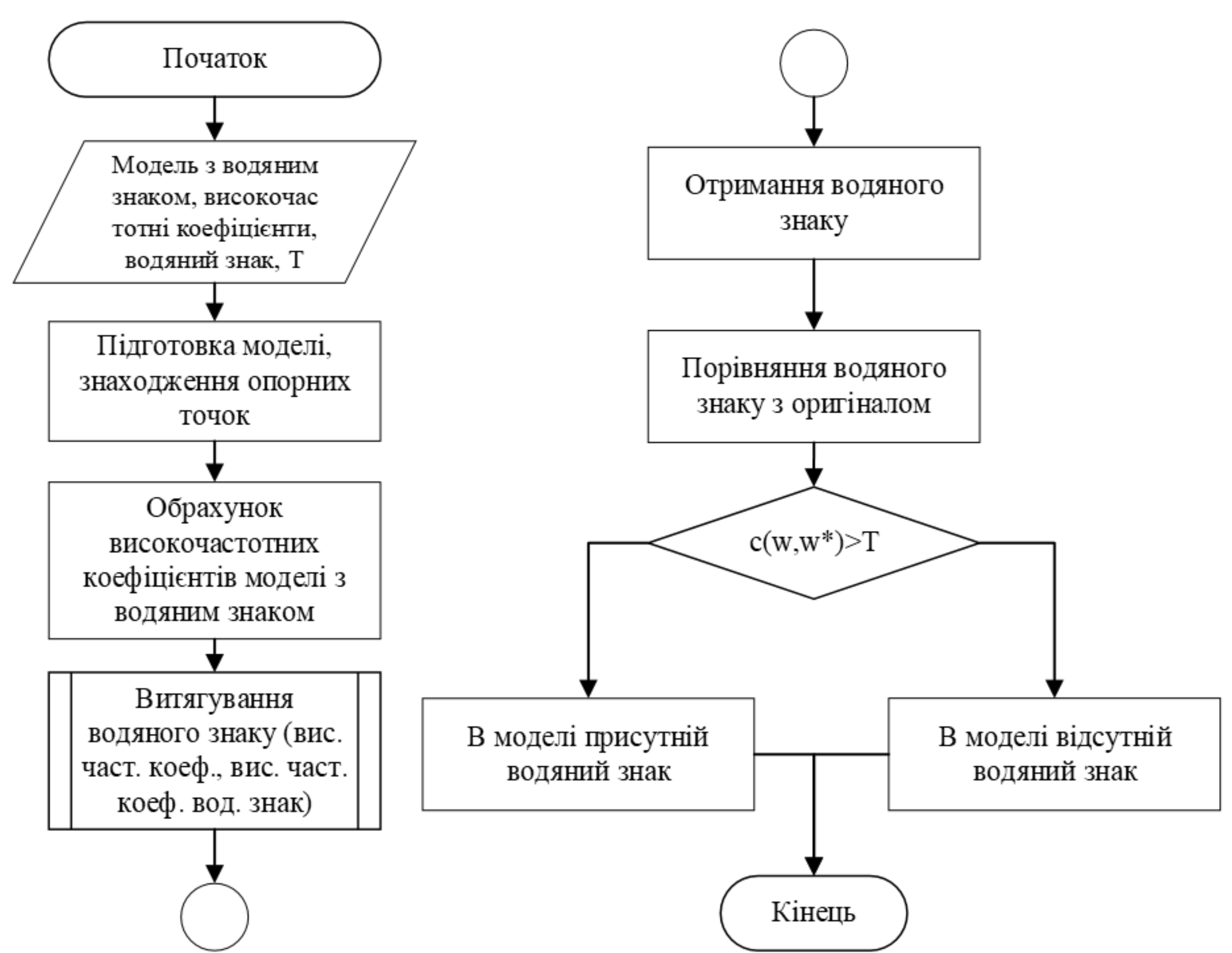


Рисунок 2.12 – Вилучення та виявлення водяного знаку з моделі

Розглянемо кожен крок блок схеми вилучення та виявлення водяного знаку з моделі:

Крок 1. Отримання наступних параметрів: високочастотні коефіцієнти, що використовувалися для вбудовування водяного знаку, модель з водяним знаком, водяний знак та значення *T*, що використовується для порівняння водяних знаків.

Крок 2. Обробка та знаходження відповідних опорних точок.

Крок 3. Обрахунок зображення діапазону з якого витягуються високочастотні коефіцієнти.

Крок 4. Витягування водяного знаку за допомогою формул 2.8.

Крок 5. Отримування остаточного водяного знаку за формулою 2.9.

Крок 6. Отриманий водяний знак порівнюється з оригіналом за допомогою формули 2.10.

Крок 6.1. Якщо отримане значення більше за поріг *T*, то це означає, що водяний знак вбудований в модель.

Крок 6.2. Якщо отримане значення менше за поріг *T*, то це означає, що водяний знак відсутній в моделі.

Таким чином було розроблено алгоритм роботи програмного забезпечення, що буде використовуватися під час його написання. Даний алгоритм було зображено у блок схемах для простішого розуміння його структури та роботи.

2.4 Висновок до розділу 2

В даному розділі проведено розбір та дослідження алгоритму вбудовування водяного знаку. В оригінальному алгоритмі було виявлено недолік у першому етапі, що відповідає за пошук опорних точок для вбудовування водяного знаку. Даний алгоритм виявився вразливим до геометричних операцій перетворення.

Тому здійснено вдосконалення даного методу за допомогою використання алгоритму пошуку реберних вершин для тривимірних моделей, що дозволяє підвищити стійкість даного алгоритму проти геометричних атак. Дане вдосконалення замінює перший етап оригінального алгоритму вбудовування водяних знаків в тривимірні моделі, що є більш надійним методом для пошуку опорних вершин в моделях, що зазнали геометричних атак.

Виходячи з запропонованого вдосконалення в даному розділі здійснено розробку алгоритмів програмної реалізації удосконаленого методу вбудовування, що підвищує стійкість водяних знаків до геометричних атак. Розроблені алгоритми будуть використані під час програмної реалізації вдосконаленого методу.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВДОСКОНАЛЕНОГО АЛГОРИТМУ ВБУДОВУВАННЯ ТА РОЗПІЗНАВАННЯ ВОДЯНИХ ЗНАКІВ

В даному розділі проведено практичну реалізацію вдосконаленого алгоритму вбудовування та розпізнавання водяних знаків для тривимірних моделей. Проведено опис класів та функцій, що використовуються для вбудовування, перевірки та отримання водяного знаку. Також наведено приклад реалізації вдосконаленого алгоритму та проведено аналіз стійкості вдосконаленого алгоритму вбудовування та розпізнавання водяних знаків для тривимірних моделей.

3.1 Практична реалізація програми вдосконаленого алгоритму вбудовування та розпізнавання водяних знаків

Через великий обсяг коду програми, в даному розділі буде описано частина класів та їх функцій, що відповідають за вбудовування та отримування водяного знаку з тривимірної моделі. Увесь програмний код міститься в Додатку Б. Для написання програми було обрано мову програмування C++ та бібліотеку IGL.

Перший клас, відповідає за сам процес вбудовування та отримування водяного знаку. Він має наступні функції та параметри:

```
class Watermark{
public:
        Watermark();
        void SetWatermark(EdgeVertexDetection, Eigen::MatrixXd&, Eigen::MatrixXd&, const vector<char>&, const int);
        void GetWatermark(EdgeVertexDetection, Eigen::MatrixXd&, Eigen::MatrixXd&, const vector<char>&, const int);
        bool DetectionWatermark(const vector<char>&, const vector<char>&, int);
private:
        Eigen::MatrixXd& V;
        Eigen::MatrixXi& F;
        EdgeVertexDetection edgeDetection;
        vector<char>& wat;
        const int R;
        void GetVoronoiPatch();
        void DCT();
        void IDCT();
        void SetBitWatermark();
        void GetBitWatermark();};
```

Клас Watermark відповідає за вбудовування та отримання водяного знаку з тривимірної моделі.

Даний клас має публічні методи, що відповідають за отримання, встановлення та виявлення водяного знаку.

Метод SetWatermark() встановлює водяний знак. Приймає в параметри клас EdgeVertexDetection, що відповідає за виявлення краї в тривимірній моделі, два параметри Eigen::MatrixXd&, що відповідають за сітчасту модель, водяний знак, що представлений як vector<char>&, та ширину округу, яка визначена const int.

Метод GetWatermark() отримує водяний знак. Приймає об'єкт EdgeVertexDetection, матриці MatrixXd та vector<char>. Отримує водяний знак з відповідних даних.

Метод DetectionWatermark() виконує виявлення водяного знаку. Приймає два вектори vector<char> та ціле число. Перевіряє наявність водяного знаку. В моделі, даний метод заснований на формулах виявлення водяного знаку представлених в попередньому розділі.

Також клас має ряд приватних методів, що використовується для вбудовування або отримування водяного знаку з моделі. Метод GetVoronoiPatch() отримує ділянку Вороного навколо обраної точки.

Метод DCT() виконує дискретне косинусне перетворення.

Метод IDCT() виконує зворотне дискретне косинусне перетворення.

Метод SetBitWatermark() встановлює біт водяного знаку за допомогою отриманого зображення після дискретного косинусного перетворення.

Метод GetBitWatermark() отримує біт водяного знаку з отриманого зображення дискретного косинусного перетворення.

Клас має відповідні поля для роботи з тривимірною сітчастою моделлю.

Поле V є посиланням на Eigen::MatrixXd і представляє собою вершини моделі.

Поле F є посиланням на Eigen::MatrixXi і представляє собою трикутники моделі.

Поле edgeDetection зберігає в собі об'єкт EdgeVertexDetection.

Поле wat є посиланням на vector<char> та зберігає в собі водяний знак.

Поле R є константою типу int та зберігає в собі розмір околу навколо точок.

Цей клас призначений для вбудовування, отримання та виявлення водяного знаку на основі операцій з матрицями та векторами.

Розглянемо детально клас EdgeVertexDetection. Даний клас має наступні поля та методи:

```
class EdgeVertexDetection{
public:
        Eigen::MatrixXd Nv;
        Eigen::MatrixXi Nf;
        std::vector<Eigen::Vector2d> TestVector2d;
        std::vector<Eigen::Vector3d> TestVector3d;
        double EdgeStrength(int i);
        double EdgeStrength(Edge* edge);
        EdgeVertexDetection(const Eigen::MatrixXd originV, const Eigen::MatrixXi originF, const int originR);
        ~EdgeVertexDetection();
        Edge** Edges;
private:
        Eigen::MatrixXd V;
        Eigen::MatrixXi F;
        int R;
        std::vector<Edge> DetermineAroundN(Edge* edge);
        void MoveToZeroCoordinates(Eigen::MatrixXd v, const Eigen::Vector3d vi);
        void RemoveCopyItem(std::vector<int>& v);
        void RemoveCopyItem(std::vector<Eigen::Vector3d>& v);
        void RemoveCopyItem(std::vector<Eigen::Vector2d>& v);
        void RemoveCopyItem(std::vector<Edge>& v);
        std::vector<int> BringingFaceArrayToVertexArray(const std::vector<int>& v, const std::vector<int>& vertexList);
        std::vector<Eigen::Vector2d> ConversionToPlaneCoordinates(const std::vector<Eigen::Vector3d> cooedinate, const Eigen::Vector3d center);
        double DotProduct(const Eigen::Vector3d& v1, const Eigen::Vector3d& v2);
        double DotProduct(const Eigen::Vector2d& v1, const Eigen::Vector2d& v2);
        double VectorLength(const Eigen::Vector3d& v);
        double VectorLength(const Eigen::Vector2d& v);
        double AngleBetweenVectors(const Eigen::Vector3d& v1, const Eigen::Vector3d& v2);
        double AngleBetweenVectors(const Eigen::Vector2d& v1, const Eigen::Vector2d& v2);
        double LagrangeBasis(double x, const std::vector<Eigen::Vector2d>& v, int i);
        double LagrangePolynomialDerivative(double x, const std::vector<Eigen::Vector2d>& v);
        Eigen::Vector2d NormalizeVector(Eigen::Vector2d vector);};
```

Клас EdgeVertexDetection виявляє реберні вершини у тривимірних об'єктах. Він має різні методи для обчислення сили ребра, перетину, видалення

повторюваних елементів і нормалізації векторів, що допомагає у виявленні структур у моделях.

Розглянемо кожне поле даного класу:

– Eigen::MatrixXd V – матриця вершин для початкової моделі;

– Eigen::MatrixXi F – матриця граней для початкової моделі;

– Eigen::MatrixXd Nv – матриця вершин (видозмінена) для об'єкта EdgeVertexDetection;

– Eigen::MatrixXi Nf – матриця граней (видозмінена) для об'єкта EdgeVertexDetection;

– std::vector<Eigen::Vector2d> TestVector2d – вектор з об'єктами типу Eigen::Vector2d для тестування;

– std::vector<Eigen::Vector3d> TestVector3d - вектор з об'єктами типу Eigen::Vector3d для тестування;

– double R - радіус околу;

– Edge** Edges – покажчик на масив покажчиків на об'єкти типу Edge.

Також даний клас має багато методі, розглянемо основний метод, конструктор та деякі допоміжні методи для пошуку реберних вершин.

Конструктор даного класу, також виконує функцію обчислення сил усіх ребер моделі. Він містить наступний код:

```
EdgeVertexDetection::EdgeVertexDetection(const Eigen::MatrixXd originV, const Eigen::MatrixXi originF, const int originR = 3) :R(originR), V(originV), F(originF){
		Edges = new Edge * [F.rows() * 3];{
			int b = 0, e = 1;
			for (size_t i = 0, j = 0; i < F.rows(); i++){
				for (size_t t = 0; t < 3; t++){
					Edges[j] = new Edge(F(i, b),	F(i, e),
						Eigen::Vector3d(V(F(i, b), 0), V(F(i, b), 1), V(F(i, b), 2)),
						Eigen::Vector3d(V(F(i, e), 0), V(F(i, e), 1), V(F(i, e), 2)));
					b == 2 ? b = 0 : b++;
					e == 2 ? e = 0 : e++;
					j++;}}}
		for (size_t i = 0; i < F.rows() * 3; i++) {EdgeStrength(i);}}
```

Конструктор класу EdgeVertexDetection приймає три параметри: originV (матриця початкових вершин), originF (матриця початкових граней) і originR (радіус округу, який за замовчуванням дорівнює 3).

Стрічка R(originR), V(originV), F(originF) – це ініціалізація константних полів класу R, V і F за допомогою переданих в конструктор значень.

Далі в змінну Edges виділяється пам'ять під масив покажчиків на об'єкти Edge. Кількість рядків у матриці F множиться на 3 (оскільки кожна грань має 3 ребра), і це число стає розміром для масиву Edges.

Цикл for ініціалізує об'єкти Edge для кожної грані у вхідних даних. Внутрішній цикл виконується тричі для кожної грані, оскільки кожна грань має три ребра. Для кожного ребра викликається конструктор Edge, ініціалізуючи об'єкти Edge з відповідними даними: номери вершин та їх координати.

Наступний цикл for обчислює силу кожного ребра в об'єктах Edge, викликаючи метод EdgeStrength для кожного ребра за його індексом.

Отже, цей конструктор виконує ініціалізацію об'єкту EdgeVertexDetection, створює ребра для кожної грані та обчислює їх силу.

Розглянемо метод EdgeStrength, що обчислює силу кожного ребра. Частина методу представлена далі:

```
double EdgeVertexDetection::EdgeStrength(Edge* edge){
    std::vector<Edge> numVertexList = DetermineAroundN(edge);
    Plane plane(edge->GetVectorAB(), edge->GetMiddleVertex());
    std::vector<Eigen::Vector3d> list(0);
    list.push_back(edge->GetMiddleVertex());
    for (size_t i = 0; i < numVertexList.size(); i++){
        Eigen::Vector3d temp{0, 0, 0};
        if (plane.DoesIntersectSegment(
            numVertexList[i].GetVertexA(),
            numVertexList[i].GetVertexB(),
            temp)) {list.push_back(temp);}}
    TestVector3d = list;
    std::vector<Eigen::Vector2d> coordinatesOnPlane = ConversionToPlaneCoordinates(list,
edge->GetMiddleVertex());
    TestVector2d = coordinatesOnPlane;
```

Розберемо даний код. Спочатку визивається метод DetermineAroundN, щоб отримати список numVertexList ребер, які оточують обране ребро. Далі створюється

площина за допомогою вектора GetVectorAB ребра та його середньої вершини GetMiddleVertex.

Заповнюється список координат точок list, який містить координати середньої вершини та будь-яких точок перетину ребра з площиною, знайдених за допомогою методу DoesIntersectSegment.

Отримані координати точок на площині конвертуються у площинні координати за допомогою методу ConversionToPlaneCoordinates.

Розглянемо наступний фрагмент коду методу EdgeStrength:

```
std::vector<Eigen::Vector2d> positiveX (0);
std::vector<Eigen::Vector2d> negativeX (0);
for (size_t i = 0; i < coordinatesOnPlane.size(); i++)  {
        coordinatesOnPlane[i].y() *= -1;
        if (coordinatesOnPlane[i].x() > 0) {
positiveX.push_back(coordinatesOnPlane[i]);}
        if (coordinatesOnPlane[i].x() < 0) {
                negativeX.push_back(coordinatesOnPlane[i]);}}
RemoveCopyItem(positiveX);
RemoveCopyItem(negativeX);
```

Цей фрагмент коду маніпулює координатами точок на площині. Створюються два вектори точок positiveX та negativeX, які відповідають точкам з позитивними і від'ємними значеннями x відповідно. Далі код проходиться по кожній точці у списку coordinatesOnPlane, і її координата *y* множиться на -1 (це операція дзеркального відображення відносно вісі).

Точки, які мають позитивне значення *x*, додаються до списку positiveX, а з від'ємним значенням до списку negativeX. Викликається метод RemoveCopyItem, щоб видалити дубльовані точки з обох списків positiveX та negativeX.

Розглянемо наступний фрагмент коду методу EdgeStrength:

```
double minCoordPositiveX = coordinatesOnPlane[coordinatesOnPlane.size() - 1].x();
for each (auto var in coordinatesOnPlane) {
        if (var.x() == 0) continue;
        double temp = var.x();
        if (var.x() < 0)   temp = var.x() * (-1.0);
        minCoordPositiveX >= temp ? minCoordPositiveX = temp : false;}
double scale = 1;
do{     scale *= 10;
        minCoordPositiveX = minCoordPositiveX * scale;
```

```
} while (minCoordPositiveX < 100);
for (size_t i = 0; i < negativeX.size(); i++){
        negativeX[i].x() *= scale;
        negativeX[i].y() *= scale;}
for (size_t i = 0; i < positiveX.size(); i++){
        positiveX[i].x() *= scale;
        positiveX[i].y() *= scale;}
```

Цей фрагмент коду масштабує координати точок на площині:

З початку відбувається пошук мінімальної позитивної x-координати: Обчислюється найменша позитивна x-координата серед усіх точок у coordinatesOnPlane.

Далі відбувається масштабування. Всі точки з від'ємними значеннями x та позитивними значеннями x масштабуються, множачи їх *x* та *y* координати на змінну scale. Це призначено для збільшення розміру цих точок на площині. В даному випадку відбувається масштабування координат у діапазоні значень, щоб розташувати їх у більш зручному масштабі.

Розглянемо наступний фрагмент коду методу EdgeStrength:

```
const double y = 1;
double pr_e_derivative = LagrangePolynomialDerivative(0, positiveX);
double pl_e_derivative = LagrangePolynomialDerivative(0, negativeX);
Eigen::Vector2d lOneVector = Eigen::Vector2d(y / VectorLength(Eigen::Vector2d(y, pl_e_derivative)), pl_e_derivative / VectorLength(Eigen::Vector2d(y, pl_e_derivative)));
Eigen::Vector2d rOneVector = Eigen::Vector2d(y / VectorLength(Eigen::Vector2d(y, pr_e_derivative)), pr_e_derivative / VectorLength(Eigen::Vector2d(y, pr_e_derivative)));
```

Константа *y* ініціалізується значенням 1. Далі обчислюються похідні функцій Лагранжа pr'(e) та pl'(e) для вхідних даних positiveX та negativeX відповідно.

Створюються вектори lOneVector та rOneVector з використанням обчислених значень похідних. Це вектори з двома значеннями (відношенням y до довжини вектора, утвореного (y, pl_e_derivative) та (y, pr_e_derivative)). Це вектори одиничної довжини, які вказують у напрямку цих похідних.

Розглянемо останній фрагмент коду методу EdgeStrength:

```
int nMinus = negativeX.size();
int nPlus = positiveX.size();
Eigen::MatrixXd A_minus_x(nMinus, 2);
Eigen::VectorXd b_minus_x(nMinus);
```

```
Eigen::MatrixXd A_plus_x(nPlus, 2);
Eigen::VectorXd b_plus_x(nPlus);
for (int i = 0; i < A_minus_x.rows(); i++) {
    A_minus_x(i, 0) = negativeX[i].x();
    A_minus_x(i, 1) = 1.0;
    b_minus_x(i) = negativeX[i].y();}
for (int i = 0; i < A_plus_x.rows(); i++) {
    A_plus_x(i, 0) = positiveX[i].x();
    A_plus_x(i, 1) = 1.0;
    b_plus_x(i) = positiveX[i].y();}
Eigen::Vector2d coefficients_minus_x = A_minus_x.colPivHouseholderQr().solve(b_minus_x);
Eigen::Vector2d coefficients_plus_x = A_plus_x.colPivHouseholderQr().solve(b_plus_x);
Eigen::Vector2d gradient_minus_x(coefficients_minus_x(0), 1.0);
Eigen::Vector2d gradient_plus_x(coefficients_plus_x(0), 1.0);
double angle = AngleBetweenVectors(gradient_minus_x, gradient_plus_x) * (180.0 / 3.14159265359);
edge->SetWeightEdge(angle);
return edge->GetWeightEdge();
```

З початку метод визначає кількість точок даних для негативних (nMinus) та позитивних (nPlus) значень. Далі метод створює матриці A_minus_x та A_plus_x та вектори b_minus_x та b_plus_x, які будуть використовуватися для методу найменших квадратів (МНК). A_minus_x та A_plus_x це матриці, де кожен рядок представляє точку даних з відповідними значеннями x та константою 1. b_minus_x та b_plus_x містять відповідні значення y для негативних та позитивних значень.

Наступним етапом вирішується система лінійних рівнянь методом найменших квадратів, щоб знайти коефіцієнти поліномів, які найкраще підходять до наборів точок даних. Розв'язки - це coefficients_minus_x та coefficients_plus_x.

З coefficients_minus_x та coefficients_plus_x витягуються значення коефіцієнтів, які представляють градієнти відповідних поліномів, тобто вектори (a, 1) для кожного.

Обчислюється кут між двома градієнтами. Цей кут використовується для встановлення ваги (weight) у ребрі графа (змінна edge). Потім це значення ваги повертається як результат методу.

Цей фрагмент виконує розрахунок кута між градієнтами після застосування МНК до наборів точок даних, які відображаються у просторі. Даний кут і є силою ребра.

Розглянемо деякі допоміжні методи, що використовуються для визначення сили ребра.

Перший метод має назву ConversionToPlaneCoordinates та має наступний код:

```
std::vector<Eigen::Vector2d>        EdgeVertexDetection::ConversionToPlaneCoordinates(const
std::vector<Eigen::Vector3d> cooedinate,const Eigen::Vector3d center){
        Edge xVector(0, 0, center, cooedinate[cooedinate.size() - 1]);
        vector<Eigen::Vector2d> result(0);
        for (size_t i = 0; i < cooedinate.size(); i++){
                Edge v(0, 0, center, cooedinate[i]);
                double angle = AngleBetweenVectors(xVector.GetVectorAB(), v.GetVectorAB());
                if (!angle || std::isnan(angle)) {
                        result.push_back(Eigen::Vector2d(v.GetDistance(), 0));
                        continue;}
                if(angle >= 90)continue;
                double x = v.GetDistance() * std::cos(angle);
                double y = v.GetDistance() * std::sin(angle);
                result.push_back(Eigen::Vector2d(x, y));}
        return result;}
```

Метод ConversionToPlaneCoordinates виконує перетворення тривимірних координат у площинні координати відносно заданої центральної точки (center).

Спочатку визначається вектор xVector, який представляє вектор між центральною точкою і останньою координатою у списку cooedinate.

В циклі для кожної координати зі списку cooedinate визначається вектор v, що представляє відстань та напрямок від центральної точки до кожної з цих координат.

Обчислюється кут між вектором xVector та кожним з векторів v. Якщо цей кут дорівнює нулю або є NaN (не числом), це означає, що вектор лежить на осі x, і тому координата y дорівнює нулю, а координата x є відстань від центру до цієї точки.

Якщо кут не дорівнює нулю або NaN, обчислюються нові значення x та y за допомогою косинусу та синусу кута між векторами. Ці значення x та y створюють новий вектор, який додається до результату у формі Eigen::Vector2d(x, y).

Отриманий результат – це вектор з площинними координатами (x, y) для кожної тривимірної координати у вихідному списку.

Наступний метод має назву AngleBetweenVectors та містить код що зображено нижче:

```
double EdgeVertexDetection::AngleBetweenVectors(const Eigen::Vector2d& v1, const Eigen::Vector2d& v2){
        double dotProduct = v1.dot(v2);
        double length1 = v1.norm();
        double length2 = v2.norm();
        double cosineTheta = dotProduct / (length1 * length2);
        double angleInRadians = std::acos(cosineTheta);
        return angleInRadians;}
```

Метод AngleBetweenVectors обчислює кут між двома векторами у двовимірному просторі, вектори подані як Eigen::Vector2d.

Спочатку використовується скалярний добуток (dot product) для обчислення добутку векторів v1 та v2. Далі обчислюються довжини кожного вектора за допомогою методу norm(). Потім визначається косинус кута між векторами, використовуючи властивості скалярного добутку та довжин векторів.

Значення косинуса перетворюється в радіани за допомогою оберненого косинусу (арккосинуса) для отримання кута між векторами. Кут повертається в радіанах.

Також є перевантажена версія даного методі, що використовується для визначення кутів між векторами у тривимірному просторі.

Наступний метод LagrangePolynomialDerivative, він містить код представлений нижче:

```
double EdgeVertexDetection::LagrangePolynomialDerivative(double x, const std::vector<Eigen::Vector2d>& v) {
        double result = 0.0;
        for (int i = 0; i < v.size(); i++) {
                double basis = LagrangeBasis(x, v, i);
                double sum = 0.0;
                for (int j = 0; j < v.size(); j++) {
                        if (j != i) {sum += 1.0 / (v[i].x() - v[j].x());}}
                result += basis * sum * v[i].y();}
        return result;}
```

Метод LagrangePolynomialDerivative обчислює похідну інтерполяційного полінома Лагранжа в точці *x*, використовуючи вхідний набір точок v (які представлені у вигляді Eigen::Vector2d).

Вона проходить по кожній точці у вхідному наборі v та обчислює базисний член Лагранжа для кожної точки за допомогою функції LagrangeBasis.

Далі розраховує суму, яка включає частку для кожної пари точок (крім поточної точки i). Підсумовує результати кожної ітерації обчислення базису, частки та y-координати кожної точки та повертає обчислене значення похідної інтерполяційного полінома Лагранжа в точці x.

Ця функція є частиною інтерполяційного підходу, який застосовується для обчислення похідних значень в заданій точці, і є важливим етапом у визначенні сили ребра.

Отже, було розглянуто два основних класи, що відповідають за вбудовування, отримування та перевірку водяного знаку в тривимірну модель. Програма містить набагато більше коду, але в більшості він відповідає за завантаження моделі, водяного знаку, збереження моделі з водяним знаком, показ моделі з водяним знаком та без нього. Дані функції не відносяться до основного завдання, а саме вбудовування, отримання та перевірка водяного знаку в тривимірну модель, тому не будуть описані в даному підрозділі.

3.2 Приклад реалізації вдосконаленого алгоритму

В підрозділі наведемо приклад використання розробленого додатку вдосконаленого алгоритму. Буде показано як вбудовувати водяний знак в тривимірну модель та отримувати його з тривимірної моделі з водяним знаком.

Інтерфейс користувача було обрано консольним, так як взаємодія з додатком обмежена лише вибором файлу з моделлю та файлом з водяним знаком.

Після вбудовування водяного знаку, буде показана модель з вбудованим водяним знаком.

Розглянемо процес запуску та налаштування програми.

Під час запуска програми, користувачеві потрібно буде ввести назву або шлях до файлу, ввести параметри околу за допомогою якого буду визначатися реберні вершини, вибрати режим вбудовування або екстракції водяного знаку, та обрати шлях до водяного знаку. Даний процес відображено на рисунку 3.1.

```
Введіть назву файлу моделі: Armadillo.ply
Введіть пареметр R: 3
Вбудовування чи екстрація водяного знаку?(1/2): 1
Введіть назву файлу водяного знаку: watermark.txt
igl::opengl::glfw::Viewer usage:
  [drag]  Rotate scene
  A,a     Toggle animation (tight draw loop)
  D,d     Toggle double sided lighting
  F,f     Toggle face based
  I,i     Toggle invert normals
  L,l     Toggle wireframe
  O,o     Toggle orthographic/perspective projection
  S,s     Toggle shadows
  T,t     Toggle filled faces
  Z       Snap to canonical view
  [,]     Toggle between rotation control types (trackball, two-axis
          valuator with fixed up, 2D mode with no rotation))
  <,>     Toggle between models
  ;       Toggle vertex labels
  :       Toggle face labels
```

Рисунок 3.1 – Запуск програми в режимі вбудовування водяного знаку

Після процесу вбудовування водяного знаку, що займе деякий час, з’явиться вікно з моделлю в яку було вбудовано водяний знак. Це показано на рисунку 3.2.

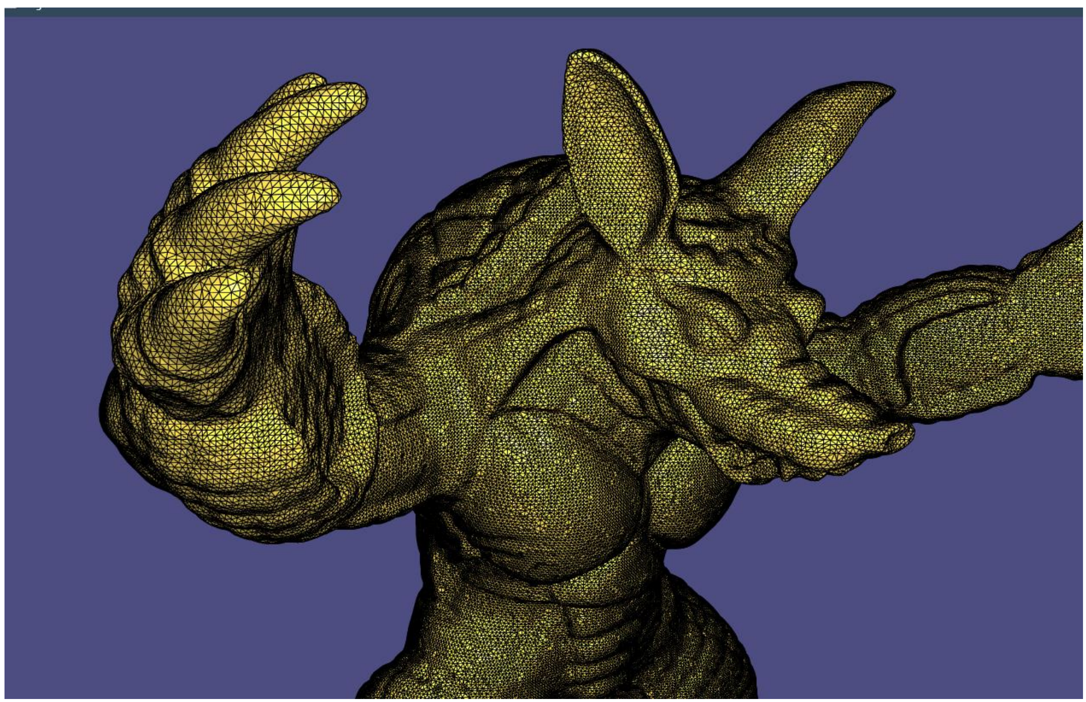


Рисунок 3.2 – Вікно з тривимірною моделлю в яку вбудовано водяний знак

Для того щоб зберегти дану модель, користувачеві потрібно натиснути клавішу «Ecs», після цього в консолі з’явиться повідомлення з пропозицією вказати куди зберегти файл та з яким ім’ям. Це показано на рисунку 3.3.

```
Введіть назву файлу моделі: Armadillo.ply
Введіть пареметр R: 3
Вбудовування чи екстрація водяного знаку?(1/2): 1
Введіть назву файлу водяного знаку: watermark.txt
igl::opengl::glfw::Viewer usage:
  [drag]  Rotate scene
  A,a     Toggle animation (tight draw loop)
  D,d     Toggle double sided lighting
  F,f     Toggle face based
  I,i     Toggle invert normals
  L,l     Toggle wireframe
  O,o     Toggle orthographic/perspective projection
  S,s     Toggle shadows
  T,t     Toggle filled faces
  Z       Snap to canonical view
  [,]     Toggle between rotation control types (trackball, two-axis
          valuator with fixed up, 2D mode with no rotation))
  <,>     Toggle between models
  ;       Toggle vertex labels
  :       Toggle face labels
Збереження моделі з водяним знаком
Введіть шлях збереження: C:\
Введіть назву моделі: ArmadilloWatermark.ply
```

Рисунок 3.3 – Процедура збереження тривимірної моделі в яку вбудовано водяний знак

Для екстракції водяного знаку потрібно вказати відповідний параметр. Після цього з’явиться подібне вікно з моделлю, з даної моделі буде проведена екстракція водяного знаку. Для того щоб отримати та визначити чи в моделі є водяний знак, котрий був вказаний на початку запуску програми потрібно також натиснути клавішу «Ecs». Даний процес показаний на рисунку 3.4

```
Введіть назву файлу моделі: ArmadilloWatermark.ply
Введіть пареметр R: 3
Вбудовування чи екстрація водяного знаку?(1/2): 2
Введіть назву файлу водяного знаку: watermark.txt
igl::opengl::glfw::Viewer usage:
  [drag]  Rotate scene
  A,a     Toggle animation (tight draw loop)
  D,d     Toggle double sided lighting
  F,f     Toggle face based
  I,i     Toggle invert normals
  L,l     Toggle wireframe
  O,o     Toggle orthographic/perspective projection
  S,s     Toggle shadows
  T,t     Toggle filled faces
  Z       Snap to canonical view
  [,]     Toggle between rotation control types (trackball, two-axis
          valuator with fixed up, 2D mode with no rotation))
  <,>     Toggle between models
  ;       Toggle vertex labels
  :       Toggle face labels
Модель містить даний водяний знак
Збереження зводяного знаку
Введіть шлях збереження: C:\
Введіть назву:  exWatermark.txt
```

Рисунок 3.4 – Процедура збереження та перевірки водяного знаку тривимірної моделі

Таким чином, після натискання клавіші «Ecs», програма виведе повідомлення про то, чи був вбудований обраний водяний знак в модель, та запропонує зберегти водяний знак, що було отримано після екстракція з тривимірної моделі.

Таким чином, було наведено приклад реалізації вдосконаленого алгоритму. Продемонстровано роботу з додатком для вбудовування та екстракції водяного знаку в тривимірну модель за допомого розробленого вдосконаленого алгоритму вбудовування та розпізнавання водяних знаків з використанням пошуку опорних точок. Було проведено процес вбудовування та збереження моделі з водяним знаком, а також процес екстракції, перевірки та збереження водяного знаку.

3.3 Аналіз стійкості вдосконаленого методу до геометричних атак

Аналіз стійкості удосконаленого алгоритму вбудовування водяних знаків у тривимірні моделі буде проводитися виконанням різних атак на обрані моделі. Результати проведеного аналізу будуть порівнюватися з показниками початкового методу вбудовування водяних знаків [51].

Для перевірки непомітності та стійкості буде використовуватися пікове відношення сигнал/шум (PSNR), воно буде визначатися між оригінальною моделлю та моделлю з водяним знаком. PSNR визначається за рівнянням 3.1:

$$PSNR = 10log_{10}\frac{N \cdot \|\vec{v}\|_{max}^2}{\sum_{i=1}^{N}\left\|\vec{v}_i^* - \vec{v}_i\right\|^2} \quad (3.1)$$

де $\vec{v}_i$ – це вершина вихідної моделі,

$\vec{v}_i^*$ – це відповідна вершина моделі з водяним знаком,

N – це загальна кількість вершин моделі,

$\|\vec{v}\|_{max}^2$ – це вершина вихідної моделі, яка є найвіддаленішою від центру.

Для надійності перевірки, було обрано моделі представлені на рисунку 3.5.

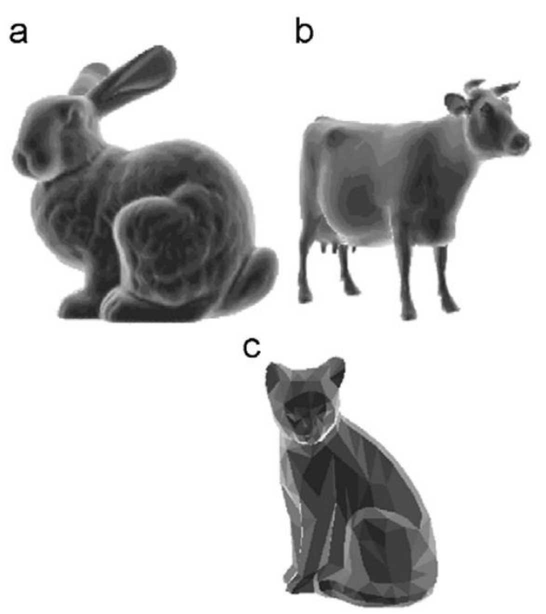


Рисунок 3.5 – Моделі, що використовується для аналізу вдосконаленого методу: (a) модель зайця, (b)модель корови та (c) модель кота [51]

Таблиця з вхідними параметрами моделей, значенням PSNR та параметрами для початкового і вдосконаленого алгоритму наведено в таблиці 3.1.

Таблиця 3.1 – Параметри моделей для аналізу непомітності

Модель	Вершини	Полігони	Ділянки	$N_x \times N_y$	L	α	β	PSNR (дБ) ориг. [51]	PSNR (дБ) вдоск.
Заєць	362272	725000	20	80×80	20	5	0,1	88.6837	88.3561
Корова	3406	6952	15	36×36	30	3	0,05	87.8950	88.1424
Кіт	418	805	8	20×20	15	2	0,01	86.5972	86.9217

Кількість ділянок поділу для моделей становить 20, 15 і 8 відповідно до складності їхньої сітки. $N_x \times N_y$ – це розмір зображення діапазону, L – це довжина водяного знаку, α визначає площу околиці, а β силу вбудовування водяного знаку. Ці параметри застосовуються до обох алгоритмів. Значення PSNR у наведеній таблиці перевищує 86 дБ для обох алгоритмів. Це свідчить про те, що алгоритми для вбудовування водяних знаків мають хорошу непомітність.

Значення, що відповідає за результативність захисту від атаки, буде розраховане за формулою (2.10), а саме значення $c(w, w^*)$. Дане значення

використовується для порівняння оригінально водяного знаку w та отриманого з сіткової моделі водяного знаку w^*. Воно визначає коефіцієнти їх кореляції.

В якості атак, було обрано атаки, що впливають на геометрію об’єкту, тобто геометричні атаки, а саме атаки обрізання, спрощення, додавання шуму, та комбінація атаки спрощення з обрізанням.

Розпочнемо аналіз стійкості вдосконаленого алгоритму за порядком обраних атак. Результат атаки обрізання наведений у таблиці 3.2.

Таблиця 3.2 – Результати стійкості проти атаки обрізання

Метод	Відсоток моделі	Заєць $c(w,w^*)$	Відсоток моделі	Корова $c(w,w^*)$	Відсоток моделі	Кіт $c(w,w^*)$
Початковий[51]	20%	0.9118	20%	0.8996	5%	0.9411
Вдосконалений		0.9432		0.9253		0.9702
Початковий[51]	50%	0.7021	50%	0.6812	25%	0.7521
Вдосконалений		0.7735		0.7236		0.8062

Результат атаки спрощення наведений у таблиці 3.3.

Таблиця 3.3 – Результати стійкості проти атаки спрощення

Метод	Відсоток моделі	Заєць $c(w,w^*)$	Відсоток моделі	Корова $c(w,w^*)$	Відсоток моделі	Кіт $c(w,w^*)$
Початковий[51]	10%	0.9732	10%	0.9517	5%	0.9441
Вдосконалений		0.9924		0.9843		0.9524
Початковий[51]	30%	0.8657	30%	0.8433	10%	0.8827
Вдосконалений		0.9135		0.8712		0.9021
Початковий[51]	50%	0.7595	45%	0.6727	20%	0.7993
Вдосконалений		0.7957		0.7004		0.8163

Результат атаки додавання шуму наведений у таблиці 3.4.

Таблиця 3.4 – Результати стійкості проти атаки додавання шуму

Метод	Відсоток шуму	Заєць $c(w,w^*)$	Відсоток шуму	Корова $c(w,w^*)$	Відсоток шуму	Кіт $c(w,w^*)$
Початковий[51]	0.15%	0.9833	0.15%	0.9704	0.05%	0.9538
Вдосконалений		0.9892		0.9794		0.9642
Початковий[51]	0.5%	0.9389	0.45%	0.9121	0.1%	0.8689
Вдосконалений		0.9535		0.9203		0.8874

Наостанок здійснимо комбінацію атак спрощення з обрізанням, результат яких наведено у таблиці 3.5.

Таблиця 3.5 – Результати стійкості проти комбінації атак спрощення з обрізанням

Метод	Відсоток спрощення та обрізання	Заєць $c(w, w^{*})$	Відсоток спрощення та обрізання	Корова $c(w, w^{*})$	Відсоток спрощення та обрізання	Кіт $c(w, w^{*})$
Початковий[51]	80% + 30%	0.6701	80% + 30%	0.6259	95% + 10%	0.7301
Вдосконалений		0.7024		0.6335		0.7394

З таблиць видно, що для складної моделі в яку було вбудовано водяний знак за допомогою вдосконаленого алгоритму, водяний знак залишається досить стійкими навіть тоді коли спрощення сітки досягало 15%, амплітуда вектору шуму досягала 0,035%, або поки модель не була обрізана на 30%. З отриманих таблиць, видно що значення коефіцієнта кореляції перевищують 0,5 після того, як моделі з водяним знаком були атаковані спрощенням сітки, додаванням шуму та обрізанням. З отриманих значень $c(w, w^{*})$, можна зробити висновок, що вдосконалений метод є більш захищеним та ефективнішим ніж початковий метод.

З експериментів можна зробити висновок, що запропонована вдосконалена методика нанесення водяних знаків є стійкою до низки поширених атак на тривимірні сітчасті моделі, та показує кращий результат ніж початковий метод.

Оскільки цей підхід і відповідні методи не вимагають попередньої інформації щодо форми 3D-моделі та ґрунтуються лише на локальних характеристиках трикутної сітки, їх можна використовувати для вбудовування цифрових водяних знаків у 3D-моделі будь-якого об'єкта, побудованого за допомогою трикутної сітки.

3.4 Висновки до розділу 3

В даному розділі проведено програмну реалізацію розробленого алгоритму вбудовування водяних знаків в тривимірні моделі. Розглянуто кожен клас, що відповідає за вбудовування та отримання водяного знаку. Також детально ознайомлено з допоміжними методами, що використовувалися в процесі вбудовування водяного знаку в тривимірну модель.

Наведено приклад реалізації вдосконаленого алгоритму, в якому продемонстровано роботу програми, та ознайомлено користувача з інтерфейсом для роботи з розробленим програмним засобом. На практичному прикладі використання описано процедуру вбудовування, перевірки та отримання водяного знаку з тривимірної моделі.

В кінці проведено аналіз стійкості вдосконаленого алгоритму до геометричних атак шляхом застосування різних атак на різні моделі, порівняння результатів з початковим методом вбудовування водяних знаків, та визначення непомітності за допомогою розрахунку значення PSNR. Проаналізовано отримані результати, які свідчать про те, що вдосконалений алгоритм є більш стійким до геометричних атак ніж початковий.

4 ЕКОНОМІЧНА ЧАСТИНА

У цьому розділі проводиться комплексне дослідження економічного потенціалу розробки. Це включає оцінку комерційного потенціалу, прогнозування витрат на виконання наукової роботи та впровадження її результатів, а також розрахунок економічної ефективності наукової роботи за її можливої комерціалізації потенційним інвестором. Проводиться розрахунок ефективності вкладених інвестицій та обчислюється період їх окупності.

В результаті проведеного аналізу буде прийнято обґрунтоване рішення щодо економічної доцільності розробки програмного засобу для вбудовування цифрового водяного знаку. Цей програмний засіб базується на вдосконаленому методі вбудовування, заснованому на 3D-розпізнаванні реберних вершин для тривимірних моделей.

4.1 Оцінка комерційного потенціалу розробки програмного забезпечення

Проведення технологічного аудиту має на меті оцінити комерційний потенціал розробки, створеної в результаті науково-технічної діяльності [53].

Результатом магістерської кваліфікаційної роботи є розробка програмного засобу на основі алгоритму для підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей. Описана розробка вирішує завдання підвищення ефективності вбудовування водяного знаку, зокрема в умовах геометричних перетворень, що спричиняються обробкою тривимірних об'єктів. Розроблений програмний засіб може мати широкий спектр застосувань, особливо в області захисту від копіювання та контролю за використанням графічних тривимірних моделей у різних галузях, таких як комп'ютерна графіка, віртуальна реальність та інші.

Для проведення технологічного аудиту залучено трьох незалежних експертів. У межах даної роботи такими експертами є викладачі кафедри МБІС:

− Карпінець В. В. (к.т.н., доцент каф. МБІС ВНТУ);

− Присяжний Д.П. (асистент каф. МБІС ВНТУ);

− Грицак А. В. (к.т.н., доцент каф. МБІС ВНТУ).

Оцінка комерційного потенціалу буде здійснюватися з використанням критеріїв, представлених у таблиці 4.1 [53]. Процес оцінювання передбачає використання 5-бальної шкали для кожного з критеріїв.

Таблиця 4.1 – Критерії оцінювання комерційного потенціалу розробки та бальна оцінка

Критерії оцінювання та бали					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експл. витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 4.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навч. наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисл. комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої к-ті дозвільних документів на вир-во та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Дані з оцінки комерційного потенціалу науково-технічної розробки були узагальнені та представлені у таблиці 4.2.

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1 – Карпінець В.В.	2 – Салієва О.В.	3 – Грицак А.В.
1	4	4	4
Ринкові переваги (недоліки):			
2	3	3	4
3	4	4	3
4	3	3	4
5	4	4	3
Ринкові перспективи			
6	4	4	3
7	3	3	4
Практична здійсненність			
8	4	4	3
9	3	3	4
10	3	4	4
11	4	3	3
12	3	4	4
Сума балів	$СБ_1 = 42$	$СБ_2 = 43$	$СБ_3 = 43$
Середньоарифметична сума балів	$\overline{СБ} = 42,7$		

Висновок можна зробити на підставі даних у таблиці 4.2, де середньоарифметична сума балів, отриманих на основі експертних висновків, становить 42,7. Це свідчить про високий рівень комерційного потенціалу розробки. З врахуванням високої середньої оцінки можна припустити, що розробка має значний практичний і комерційний інтерес, що може виявитися в успішному впровадженні на ринку або в інших сферах застосування.

Наукова новизна розробки полягає у вдосконаленні методу вбудовування невидимого водяного знаку на основі 3D-розпізнавання реберних вершин для тривимірних моделей. Це представляє собою новий підхід до розв'язання проблеми стійкості водяного знаку в умовах геометричних операцій, які можуть виникнути при обробці тривимірних об'єктів. Удосконалений алгоритм враховує специфіку тривимірних структур та дозволяє підвищити ефективність захисту від незаконного видалення або модифікації водяного знаку в тривимірних моделях.

Розробка програмного засобу має на меті практичну реалізацію удосконаленого методу вбудовування невидимого водяного знаку на основі 3D-розпізнавання реберних вершин для тривимірних моделей, що вирізняється підвищеною стійкістю до геометричних атак. Це підвищує захист ЦВЗ від несанкціонованого видалення чи модифікації водяного знаку в тривимірних моделях.

Розроблений програмний додаток, використовуючи вдосконалений алгоритм вбудовування невидимого водяного знаку через 3D-розпізнавання реберних вершин, забезпечує ефективне вбудовування водяного знаку в тривимірні моделі. Використання технологій 3D-розпізнавання дозволяє алгоритму інтегрувати водяний знак таким чином, щоб він залишався невидимим для спостерігача, забезпечуючи високу стійкість до геометричних операцій та зберігаючи при цьому оригінальний вигляд тривимірної моделі.

Враховуючи отримані результати дослідження розробленого програмного додатку, можна зазначити, що він продемонстрував високу ефективність у вбудовуванні невидимого водяного знаку в тривимірні моделі. Отримані результати підтверджують високий рівень захисту від незаконного видалення, модифікації або порушення цілісності водяного знаку, забезпечуючи його стійкість до геометричних атак на тривимірну модель. Таких результатів досягнуто шляхом модифікації головного методу вбудовування ЦВЗ за допомогою алгоритму пошуку реберних кутових вершин для тривимірних моделей.

Враховуючи виявлені переваги розробленого методу, порівняємо його з аналогами, використовуючи таблицю 4.3, де представлені основні технічні показники аналога і нового програмного продукту.

Таблиця 4.3 – Основні технічні показники аналога і нового програмного продукту

Показники, %	Аналог	Нова розробка	Відношення параметрів нової розробки до параметрів аналога
Ефективність вбудовування	60%	95%	1,6
Надійність	40%	90%	2,25

Продовження таблиці 4.3

Стійкість до геометричних атак	5%	90%	18
Непомітність водяного знаку	55%	100%	1,8
Швидкість обробки	70%	100%	1,4
Простота експлуатації	85%	100%	1,17

Розроблений додаток відзначається високою ефективністю вбудовування, значною стійкістю до геометричних операцій і високою непомітністю водяного знаку, що робить його значущим вдосконаленням порівняно з існуючими аналогами.

Таким чином, розроблений програмний додаток, використовуючи вдосконалений алгоритм вбудовування невидимого водяного знаку шляхом вбудовування ЦВЗ на основі 3D-розпізнавання реберних вершин, виокремлюється високою ефективністю, стійкістю та непомітністю водяного знаку. Порівняно із конкуруючими аналогами, розробка видається перспективною, пропонуючи значне поліпшення у таких аспектах, як ефективність вбудовування (95%), стійкість до геометричних операцій (90%), та непомітність водяного знаку (100%). Зазначена значущість розробки виражається у її здатності надійно захищати інтелектуальну власність у галузі обробки тривимірних графічних об'єктів, що робить її важливим інструментом для сучасного цифрового середовища та застосування в різних галузях, де забезпечення конфіденційності та цілісності є вирішальними факторами.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Процес прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи складається з трьох етапів [53].

На першому етапі проводиться розрахунок витрат, які безпосередньо пов'язані з участю виконавців у даному розділі роботи. Це включає витрати, що виникають при розробці, тестуванні та вдосконаленні програмного забезпечення.

На другому етапі виконується розрахунок загальних витрат на виконання роботи, які виходять за рамки безпосереднього внеску виконавців. Це охоплює витрати на необхідні матеріали, обладнання, послуги та інші загальні витрати.

Третій етап включає в себе прогнозування загальних витрат на виконання та впровадження результатів роботи. Тут розглядаються витрати, пов'язані з впровадженням розробленої інформаційної технології, такі як навчання персоналу, підтримка та інші витрати. Цей процес спрямований на прогнозування всіх ключових аспектів витрат, які можуть виникнути під час виконання проекту, забезпечуючи об'єктивну оцінку його фінансової складової.

Подальші розрахунки включатимуть витрати, пов'язані з діяльністю обох учасників проекту: керівника проекту і розробника програмного забезпечення. Це охоплює безпосередні витрати на трудові ресурси кожного учасника. Крім того, враховуються витрати на необхідне обладнання, програмний інструментарій, технічну підтримку та інші ресурси, необхідні для успішної реалізації проекту.

Заробітна плата визначається відповідно до формули 4.1:

$$З_о = \sum_{i=1}^{k} \frac{М}{Т_р} \times t \quad (4.1)$$

де M – місячна зарплата розробника;

T_p – кількість робочих днів у місяці, $Т_р = 23$ дні;

t – кількість днів роботи.

Обраховуємо заробітну плату для керівника та розробника, за умов, що наведені у таблиці 4.4.

Таблиця 4.4 – Витрати по заробітній платі

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату
Керівник проекту	38 000	1 652,17	46	76 000
Розробник ПЗ	28 000	1 217,4	46	56 000

Обраховуємо заробітну плату керівника проекту:

$$З_{к.} = \frac{38\,000}{23} \times 46 = 76\,000\ (грн.)$$

Обраховуємо заробітну плату розробника ПЗ:

$$З_{р.} = \frac{28\,000}{23} \times 46 = 56\,000\ (грн.)$$

Основна заробітна плата керівника проекту та розробника ПЗ становить:

$$З_{о} = З_{к.} + З_{р.} = 76\,000 + 56\,000 = 132\,000\ (грн.)$$

Додаткова заробітна плата розраховується у розмірі 11% від основної зарплати розробника:

$$З_{дод} = З_{о} \times 0{,}11 = 132\,000 \times 0{,}11 = 14\,520\ (грн.)$$

Нарахування на зарплату складають 22% від сум базової та додаткової грошової оплати (формула 4.2):

$$З_{н} = \left(З_{о} + З_{дод}\right) \times 0{,}22 \tag{4.2}$$

Величина нарахувань на заробітну плату розробника визначається як 22% від суми основної та додаткової заробітної плати.

$$З_{н} = (132\,000 + 14\,520\ ) \times 0{,}22 = 32\,234{,}4\ (грн.)$$

Визначення амортизації для обладнання, комп'ютерів та приміщень, використовуваних під час виконання даного етапу роботи, здійснюється шляхом застосування спеціальної формули:

$$А_{обл} = \frac{Ц_{б}}{Т_{в}} \times \frac{t_{вик}}{12} \tag{4.3}$$

де $Ц_{б}$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_{в}$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Обраховуємо амортизацію для офісного приміщення, ноутбука, графічної карти та сканера для тривимірних моделей. Орієнтовно 2 місяці визначено як фактична тривалість розробки програмного забезпечення.

Для офісного приміщення:

$$А_{оп} = \frac{145\ 000}{20} \times \frac{2}{12} = 1\ 208{,}3\ (грн.)$$

Для ноутбука:

$$А_{ноут} = \frac{2\ 500}{3} \times \frac{2}{12} = 1\ 041{,}6\ (грн.)$$

Для Wi-fi-роутер:

$$А_{роут} = \frac{25\ 000}{4} \times \frac{2}{12} = 138{,}8 (грн.)$$

Для додаткового монітора:

$$А_{мон} = \frac{5\ 000}{3} \times \frac{2}{12} = 277{,}7\ (грн.)$$

Для графічної карти:

$$А_{грк} = \frac{12\ 900}{3} \times \frac{2}{12} = 716{,}6\ (грн.)$$

Для сканера:

$$А_{с} = \frac{12\ 000}{2} \times \frac{2}{12} = \ 1\ 000\ (грн.)$$

Проведені розрахунки наведено до таблиці 4.5.

Таблиця 4.5 – Амортизаційні відрахування

Найменування	Балансова вартість (грн.)	Термін використання (років)	Фактична тривалість в-ня, (міс.)	Величина ам. відрахувань, (грн.)
Офісне приміщення	145 000	20	2	1 208,3
Ноутбук	25 000	4	2	1041,6
Wi-fi-роутер	2 500	3	2	138,8
Додатковий монітор	5 000	3	2	277,7
Графічна карта	12 900	3	2	716,6

Продовження таблиці 4.5

Сканер для тривимірних моделей	12 000	2	2	1 000
			Всього	4 383

У розробці використовувалися різноманітні матеріали, інформацію про які можна знайти в таблиці 4.6. Ця таблиця містить деталі щодо кількості та вартості використаних матеріалів у процесі розробки. Вона надає конкретні дані щодо ресурсів, використаних у проекті, що допомагає оцінити загальні витрати та ефективність їх використання.

Таблиця 4.6– Використані матеріали

Найменування	Кількість	Ціна за штуку, грн.	Сума, грн.
Упаковка паперу формату А4	2	165	330
Мишка	1	350	350
Флеш-накопичувач USB	1	480	480
Маркери	3	20	60
Ручка	3	15	45
Степлер та скоби (набір)	1	120	120
Блокнот-записник	1	80	80
Папка для паперів	3	16	48
Файли (набір 100 шт.)	1	50	50
Всього:			1 563

Витрати на силову електроенергію розраховують за формулою:

$$В_е = \sum_{i=1}^{n} \frac{W_{yi} \times t_i \times Ц_в \times К_{впі}}{\eta_i} \quad (4.4)$$

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i– тривалість роботи обладнання на етапі дослідження, год;

$Ц_в$ – вартість 1 кВт-години електроенергії, 7,5 грн; (вартість електроенергії визначається за даними енергопостачальної компанії);

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, η_i <1.

Вартість електроенергії, використовуваної для розробки та тестування програмного забезпечення, визначається з урахуванням енергоспоживання, пов'язаного із функціонуванням різноманітного обладнання та технічних засобів. Витрати на силову електроенергію розраховуються за ставкою 7,5 грн за 1 кВт-годину.

$$В_е = \left(\frac{0{,}15 \times 264 \times 0{,}75 \times 7{,}5}{0{,}7}\right) + \left(\frac{0{,}02 \times 264 \times 0{,}7 \times 7{,}5}{0{,}8}\right) + \left(\frac{0{,}7 \times 44 \times 0{,}8 \times 7{,}5}{0{,}6}\right) = 660{,}85 \text{ (грн)}$$

Проведені розрахунки занесено до таблиці 4.7.

Таблиця 4.7 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Коефіцієнт, що враховує використання потужності	Коефіцієнт корисної дії обладнання	Сума, грн
Ноутбук	0,15	264	0,75	0,7	318,2
Wi-fi роутер	0,02	264	0,7	0,8	34,65
Сканер для тривимірних моделей	0,7	44	0,8	0,6	308
Всього					660,85

Витрати на «Службові відрядження» розраховуються як 23% від загальної суми основної заробітної плати за допомогою формули 4.5.

$$В_{св} = (З_о + З_р) \times \frac{Н_{св}}{100\%} \tag{4.5}$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», 23%.

$$В_{св} = 132\,000 \times 0{,}23 = 30\,360 \text{ (грн.)}$$

Витрати за статтею «Інші витрати» розраховуються як 50% від суми основної заробітної плати дослідників та робітників за формулою 4.6.

$$І_в = (З_о + З_р) \times \frac{Н_в}{100\%} \tag{4.6}$$

де $Н_в$ – норма нарахування за статтею «Інші витрати», 50%.

$$І_в = 132\,000 \times 0{,}5 = 66\,000 \text{ (грн.)}$$

Отже, обрахуємо витрати на проведення науково-дослідної роботи:

$$В_{заг} = 132\,000 + 14\,520 + 32\,234{,}4 + 4\,383 + 1\,563 + 660{,}85 + 30\,360 + 66\,000 = 281\,721{,}25 (грн.)$$

Витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів обчислюються відповідно до формули 4.7.

$$ЗВ = \frac{В_{заг}}{\eta} \quad (4.7)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи. Визначено науковим керівником як 0,7.

$$ЗВ = \frac{281\,721{,}25}{0{,}7} = 402\,458{,}9 \text{ (грн.)}$$

Таким чином, після врахування всіх витрат, прогнозованих для виконання та впровадження результатів виконаної наукової роботи, можна очікувати витрати на суму 402 458,9 грн.

4.3 Розрахунок економічної ефективності наукової роботи за її можливої комерціалізації потенційним інвестором

У контексті ринкових умов важливим показником успіху для потенційного інвестора є можливість досягнення позитивного фінансового ефекту внаслідок впровадження розроблюваного програмного додатку. Збільшення чистого прибутку виступає ключовим фактором, оскільки це сприяє залученню додаткових інвестицій, поліпшенню фінансових показників та підвищенню конкурентоспроможності. Отримання додаткових коштів завдяки зростанню чистого прибутку не лише оптимізує фінансові результати інвестора, але й може позитивно вплинути на прийняття стратегічного рішення щодо комерціалізації даної розробки.

Виконання даної наукової роботи та впровадження отриманих результатів планується завершити протягом приблизно одного року. Цей термін включає в себе всі етапи вивчення, аналізу та розробки, необхідні для успішного завершення проекту. При цьому передбачається, що позитивні висновки від впровадження

результатів дослідження будуть очікуватися вже на початковому етапі реалізації, що сприятиме оперативному підтвердженню ефективності розробленого рішення.

З метою більш детального прогнозування та кількісного оцінювання позитивних результатів протягом кількох років, проведемо ретельний аналіз та обчислення збільшення чистого прибутку підприємства ($\Delta П_i$) для кожного року, коли очікується досягнення позитивних результатів від впровадження розробки. Для цього використовується формула 4.8.

$$\Delta П_i = (\pm\Delta Ц_0 \times N + Ц_0 \times \Delta N) \times \lambda \times \rho \times (1 - \frac{\vartheta}{100\%}) \quad (4.8)$$

де $\pm\Delta Ц_0$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році.

N – основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_0$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_0 = Ц_б + \pm\Delta Ц_0$;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році. Зазвичай таким показником може бути зростання попиту на науково-технічну розробку в аналізованому році (відносно року до впровадження цієї розробки);

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість, λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ =0,2...0,5;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, ϑ =18%.

Вартість програмного продукту у році до впровадження результатів розробки становить 870 грн. Проаналізуємо збільшення кількості споживачів продукту

протягом 3 років від покращення його певних характеристик. За перший рік кількість одиниць ПЗ збільшиться на 750 одиниць, за другий рік – на 1020, а за третій – на 1400. Кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки становить 750.

Зміна вартості програмного продукту від впровадження результатів науково-технічної розробки в аналізовані періоди часу становить 600 грн.

Потрібно спрогнозувати збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового. Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1 = (600 \times 750 + 1\,470 \times 700) \times 0{,}8333 \times 0{,}4 \times \left(1 - \frac{18\%}{100\%}\right)$$
$$= 492\,981{,}1\ (\text{грн.})$$

Збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року складе:

$$\Delta П_2 = (600 \times 750 + 1\,470 \times (700 + 1020)) \times 0{,}8333 \times 0{,}4 \times \left(1 - \frac{18\%}{100\%}\right)$$
$$= 992\,761{,}1\ (\text{грн.})$$

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом третього року складе:

$$\Delta П_3 = (600 \times 750 + 1\,470 \times (700 + 1020 + 1400)) \times 0{,}8333 \times 0{,}4$$
$$\times \left(1 - \frac{18\%}{100\%}\right) = 1\,678\,733{,}7\ (\text{грн.})$$

Отже, згідно з проведеними розрахунками можна визначити, що прогнозований комерційний ефект від впровадження розробки проявляється в значному підвищенні чистого прибутку підприємства.

Наступним етапом є розрахунок приведеної вартості збільшення чистого прибутку для всіх можливих прибутків, які потенційний інвестор може отримати внаслідок можливого впровадження та комерціалізації науково-технічної розробки. Цей розрахунок виконується за допомогою формули 4.9.

$$ПП = \sum_{i=1}^{T} \frac{ΔП_i}{(1+\tau)^t} \tag{4.9}$$

де $ΔП_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{492\ 981{,}1}{(1+0{,}12)^1} + \frac{992\ 761{,}1}{(1+0{,}12)^2} + \frac{1\ 678\ 733{,}7}{(1+0{,}12)^3} = 2\ 426\ 474{,}28\ (грн.)$$

Подальший обчислення включає визначення початкових інвестицій (PV), які потенційний інвестор повинен вкласти для успішного впровадження та комерціалізації науково-технічної розробки. Для цього можна скористатися відповідною формулою:

$$PV = k_{інв} \times ЗВ \tag{4.10}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$ =2...5, але може бути і більшим;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 \times 402\ 458{,}9\ = 804\ 917{,}8\ (грн.)$$

Абсолютний економічний ефект ($E_{абс}$) або чистий приведений дохід для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки обчислюється наступним чином:

$$E_{абс} = ПП - PV \tag{4.11}$$

де ПП – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

$$E_{абс} = 2\ 426\ 474{,}28\ - 804\ 917{,}8 = 1\ 621\ 556{,}48$$

Високе позитивне значення для $E_{абс}$ свідчить про значущий інтерес інвесторів у впровадженні та комерціалізації даної науково-технічної розробки. Проте, для остаточного прийняття рішення щодо впровадження на ринок цієї розробки, більше відомостей та аналізу є необхідними.

Необхідно розрахувати внутрішню економічну дохідність $\text{Е}_{\text{в}}$ за формулою 4.12.

$$\text{Е}_{\text{в}} = \sqrt[\text{Т}_{\text{ж}}]{1 + \frac{E_{абс}}{PV}} - 1 \qquad (4.12)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$\text{Т}_{\text{ж}}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$\text{Е}_{\text{в}} = \sqrt[3]{1 + \frac{1\ 621\ 556{,}48}{804\ 917{,}8}} - 1 = 0{,}4446$$

Визначаємо бар’єрну ставку дисконтування $\tau_{мін}$. Це найнижчий рівень внутрішньої економічної доцільності інвестицій, який визначає нижню межу, нижче якої інвестиції у впровадження науково-технічної розробки та її комерціалізацію стають неприйнятними.

$$\tau_{мін} = 0{,}12 + 0{,}2 = 0{,}32$$

$\text{Е}_{\text{в}}$ перевищує $\tau_{мін}$, що свідчить про можливий інтерес потенційного інвестора у фінансуванні впровадження науково-технічної розробки та її введенні на ринок, іншими словами, у її комерціалізації.

Проводимо розрахунок періоду окупності інвестицій ($\text{Т}_{\text{ок}}$), які потенційний інвестор може вкласти в реалізацію та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_{в}} = \frac{1}{0,4446} = 2,25$$

Таким чином, $T_{ок} < 3$-х років, що вказує на те, що науково-технічна розробка є комерційно привабливою і може стимулювати зацікавленість потенційного інвестора у фінансуванні впровадження та виведенні її на ринок.

4.4 Висновок до розділу 4

У даному розділі здійснено оцінювання комерційного потенціалу розроблюваного програмного додатку, що є реалізацією вдосконаленого алгоритму вбудовування цифрового водяного знаку у тривимірні моделі. Удосконалення вдалося досягнути шляхом підвищення стійкості ЦВЗ невидимого водяного знаку від геометричних операцій перетворення.

Здійснено технологічний аудит з використанням участі трьох незалежних експертів. Результати аудиту вказують на високий рівень комерційного потенціалу цієї розробки.

Крім того, було здійснено порівняльний аналіз з аналогічними розробками, що підтверджує конкурентоспроможність та унікальність запропонованого методу вбудовування в контексті покращення стійкості водяних знаків під час геометричних операцій перетворення для тривимірних моделей.

Враховуючи всі прогнозовані витрати на виконання та впровадження результатів наукової роботи, здійснені розрахунки вказують на очікувану суму витрат у розмірі 402 458,9 грн.

Здійснено розрахунок приведеної вартості збільшення чистого прибутку включав аналіз всіх потенційних доходів, які може отримати потенційний інвестор внаслідок можливого впровадження та комерціалізації науково-технічної розробки. Отримана величина складає 2 426 474,28 грн.

Річна ефективність інвестицій, вкладених у наукову розробку, становить 44,46%, що перевищує мінімальну бар'єрну ставку дисконтування на рівні 32%. Це свідчить про потенційний інтерес інвесторів до фінансування даної розробки.

Час, необхідний для повного повернення інвестицій, які потенційний інвестор може вкласти у впровадження та комерціалізацію науково-технічної розробки, складає 2,25 роки. Це також підтверджує раціональність і доцільність фінансування даної нової розробки.

Отже, аналіз отриманих економічних показників підтверджує, що розроблюваний програмний додаток, який ґрунтується на вдосконаленому методі вбудовування ЦВЗ на основі 3D-розпізнавання реберних вершин для тривимірних моделей, виявив високий комерційний потенціал. Таким чином, його подальше впровадження вважається обґрунтованим.

ВИСНОВОК

Отже, в даній магістерській кваліфікаційній роботі здійснено дослідження, удосконалення та практична реалізація методу вбудовування невидимого ЦВЗ на основі 3D-розпізнавання реберних вершин для тривимірних моделей. Головною метою роботи є підвищення стійкості невидимого цифрового водяного знаку від геометричних операцій перетворення.

Перший розділ роботи присвячений загальному аналізу стеганографічного захисту інформації на основі цифрового водяного знаку. Детально проаналізовано найпоширеніші алгоритми вбудовування ЦВЗ та їх характеристики. Також досліджено використання ЦВЗ для тривимірних моделей, визначено їхню ефективність та потенційні загрози для ЦВЗ.

У другому розділі здійснено удосконалення обраного алгоритму вбудовування ЦВЗ для тривимірних моделей для підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення. Початковий досліджуваний метод вбудовування ЦВЗ використовує алгоритм пошуку опорних точок для вбудовування водяного знаку, що базується на знаходженні оцінки різкої зміни площі. Даний алгоритм має ряд недоліків, а саме опорні точки, що вираховуються для заданої сітчастої моделі, не завжди є надійними для вбудовування інформації, а також їх пошук та впорядкування може бути ускладнене геометричними атаками на модель. Для усунення такого суттєвого недоліку було впроваджено алгоритм 3D-розпізнавання реберних вершин для пошуку опорних точок. Його застосування спрямоване на пошук опорних точок, які виявилися більш стійкими для вбудовування водяного знаку. Використання алгоритму 3D-розпізнавання реберних вершин у вдосконаленому методі забезпечує більшу стійкість до геометричних атак, що робить його надійнішим порівняно з початковим методом пошуку опорних точок для вбудовування водяного знаку. Також, у даному розділі здійснено розробку алгоритму програмного додатку, що буде реалізовувати удосконалений метод вбудовування невидимого цифрового знаку у тривимірну модель.

У третьому розділі даної магістерської кваліфікаційної роботи описано програмну реалізацію розробленого алгоритму вбудовування водяних знаків в тривимірні моделі. Розглянуто кожен клас, що відповідає за вбудовування та отримання водяного знаку. Наведено приклад реалізації вдосконаленого алгоритму, який демонструє роботу програми.

Проведено аналіз стійкості вдосконаленого методу, використовуючи різні види атак на різноманітні тривимірні моделі. Порівняння результатів з початковим методом та оцінювання непомітності здійснювалися за допомогою розрахунку значень PSNR. Після цього було проведено аналіз отриманих результатів, що підтверджує, що вдосконалений алгоритм є більш стійким до геометричних атак порівняно з початковим методом.

Стійкість водяного знаку, вбудованого у складну тривимірну модель, залишається високою при спрощенні сітки до 15%, амплітуді шуму до 0,035% та обрізці моделі на 30%. Таким чином, зроблено висновок, що удосконалений метод є більш захищеним та ефективнішим, ніж початковий.

У четвертому розділі було проведено оцінювання комерційного потенціалу розробленого програмного додатку, що є реалізацією вдосконаленого методу вбудовування цифрового водяного знаку у тривимірні моделі. Аналіз економічних показників підтверджує високий комерційний потенціал цього програмного засобу. Таким чином, подальше впровадження даного програмного продукту вважається обґрунтованим.

В результаті успішної реалізації поставлених на початку роботи задач вдалося досягти ключової мети цієї роботи – підвищити стійкість невидимого водяного знаку від геометричних операцій перетворення. Це було досягнуто завдяки ефективному вдосконаленню методу вбудовування ЦВЗ, який ґрунтується на 3D-розпізнаванні реберних вершин для тривимірних моделей. Отримані результати свідчать про високий рівень стійкості водяних знаків, вбудованих у тривимірні моделі, що робить їх менш вразливими до геометричних операцій перетворення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Su J. K., Eggers J. J., Girod B. Analysis of digital watermarks subjected to optimum linear filtering and additive noise. Signal Processing. 2001. Т. 81, № 6. С. 1141–1175. URL: https://doi.org/10.1016/s0165-1684(01)00038-x (дата звернення: 9.10.2023).

2. Ashourian M., Enteshari R., Jeonghee Jeon. Digital watermarking of three-dimensional polygonal models in the spherical coordinate system. Proceedings Computer Graphics International, 2004., м. Crete, Greece. URL: https://doi.org/10.1109/cgi.2004.1309270 (дата звернення: 9.10.2023).

3. Robust Watermarking of Polygonal Meshes Based on Vertex Norms Variance Distortion / Y. Ben Amar та ін. Journal of Global Information Management. 2017. Т. 25, № 4. С. 46–60. URL: https://doi.org/10.4018/jgim.2017100104 (дата звернення: 10.10.2023).

4. Borah S., Borah B. Watermarking Techniques for Three Dimensional (3D) Mesh Authentication in Spatial Domain. 3D Research. 2018. Т. 9, № 3. URL: https://doi.org/10.1007/s13319-018-0194-7 (дата звернення: 10.10.2023).

5. Wang X., Zhan Y. A zero-watermarking scheme for three-dimensional mesh models based on multi-features. Multimedia Tools and Applications. 2017. Т. 78, № 19. С. 27001–27028. URL: https://doi.org/10.1007/s11042-017-4666-1 (дата звернення: 10.10.2023).

6. Мовчанюк М. Т., Салієва О. В.Дослідження методів пошуку опорних точок для вбудовування ЦВЗ для підвищення його стійкості від геометричних операцій перетворення. Молодь в науці: дослідження, проблеми, перспективи (МН-2024) : Міжнар. науково-практ. інтернет-конф., м. Вінниця, 15 жовт. 2023 р. Вінниця, 2023. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/19601 (дата звернення: 17.10.2023).

7. What Is a Digital Watermark?. Makeuseof: веб-сайт. URL:https://www.makeuseof.com/what-is-a-digital-watermark/ (дата звернення: 12.10.2023).

8. Evsutin O., Melman A., Meshcheryakov R. Digital Steganography and Watermarking for Digital Images: A Review of Current Research Directions. IEEE Access. 2020. Vol. 8. — P.166589 -166610

9. Юсеф Ш. М., Гатварі Н. Адаптивні водяні знаки для відео, що інтегрують нечітку модель сприйняття людської зорової системи на основі вейвлетів. Мультимедійні засоби та програми. 2014. Т. 73, № 3. С. 1545–1573. URL: https://www.researchgate.net/publication/278148701_Adaptive_video_watermarking_integrating_a_fuzzy_wavelet-based_human_visual_system_perceptual_model (дата звернення: 10.11.2023).

10. What is Robust Watermarking. IGI Global: веб-сайт. URL: https://www.igi-global.com/dictionary/multimedia-encryption-watermarking-wireless-environment/25480 (дата звернення: 10.10.2023).

11. Zhe-Ming L., Shi-Ze G. Lossless Information Hiding in Images. Zhejiang University Press Co. 2017.

12. Yongqiang M. Comparative and Analysis of spatial domain and frequency domaindigital image watermarking algorithm. Journal of Physics: Conference Series. 2020. No. 1486. — P. 5

13. Бегум M. Методи нанесення водяних знаків на цифрові зображення: огляд. Information. 2020. Т. 11, № 2. С. 2–38. URL: https://www.mdpi.com/2078-2489/11/2/110 (дата звернення: 11.10.2023).

14. Kenneth L. Levy, Geoffrey B. Rhoads. DIGITAL WATERMARKING SYSTEMS AND METHODS. 2005.

15. Embaby A. Al., Mohamed A. , Wahby Shalaby, Khaled Mostafa Elsayed. Digital Watermarking Properties, Classification and Techniques. International Journal of Engineering and Advanced Technology (IJEAT). 2020. Vol. 9, no. 3.

16. Rawat R., Kaushik N., Tiwari S. Digital Watermarking Techniques. International Journal of Advanced Research in Computer and Communication Engineering. 2016. Vol. 5, no. 4.

17. DCT Transform Digital Watermarking. VOCAL:веб-сайт. URL: https://vocal.com/video/dct-transform-digital-watermarking/ (дата звернення: 12.10.2023).

18. Полігональна сітка. Вікіпедія: веб-сайт. URL: https://uk.wikipedia.org/wiki/Полігональна_сітка#/media/Файл:Dolphin_triangle_mesh.png (дата звернення: 14.10.2023).

19. What Are Polygons in 3D Modeling. CGI Furniture: веб-сайт. URL: https://cgifurniture.com/what-are-polygons-in-3d-modeling/ (дата звернення: 15.10.2023).

20. Ryutarou O., Hiroshi M., Masaki A.. Watermarking Three-Dimensional Polygonal ModelsThrough Geometric and Topological Modifications. IEEE Journal on Selected Areas in Communications. 1997. — P. 1- 38.

21. Куо Ч., Ченг Ш.-Ч. Сліпа стійка схема нанесення водяних знаків для тривимірних трикутних сітчастих моделей з використанням 3D-розпізнавання реберних вершин. Asian Journal of Health and Information Sciences. 2009. Т. 4, № 1. С. 36–63. URL: https://static.aminer.org/pdf/PDF/000/236/457/a_new_mixed_spatial_domain_watermarking_of_three_dimensional_triangle.pdf (дата звернення: 12.10.2023).

22. Uccheddu F., Corsini M., Barni M. Wavelet-based blind watermarking of 3D models. Proceedings of the 6th workshop on Multimedia & Security, Magdeburg, 20 August 2004.

23. Чжу Б., Фан С. Надійне нанесення водяних знаків на сліпе зображення з використанням різниці коефіцієнтів середньої частоти між блоками. Останні розробки та застосування водяних знаків зображень. 2023. Т. 12, № 19. URL: https://www.mdpi.com/2079-9292/12/19/4117 (дата звернення: 21.10.2023).

24. G. Bors A. Optimized 3D Watermarking for Minimal Surface Distortion. IEEE Transactions on Image Processing. 2013. Vol. 22, no. 5. — P. 1822–1835.

25. Jabra S. B., Zagrouba E. A New Approach of 3D Watermarking Based on Image Segmentation. IEEE Xplore. 2008. —P. 1–5.

26. Polygonal Modelling. Devopedia: веб-сайт. URL: https://devopedia.org/polygonal-modelling (дата звернення: 14.10.2023).

27. 3D model. TechTarget: веб-сайт. URL: https://www.techtarget.com/whatis/definition/3D-model (дата звернення: 16.10.2023).

28. Сітка з трикутників. Вікіпедія: веб-сайт. URL: https://uk.wikipedia.org/wiki/Сітка_з_трикутників (дата звернення: 15.10.2023).

29. What is Polygonal Modeling?. ThePro3DStudio: веб-сайт. URL: https://professional3dservices.com/blog/polygonal-modeling.html (дата звернення: 17.10.2023).

30. What Are Polygons in 3D Modeling. CGI Furniture: веб-сайт. URL: https://cgifurniture.com/what-are-polygons-in-3d-modeling/ (дата звертання: 15.10.2023).

31. Ryutarou O., Hiroshi M., Masaki A.. Watermarking Three-Dimensional Polygonal ModelsThrough Geometric and Topological Modifications. IEEE Journal on Selected Areas in Communications. 1997. — P. 1- 38.

32. Ryutarou O., Hiroshi M., Masaki A.. Watermarking Three-Dimensional Polygonal ModelsThrough Geometric and Topological Modifications. IEEE Journal on Selected Areas in Communications. 1997. —P. 1- 38.

33. Hang Z., Weiming Z., Kejiang C., Weixiang L., and Nenghai Y.. Three-Dimensional Mesh Steganography and Steganalysis: A Review. Cryptography and Security. 2021. —P. 1–20.

34. Uccheddu F., Corsini M., Barni M. Wavelet-based blind watermarking of 3D models. Proceedings of the 6th workshop on Multimedia & Security, Magdeburg, 20 August 2004.

35. El Hanafy Y. M. Watermarking 3D Models. The German University in Cairo. 2022. — P. 1–87.

36. Giao N. Ph.,Suk-Hwan L.,Oh-Heum K., Ki-Ryong K. A 3D Printing Model Watermarking Algorithm Based on 3D Slicing and Feature Points. Electronics. 2018. Vol. 7, no. 23. —13 p.

37. A Comprehensive Survey on Three-Dimensional Mesh Watermarking. IEEExplore: веб-сайт. URL: https://ieeexplore.ieee.org/abstract/document/4694850 (дата звернення: 16.10.2023).

38. Kai W., Guillaume L., Florence D., Atilla B. A Comprehensive Survey on Three-Dimensional Mesh Watermarking. IEEE TRANSACTIONS ON MULTIMEDIA. 2008. Vol. 10, no. 8. —P. 1513–1527.

39. A new approach of 3D watermarking based on image segmentation. Semanticsscholar: веб-сайт. URL: https://www.semanticscholar.org/paper/A-new-approach-of-3D-watermarking-based-on-image-Jabra-Zagrouba/7bb5c3d3479fa2af3b83155d3a6cc860be2ae4ea (дата звернення: 18.10.2023).

40. Zagrouba E., Jabra S. B. A Robust Embedding Scheme and an Efficient Evaluation Protocol for 3D Meshes Watermarking. International Journal of Computer Vision and Image Processing. 2011. Vol. 1, no. 2. —P. 13–29.

41. A Robust Embedding Scheme and an Efficient Evaluation Protocol for 3D Meshes Watermarking. IGI Global: веб-сайт. URL: https://www.igi-global.com/chapter/robust-embedding-scheme-efficient-evaluation/77034 (дата звернення: 18.10.2023).

42. Modigari N., Valarmathi M., Jani A. Watermarking techniques for three-dimensional (3D) mesh models: a survey. Multimedia Systems. 2021.

43. Yinghui W., Jing L., Yajie Y., Douli M., Ruijiao L.. 3D model watermarking algorithm robust to geometric attacks. IET Image Processing. 2017. Vol. 11, no. 10. —P. 822–832.

44. MODELLING OF GEOMETRIC ATTACKS FOR DIGITAL IMAGE WATERMARKING. Semanticscholar: веб-сайт. URL: https://www.semanticscholar.org/paper/MODELLING-OF-GEOMETRIC-ATTACKS-FOR-DIGITAL-IMAGE-Jabade/ffe58dbde8e9b3b7a43ecedce8a6e990996bf7d3 (дата звернення: 19.10.2023).

45. Vaishali S. J., Sachin R. G. COMPREHENSIVE SURVEY OF IMAGE WATERMARKING. International Journal of Advances in Engineering & Technology. 2013. Vol. 6, no. 3. —P. 1271–1282.

46. Wenbo W., Jun W., Yunming Zh., Jing L., Hui Y., Jiande S.. A comprehensive survey on robust image watermarking. Neurocomputing. 2021. Vol. 488. — P. 226–247.

47. Hadid M., Solima M., Darwish A., Hassanien A.. Robust and blind watermark to protect 3D mesh models against connectivity attacks. Eighth International Conference on Intelligent Computing and Information Systems (ICICIS). 2017. — P. 11-20.

48. Jae-Won Ch., Rémy Pr., Ho-Youl J.. An Oblivious Watermarking for 3-D Polygonal Meshes Using Distribution of Vertex Norms. IEEE Transactions on Signal Processing. 2006.

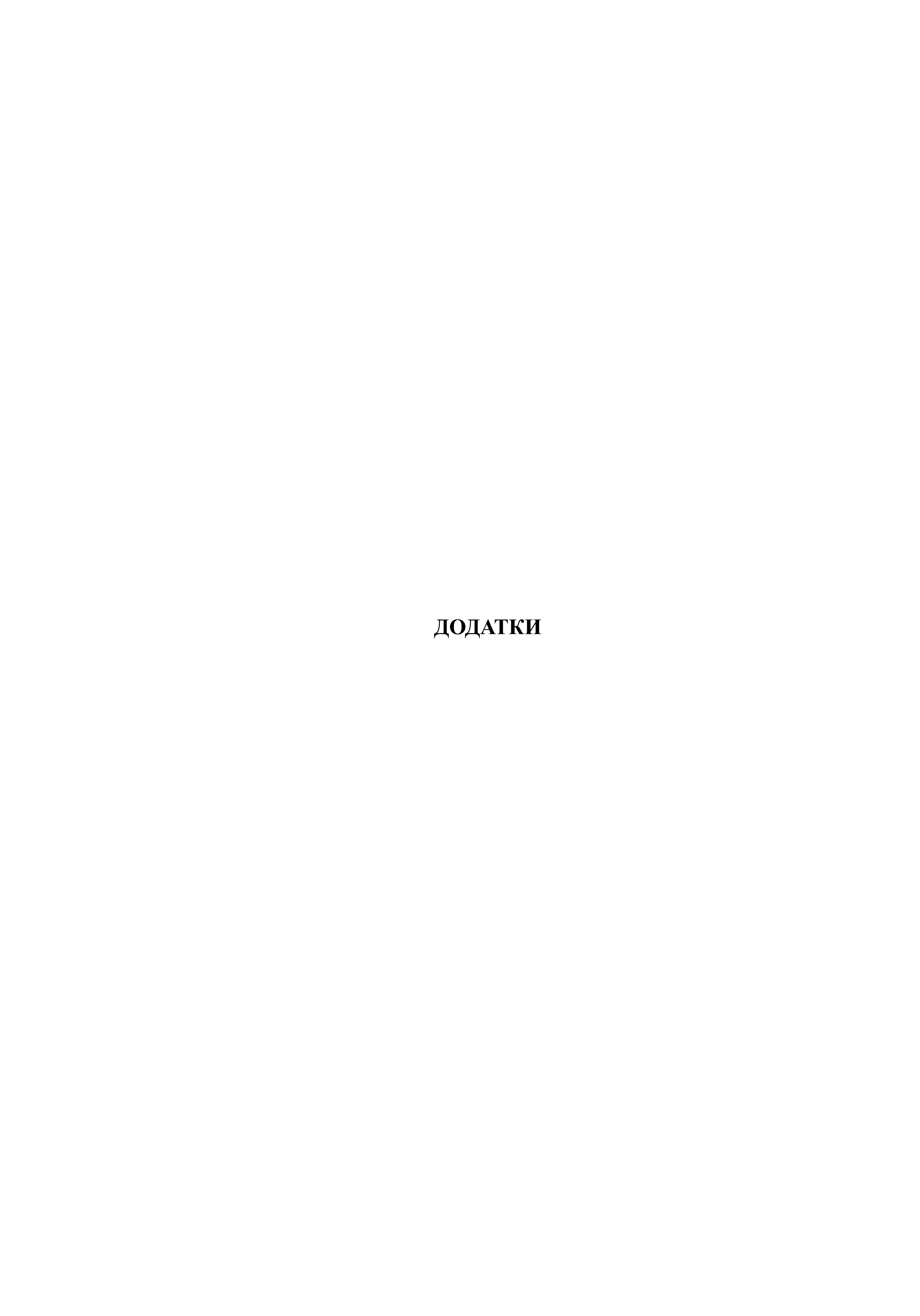
49. Zhuorong L., Huawei T., Yanhui X., Yunqi T. , Anhong W. An Error-Correcting Code-Based Robust Watermarking Scheme for Stereolithographic Files. Computer Systems Science & Engineering. 2021. Vol. 37, no. 2. — P. 247–263.

50. Wilson P. High Speed Video Application. Design Recipes for FPGAs (Second Edition). 2016. — P. 67–77.

51. Ai Q.S. , Liu Q., Zhou Z.D. , Yang L. , Xie S.Q. A new digital watermarking scheme for 3D triangular mesh models. Signal Processing. 2009. Vol. 89, no. 11. — P. 2159–2170.

52. Hubeli A., Gross M. Multiresolution Feature Extraction from Unstructured Meshes. IEEE Visualization. 2001. —8 p.

53. Козловський В. О., Лесько О. Й., Кавецький В. В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця : ВНТУ, 2021. 42 с. URL: https://epvm.vntu.edu.ua/wp-content/uploads/2021/09/Ekonom_magister_tehn_2021.pdf (дата звернення: 26.10.2023).



ДОДАТКИ

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
________ Юрій ЯРЕМЧУК
"20" ____вересня____ 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:

Підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей

08-72.МКР.010.00.092.ТЗ

Керівник магістерської кваліфікаційної роботи
д.ф. (PhD), доц. каф. МБІС Салієва О.В.

Вінниця – 2023 р.

1. Найменування та область застосування

Підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей.

Область застосування: захист авторських прав та інтелектуальної власності в галузі комп'ютерної графіки для тривимірних моделей.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №247 від 18.09.2023 р.

3. Мета та призначення розробки

3.1 Мета розробки: підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей.

3.2 Призначення: розроблений програмний засіб забезпечує надійний захист авторських прав і визначення походження тривимірних моделей, вдосконалюючи стійкість невидимого водяного знаку під час геометричних операцій перетворення.

4. Джерела розробки

4.1. Zhuorong L., Huawei T., Yanhui X., Yunqi T. , Anhong W. An Error-Correcting Code-Based Robust Watermarking Scheme for Stereolithographic Files. Computer Systems Science & Engineering. 2021. Vol. 37, no. 2. — P. 247–263.

4.2. El Hanafy Y. M. Watermarking 3D Models. The German University in Cairo. 2022. — P. 1–87.

4.3. Чжу Б., Фан С. Надійне нанесення водяних знаків на сліпе зображення з використанням різниці коефіцієнтів середньої частоти між блоками. Останні розробки та застосування водяних знаків зображень. 2023. Т. 12, № 19.

4.4. Юсеф Ш. М., Гатварі Н. Адаптивні водяні знаки для відео, що інтегрують нечітку модель сприйняття людської зорової системи на основі вейвлетів. Мультимедійні засоби та програми. 2014. Т. 73, № 3. С. 1545–1573.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Intel Core i7-8750H 2.20 ГГц і подібні до них;

– оперативна пам’ять – не менше 16 Gb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.2.

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист тривимірних моделей від несанкціонованого копіювання.

7.2 Неможливість отримання даних сторонніми особами для видобування ЦВЗ з тривимірної моделі.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Початок	Закінчення
1	Визначення напрямку магістерської кваліфікаційно роботи, формулювання теми	20.09.2023	24.09.2023
2	Аналіз предметної області обраної теми	25.09.2023	17.10.2023
3	Розробка алгоритму роботи	18.10.2023	25.10.2023
4	Написання магістерської роботи на основі розробленої теми	26.10.2023	14.11.2023
5	Апробація отриманих результатів	15.11.2023	18.11.2023
6	Розробка економічної частини	19.11.2023	23.11.2023
7	Передзахист магістерської кваліфікаційної роботи	24.11.2023	25.11.2023
8	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	25.11.2023	03.12.2023
9	Захист магістерської кваліфікаційної роботи	11.12.2023	15.12.2023

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

– ПЗ до магістерської кваліфікаційної роботи;

– програмний додаток;

– презентація;

– відзив керівника роботи;

– відзив опонента

Технічне завдання до виконання прийняв ____________ Мовчанюк М.Т.

Додаток Б. Лістинг програми

main.cpp

```
#include <Windows.h>
#include <iostream>
#include <pugixml.hpp>
#include "../main.h"
void saveDataToXML(const std::vector<std::vector<std::vector<float>>>& data) {
	pugi::xml_document doc;
	pugi::xml_node root = doc.append_child("data");
	for (const auto& outerVec : data) {
		pugi::xml_node outerNode = root.append_child("outer_vector");
		for (const auto& innerVec : outerVec) {
			pugi::xml_node innerNode = outerNode.append_child("inner_vector");
			for (const auto& value : innerVec) {
				innerNode.append_child("value").text() = std::to_string(value).c_str();
			}
		}
	}
	doc.save_file("data.xml");
}
std::vector<std::vector<std::vector<float>>> readDataFromXML() {
	std::vector<std::vector<std::vector<float>>> data;
	pugi::xml_document doc;
	if (doc.load_file("data.xml")) {
		pugi::xml_node root = doc.child("data");
		for (pugi::xml_node outerNode = root.child("outer_vector"); outerNode; outerNode = 
outerNode.next_sibling("outer_vector")) {
			std::vector<std::vector<float>> outerVec;
			for (pugi::xml_node innerNode = outerNode.child("inner_vector"); innerNode; 
innerNode = innerNode.next_sibling("inner_vector")) {
				std::vector<float> innerVec;
				for (pugi::xml_node valueNode = innerNode.child("value"); valueNode; 
valueNode = valueNode.next_sibling("value")) {
					innerVec.push_back(std::stof(valueNode.text().as_string()));
				}
				outerVec.push_back(innerVec);
			}
			data.push_back(outerVec);
		}
	}
	return data;
}
void saveWatermarkToXML(const std::vector<float>& data, const char* name) {
	pugi::xml_document doc;
	pugi::xml_node root = doc.append_child("watermark");
	for (const auto& value : data) {
		root.append_child("value").text() = std::to_string(value).c_str();
	}
	doc.save_file(name);
}
std::vector<float> readWatermarkFromXML(const char* name) {
	std::vector<float> data;
	pugi::xml_document doc;
	if (doc.load_file(name)) {
```

```
				pugi::xml_node root = doc.child("watermark");
				for (pugi::xml_node valueNode = root.child("value"); valueNode; valueNode = valueNode.next_sibling("value")) {
					data.push_back(std::stof(valueNode.text().as_string()));
				}
			}
			return data;}
		std::string requestFilenameFromUser(const std::string& prompt) {
			std::string filename;
			std::cout << prompt;
			std::cin >> filename;
			return filename;}
		int main(int argc, char* argv[]){
			int r = 3;
			SetConsoleCP(1251);
			SetConsoleOutputCP(1251);
			std::string path = requestFilenameFromUser("Введіть назву файлу моделі: ");
			requestFilenameFromUser("Введіть пареметр R: ");
			std::string modeProgram = requestFilenameFromUser("Вбудовування чи екстрація водяного знаку?(1/2): ");
			std::vector<float> watermark;
			watermark = readWatermarkFromXML(requestFilenameFromUser("Введіть назву файлу водяного знаку: ").data());
			std::vector<std::vector<float>> watermarks;
			igl::readOFF(path, V, F);
			EdgeVertexDetection EdgeVertexDetected = EdgeVertexDetection(V, F, r);
			Watermark watermarking = Watermark(V, F);
			if (modeProgram == "1") {
				saveDataToXML(watermarking.SetWatermark(EdgeVertexDetected, watermark));}
			else{
				std::vector<std::vector<std::vector<float>>> top10DCT = readDataFromXML();
				watermarks = watermarking.GetWatermark(EdgeVertexDetected, top10DCT);
			}
			igl::opengl::glfw::Viewer viewer;
			viewer.data().set_mesh(V, F);
			viewer.data().show_custom_labels = true;
			viewer.data().set_face_based(true);
			viewer.launch();
			if (modeProgram == "1") {
				std::string fullname;
				std::cout << "Збереження моделі з водяним знаком" << std::endl;
				fullname = requestFilenameFromUser("Введіть шлях збереження: ");
				fullname += requestFilenameFromUser("Введіть назву моделі: ");
				igl::writeOFF(fullname, V, F);
			}
			else{
				if (watermarking.DetectionWatermark(watermark, watermarks[0])) {
					std::string fullname;
					std::cout << "Модель містить даний водяний знак" << std::endl;
					std::cout << "Збереження зводяного знаку" << std::endl;
					fullname = requestFilenameFromUser("Введіть шлях збереження: ");
					fullname += requestFilenameFromUser("Введіть назву: ");
					saveWatermarkToXML(watermarks[0], fullname.data());
				}
				else
```

```
                    {std::cout << "Модель не містить даний водяний знак" << std::endl;}
            }
        }

        main.h
        #pragma once
        #define _USE_MATH_DEFINES
        #include <iostream>
        #include <filesystem>
        #include <vector>
        #include <unordered_set>
        #include <igl/readOFF.h>
        #include <igl/exact_geodesic.h>
        #include <igl/opengl/glfw/Viewer.h>
        #include <igl/per_vertex_normals.h>
        #include "EdgeVertexDetection.h"
        #include "build/Watermark.h"
        const int R = 3;
        Eigen::MatrixXd V;
        Eigen::MatrixXi F;
        Eigen::MatrixXd N_vertices;
        Eigen::MatrixXd Nv;
        Eigen::MatrixXi Nf;
        CoordinateGrid.cpp
        #include "CoordinateGrid.h"
        CoordinateGrid::CoordinateGrid(PlanePointNormal p) : plane(p){center = p.GetPoint();}
        CoordinateGrid::CoordinateGrid(PlanePointNormal p, Eigen::Vector3d x) : CoordinateGrid(p){
            xAxis = p.ProjectPointOntoPlane(x);
            if (yAxis.isZero()) {yAxis = p.GetPerpendicular(xAxis);}
        }
        CoordinateGrid::CoordinateGrid(PlanePointNormal p, Eigen::Vector3d x, Eigen::Vector3d y) :
    CoordinateGrid(p, x){this->yAxis = y;}
        Eigen::Vector3d CoordinateGrid::GetCoordPointOnGrid(Eigen::Vector2d coord){
            Eigen::Vector3d pointOnGrid = coord.x() * xAxis + coord.y() * yAxis;
            Eigen::Vector3d pointInSpace = center + pointOnGrid;
            return pointInSpace;
        }
        std::vector<Eigen::Vector3d> CoordinateGrid::GetCoordsPointOnGrid(std::vector<Eigen::Vector2d>
    coords){
            std::vector<Eigen::Vector3d> pointsInSpace;
            for each (Eigen::Vector2d coord in coords)
            {pointsInSpace.push_back(center + (coord.x() * xAxis + coord.y() * yAxis));}
            return pointsInSpace;
        }
        Eigen::Vector2d CoordinateGrid::ProjectPointToGrid(const Eigen::Vector3d& pointInSpace) {
            Eigen::Vector3d diff = pointInSpace - center;
            double xCoord = diff.dot(xAxis);
            double yCoord = diff.dot(yAxis);
            return Eigen::Vector2d(xCoord, yCoord);
        }
        double CoordinateGrid::DistanceBetweenPoints(const Eigen::Vector2d& point1, const
    Eigen::Vector2d& point2) {
            double distance = (point2 - point1).norm();
            return distance;
        }
        PlanePointNormal& CoordinateGrid::Plane(){return plane;}
```

```
CoordinateGrid.h
#pragma once
#include <vector>
#include "Edge.h"
#include "Plane.h"
class CoordinateGrid{
public:
	CoordinateGrid(PlanePointNormal p);
	CoordinateGrid(PlanePointNormal p, Eigen::Vector3d x);
	CoordinateGrid(PlanePointNormal p, Eigen::Vector3d x, Eigen::Vector3d y);
	Eigen::Vector3d GetCoordPointOnGrid(Eigen::Vector2d coord);
	std::vector <Eigen::Vector3d> GetCoordsPointOnGrid(std::vector<Eigen::Vector2d>
coords);
	Eigen::Vector2d CoordinateGrid::ProjectPointToGrid(const Eigen::Vector3d&
pointInSpace);
	double CoordinateGrid::DistanceBetweenPoints(const Eigen::Vector2d& point1, const
Eigen::Vector2d& point2);
	PlanePointNormal& Plane();
	Eigen::Vector3d GetCenter() { return center; }
	Eigen::Vector3d GetXAxis() { return xAxis; }
	Eigen::Vector3d GetYAxis() { return yAxis; }
private:
	PlanePointNormal plane;
	Eigen::Vector3d center;
	Eigen::Vector3d xAxis;
	Eigen::Vector3d yAxis;
};

DistanceImage.h
#pragma once
#include <vector>
#include <cmath>
#include <algorithm>
#include "Plane.h"
constexpr double PI = 3.14159265358979323846;
class DistanceImageBlock{
	struct ValueWithCoordinates {
		double& value;
		Eigen::Vector2i coordinates;
	};
public:
	DistanceImageBlock(std::vector<std::vector<double>>& pInDist,
std::vector<std::vector<double>>& pInNormDist, int startX, int startY, size_t size): sizeBlok(size){
		for (size_t y = 0; y < sizeBlok; y++){
			std::vector<Eigen::Vector2i> temp;
			std::vector<double&> tempPixelsInDist;
			std::vector<double&> tempPixelsInNormDist;
			for (size_t x = 0; x < sizeBlok; x++){
				Eigen::Vector2i coor;
				coor.x() = startX + x;
				coor.y() = startY + y;
				temp.push_back(coor);
				tempPixelsInDist.push_back(pInDist[coor.x()][coor.y()]);
			tempPixelsInNormDist.push_back(pInNormDist[coor.x()][coor.y()]);
			}
			coords.push_back(temp);
			pixelsInDist.push_back(tempPixelsInDist);
```

```
					pixelsInNormDist.push_back(tempPixelsInNormDist);
				}
				coordsInPlane = std::vector<std::vector<Eigen::Vector2d>>(size,
std::vector<Eigen::Vector2d>(size));
	}
			void SetCoordInPlaneAndIntersectionPlane(Eigen::Vector2i coord, Eigen::Vector2d
coordInPlane, PlaneThreePoint& intersectionPlane)
			{coordsInPlane[coord.x()][coord.y()] = coordInPlane;
				if (intersectionPlanes.size() == 0)
					intersectionPlanes.push_back(intersectionPlane);
				else{
					bool alreadyIs = false;
					for (size_t i = 0; i < intersectionPlanes.size(); i++){
						if (intersectionPlanes[i] == intersectionPlane) {
							alreadyIs = true;
							break;
						}
					}
					if (!alreadyIs) { intersectionPlanes.push_back(intersectionPlane);}
				}
			}
			bool IsInBlock(Eigen::Vector2i coord) {
				int minX = coords[0][0].x();
				int maxX = coords[0][sizeBlok].x();
				int minY = coords[0][0].y();
				int maxY = coords[sizeBlok][sizeBlok].y();
				return (coord.x() >= minX && coord.x() <= maxX && coord.y() >= minY &&
coord.y() <= maxY) ? true : false;
			}
			std::vector<float> EmbedValue(float value, float b, double minOrg, double maxOrg, double
maxNorm, CoordinateGrid& grid){
				DCT(pixelsInNormDist);
				std::vector<ValueWithCoordinates> top10DCT = top10DCTValues(pixelsInNormDist);
				for (size_t i = 0; i < top10DCT.size(); i++){top10DCT[i].value = top10DCT[i].value + b
* value;}
				IDCT(pixelsInNormDist);
				denormalizePixels(pixelsInNormDist, pixelsInDist, minOrg, maxOrg, maxNorm);
				double curentDistance = 0;
				int numIntersectionPlate = 0;
				for (size_t y = 0; y < pixelsInDist.size(); y++){
					for (size_t x = 0; x < pixelsInDist[y].size(); x++){
						Eigen::Vector2d coordOnGrid = coordsInPlane[x][y];
						Eigen::Vector3d pointOnGrid =
grid.GetCoordPointOnGrid(coordOnGrid);
						for (size_t j = 0; j < intersectionPlanes.size(); j++){
							Eigen::Vector3d intersectionPoint;
							if
(intersectionPlanes[j].intersectionWithLinAndIsInsideBounds(pointOnGrid, grid.Plane().GetNormalVector(),
intersectionPoint)) {
								curentDistance = (intersectionPoint -
pointOnGrid).norm();
								numIntersectionPlate = j;
								break;
							}
						}
						PointD& p = SmallestDistance(grid, coordOnGrid,
intersectionPlanes[numIntersectionPlate]);
```

```
double requiredDistance = pixelsInDist[x][y];
double difference = requiredDistance - curentDistance;
MovePointAlongNormal(p, grid.Plane().GetNormalVector(),
difference);
}
}
std::vector<float> result;
for (size_t i = 0; i < top10DCT.size();
i++){result.push_back(top10DCT[i].value);}
return result;
}
float ExtractValue(std::vector<float> origTop10DCTValues, float b){
float secretValue = 0;
int numTopVal = 10;
if(origTop10DCTValues.size() != numTopVal)
DCT(pixelsInNormDist);
std::vector<ValueWithCoordinates> top10DCT =
top10DCTValues(pixelsInNormDist);
for (size_t i = 0; i < top10DCT.size(); i++){
secretValue += (top10DCT[i].value - origTop10DCTValues[i]) / b ;
}
secretValue /= numTopVal;
}
private:
void DCT(std::vector<std::vector<double&>>& data) {
int M = data.size();
int N = data[0].size();
for (int u = 0; u < M; ++u) {
for (int v = 0; v < N; ++v) {
double sum = 0.0;
for (int i = 0; i < M; ++i) {
for (int j = 0; j < N; ++j) {
sum += data[i][j] * cos((2 * i + 1) * u * PI / (2.0
* M)) *
cos((2 * j + 1) * v * PI / (2.0 * N));
}
}
sum *= (u == 0 ? sqrt(1.0 / M) : sqrt(2.0 / M)) * (v == 0 ?
sqrt(1.0 / N) : sqrt(2.0 / N));
data[u][v] = sum;
}
}
}
void IDCT(std::vector<std::vector<double&>>& data) {
int M = data.size();
int N = data[0].size();
for (int i = 0; i < M; ++i) {
for (int j = 0; j < N; ++j) {
double sum = 0.0;
for (int u = 0; u < M; ++u) {
for (int v = 0; v < N; ++v) {
double Cu = (u == 0) ? sqrt(1.0 / M) : sqrt(2.0 /
M);
double Cv = (v == 0) ? sqrt(1.0 / N) : sqrt(2.0 /
N);
```

```
                                        sum += Cu * Cv * data[u][v] * cos((2 * i + 1) * u * PI /
(2.0 * M)) *
                                            cos((2 * j + 1) * v * PI / (2.0 * N));
                                    }
                                }
                                data[i][j] = sum;
                            }
                        }
                    }
        std::vector<ValueWithCoordinates> top10DCTValues(const std::vector<std::vector<double&>>&
dctMatrix) {
                        int rows = dctMatrix.size();
                        int cols = dctMatrix[0].size();
                        std::vector<ValueWithCoordinates> values;
                        for (int i = 0; i < rows; ++i) {
                            for (int j = 0; j < cols; ++j) {
                                ValueWithCoordinates val = { dctMatrix[i][j] };
                                val.coordinates = Eigen::Vector2i(i, j);
                                values.push_back(val);
                            }
                        }
                        std::sort(values.begin(), values.end(), compareValues);
                        std::vector<ValueWithCoordinates> top10Coordinates;
                        for (int i = 0; i < 10 && i < values.size(); ++i) {top10Coordinates.push_back(values[i]);
        }
                        return top10Coordinates;
                    }
                    bool compareValues(const ValueWithCoordinates& a, const ValueWithCoordinates& b) {
                        return a.value > b.value;}
                    void denormalizePixels(const std::vector<std::vector<double&>>& matrix,
std::vector<std::vector<double&>>& res, double minOrg, double maxOrg, double maxNorm){
                        for (size_t i = 0; i < matrix.size(); i++){
                            for (size_t j = 0; j < matrix[i].size(); j++){
                                res[i][j] = denormalizeFrom255(matrix[i][j], minOrg, maxOrg,
maxNorm);}
                        }
                    }
                    double denormalizeFrom255(double value, double minOrg, double maxOrg, double maxNorm)
{
                        return (value * (maxOrg - minOrg) / maxNorm) + minOrg;}
                    PointD& SmallestDistance(CoordinateGrid& grid, Eigen::Vector2d coordOnGrid,
PlaneThreePoint& plane){
                        Eigen::Vector2d point1 = grid.ProjectPointToGrid(Eigen::Vector3d(plane.p1.x,
plane.p1.y, plane.p1.z));
                        Eigen::Vector2d point2 = grid.ProjectPointToGrid(Eigen::Vector3d(plane.p2.x,
plane.p2.y, plane.p2.z));
                        Eigen::Vector2d point3 = grid.ProjectPointToGrid(Eigen::Vector3d(plane.p3.x,
plane.p3.y, plane.p3.z));
                        double distanceToPoint1 = grid.DistanceBetweenPoints(point1, coordOnGrid);
                        double distanceToPoint2 = grid.DistanceBetweenPoints(point2, coordOnGrid);
                        double distanceToPoint3 = grid.DistanceBetweenPoints(point3, coordOnGrid);
                        if (distanceToPoint1 <= distanceToPoint2 && distanceToPoint1 <= distanceToPoint3)
                            return plane.p1;
                        else if (distanceToPoint2 <= distanceToPoint1 && distanceToPoint2 <=
distanceToPoint3)
                            return plane.p2;
                        else
```

```
				return plane.p3;
			}
			void MovePointAlongNormal(PointD& point, const Eigen::Vector3d& normalVector,
double distance) {
				Eigen::Vector3d normalizedNormal = normalVector.normalized();
				point.x += distance * normalizedNormal.x();
				point.y += distance * normalizedNormal.y();
				point.z += distance * normalizedNormal.z();}
			std::vector<std::vector<Eigen::Vector2i>> coords;
			std::vector<std::vector<Eigen::Vector2d>> coordsInPlane;
			std::vector<std::vector<double&>> pixelsInDist;
			std::vector<std::vector<double&>> pixelsInNormDist;
			std::vector<PlaneThreePoint&> intersectionPlanes;
			size_t sizeBlok;
		};
		class DistanceImage{
		public:
			DistanceImage(size_t w, size_t h) : width(w), hight(h){
				originalDistancesPixels = std::vector<std::vector<double>>(width,
std::vector<double>(hight));
				modificateDistancesPixels = std::vector<std::vector<double>>(width,
std::vector<double>(hight));
				originalNormalizationDistancesPixels =
std::vector<std::vector<double>>(width, std::vector<double>(hight));
				modificateNormalizationDistancesPixels =
std::vector<std::vector<double>>(width, std::vector<double>(hight));
				int numColumns = width / sizeBlok;
				int numRows = hight / sizeBlok;

				for (size_t i = 0; i < numRows; i++){
					for (size_t j = 0; j < numColumns; j++){
						int startX = j * sizeBlok;
						int startY = i * sizeBlok;
						distanceImageBlocks.push_back(DistanceImageBlock(
							modificateDistancesPixels,
							modificateNormalizationDistancesPixels,
							startX, startY, sizeBlok));
					}
				}
			}
			std::vector<std::vector<float>> EmbedWatermark(const std::vector<float>&
watermark, float b, CoordinateGrid& grid)	{
				std::vector <std::vector<float>> result;
				for (size_t i = 0; i < watermark.size(); i++){
					result.push_back(distanceImageBlocks[i].EmbedValue(watermark[i], b,
minOrg, maxOrg, maxNorm, grid));
				}
				return result;
			}
			void ExtractWatermark(std::vector<std::vector<float>>& origTop10DCTValues, float
b, std::vector<float>& watermark){
				for (size_t i = 0; i < origTop10DCTValues.size(); i++){
					float secret =
distanceImageBlocks[i].ExtractValue(origTop10DCTValues[i], b);
					watermark.push_back(secret);
				}
			}
```

```
		void InitializeOriginalDistancesPixels(Eigen::Vector2d coords, double value, Eigen::Vector2d
coordsInPlane, PlaneThreePoint& intersectionPlanes){
			int indexBlock = 0;
			for (size_t i = 0; i < distanceImageBlocks.size(); i++){
				if (distanceImageBlocks[i].IsInBlock(coords)) {
					indexBlock = i;
					break;
				}
			}
		distanceImageBlocks[indexBlock].SetCoordInPlaneAndIntersectionPlane(coords,
coordsInPlane, intersectionPlanes);
			SetOriginalDistancesPixels(coords.x(), coords.y(), value);
			SetModificateDistancesPixels(coords.x(), coords.y(), value);	}
		double& OriginalDistancesPixels(int x, int y){ return originalDistancesPixels[x][y];}
		double GetOriginalDistancesPixels(int x, int y)  { return originalDistancesPixels[x][y]; }
		double SetOriginalDistancesPixels(int x, int y, double value)	{
			originalDistancesPixels[x][y] = value;
			originalNormalizationDistancesPixels[x][y] = normalizeTo255(value);
			if (modificateDistancesPixels[x][y] == 0) {
				modificateDistancesPixels[x][y] = value;
				modificateNormalizationDistancesPixels[x][y] = normalizeTo255(value);}
			return originalDistancesPixels[x][y];
		}
		double& OriginalNormalizationDistancesPixels(int x, int y) { return
originalNormalizationDistancesPixels[x][y]; }
		double GetOriginalNormalizationDistancesPixels(int x, int y) { return
originalNormalizationDistancesPixels[x][y]; }
		double SetOriginalNormalizationDistancesPixels(int x, int y, double value){
			originalNormalizationDistancesPixels[x][y] = value;
			originalDistancesPixels[x][y] = denormalizeFrom255(value);
			if (modificateNormalizationDistancesPixels[x][y] == 0) {
				modificateNormalizationDistancesPixels[x][y] = value;
				modificateDistancesPixels[x][y] = denormalizeFrom255(value);}
			return originalNormalizationDistancesPixels[x][y]; }
		double& ModificateDistancesPixels(int x, int y) { return modificateDistancesPixels[x][y]; }
		double GetModificateDistancesPixels(int x, int y) { return modificateDistancesPixels[x][y]; }
		double SetModificateDistancesPixels(int x, int y, double value)  {
			modificateDistancesPixels[x][y] = value;
			modificateNormalizationDistancesPixels[x][y] = normalizeTo255(value);
			return modificateDistancesPixels[x][y]; }
		double& ModificateNormalizationDistancesPixels(int x, int y) { return
modificateNormalizationDistancesPixels[x][y]; }
		double GetModificateNormalizationDistancesPixels(int x, int y) { return
modificateNormalizationDistancesPixels[x][y]; }
		double SetModificateNormalizationDistancesPixels(int x, int y, double value) {
			modificateNormalizationDistancesPixels[x][y] = value;
			modificateDistancesPixels[x][y] = denormalizeFrom255(value);
			return modificateNormalizationDistancesPixels[x][y]; }
		void SetMinOrg(double val) { minOrg = val; }
		void SetMaxOrg(double val) { maxOrg = val; }
	private:
		double minNorm = 0;
		double maxNorm = 255;
		double minOrg = 0;
		double maxOrg = 0;
		size_t width = 0;
		size_t hight = 0;
```

```
        size_t sizeBlok = 8;
        std::vector<std::vector<double>> originalDistancesPixels;
        std::vector<std::vector<double>> originalNormalizationDistancesPixels;
        std::vector<std::vector<double>> modificateDistancesPixels;
        // Пікселі у вигляді дистанцій
        std::vector<std::vector<double>> modificateNormalizationDistancesPixels;
        std::vector<DistanceImageBlock> distanceImageBlocks;
        double normalizeTo255(double value) {return ((value - minOrg) * maxNorm) /
(maxOrg - minOrg);}
        double denormalizeFromS255(double value) {return (value * (maxOrg - minOrg) /
maxNorm) + minOrg;}
    };
```

Edge.cpp

```
    #include "Edge.h"
    #include <cmath>
    Edge::Edge      (unsigned originNumVerA, unsigned originNumVerB, Eigen::Vector3d
originVerA, Eigen::Vector3d originVerB):        numVertexA(originNumVerA),
numVertexB(originNumVerB), vertexA(originVerA), vertexB(originVerB){
        distance = std::sqrt(std::pow(vertexB.x() - vertexA.x(), 2) +
            std::pow(vertexB.y() - vertexA.y(), 2) +
            std::pow(vertexB.z() - vertexA.z(), 2));
        vectorAB.x() = vertexB.x() - vertexA.x();
        vectorAB.y() = vertexB.y() - vertexA.y();
        vectorAB.z() = vertexB.z() - vertexA.z();
        vectorBA.x() = vertexA.x() - vertexB.x();
        vectorBA.y() = vertexA.y() - vertexB.y();
        vectorBA.z() = vertexA.z() - vertexB.z();
        middleVertex.x() = (vertexA.x() + vertexB.x()) / 2.0;
        middleVertex.y() = (vertexA.y() + vertexB.y()) / 2.0;
        middleVertex.z() = (vertexA.z() + vertexB.z()) / 2.0;
    }
    bool Edge::isThisEdge(unsigned num1, unsigned num2){return numVertexA == num1 &&
numVertexB == num2 ? true : false;}
    unsigned Edge::GetNumVertexA(){ return numVertexA; }
    unsigned Edge::GetNumVertexB(){ return numVertexB; }
    double Edge::GetDistance(){ return distance; }
    double Edge::GetWeightEdge(){ return weightEdge; }
    Eigen::Vector3d Edge::GetVertexA(){ return vertexA; }
    Eigen::Vector3d Edge::GetVertexB(){ return vertexB; }
    Eigen::Vector3d Edge::GetVectorAB(){ return vectorAB; }
    Eigen::Vector3d Edge::GetVectorBA(){ return vectorBA; }
    Eigen::Vector3d Edge::GetMiddleVertex(){ return middleVertex; }
    void Edge::SetWeightEdge(double w) { weightEdge = w; }
    bool Edge::operator==(const Edge& other) const{
        return (numVertexA == other.numVertexA && numVertexB == other.numVertexB) ||
            (numVertexA == other.numVertexB && numVertexB == other.numVertexA)
            ? true : false;
    }
```

Edge.h

```
    #pragma once
    #include <igl/exact_geodesic.h>
    class Edge{
    private:
        unsigned numVertexA;
        unsigned numVertexB;
```

```
		Eigen::Vector3d vertexA;
		Eigen::Vector3d vertexB;
		Eigen::Vector3d vectorAB;
		Eigen::Vector3d vectorBA;
		Eigen::Vector3d middleVertex;
		double distance = 0;
		double weightEdge = 0;
	public:
		Edge(unsigned originNumVetrA, unsigned originNumVetrB, Eigen::Vector3d originVetrA,
Eigen::Vector3d originVetrB);
		bool isThisEdge(unsigned num1, unsigned num2);
		unsigned GetNumVertexA();
		unsigned GetNumVertexB();
		double GetDistance();
		double GetWeightEdge();
		Eigen::Vector3d GetVertexA();
		Eigen::Vector3d GetVertexB();
		Eigen::Vector3d GetVectorAB();
		Eigen::Vector3d GetVectorBA();
		Eigen::Vector3d GetMiddleVertex();
		void SetWeightEdge(double w);
		bool operator==(const Edge& other) const;
		const double thresholdValue = 60;
	};
```

EdgeVertexDetection.cpp

```
	#include "EdgeVertexDetection.h"
	EdgeVertexDetection::EdgeVertexDetection(const Eigen::MatrixXd originV, const Eigen::MatrixXi
originF, const int originR = 3) : R(originR), V(originV), F(originF){
		Edges = new Edge * [F.rows() * 3];
		{
			int b = 0, e = 1;
			for (size_t i = 0, j = 0; i < F.rows(); i++){
				for (size_t t = 0; t < 3; t++){
					Edges[j] = new Edge(
						F(i, b),
						F(i, e),
						Eigen::Vector3d(V(F(i, b), 0), V(F(i, b), 1), V(F(i, b), 2)),
						Eigen::Vector3d(V(F(i, e), 0), V(F(i, e), 1), V(F(i, e), 2)));
					b == 2 ? b = 0 : b++;
					e == 2 ? e = 0 : e++;
					j++;
				}
			}
		}
		for (size_t i = 0; i < F.rows() * 3; i++) {EdgeStrength(i);}
	}
	double EdgeVertexDetection::EdgeStrength(int i) {return EdgeStrength(Edges[i]);}
	double EdgeVertexDetection::EdgeStrength(Edge* edge){
		std::vector<Edge> numVertexList = DetermineAroundN(edge);
		PlanePointNormal plane(edge->GetVectorAB(), edge->GetMiddleVertex());
		std::vector<Eigen::Vector3d> list(0);
		list.push_back(edge->GetMiddleVertex());
		for (size_t i = 0; i < numVertexList.size(); i++){
			Eigen::Vector3d temp{0, 0, 0};
			if (plane.DoesIntersectSegment(
				numVertexList[i].GetVertexA(),
```

```
                        numVertexList[i].GetVertexB(),
                        temp))  {list.push_back(temp);  }
            }
            TestVector3d = list;
            std::vector<Eigen::Vector2d> coordinatesOnPlane =
ConversionToPlaneCoordinates(list, edge->GetMiddleVertex());
            TestVector2d = coordinatesOnPlane;
            std::vector<Eigen::Vector2d> positiveX (0);
            std::vector<Eigen::Vector2d> negativeX (0);
            for (size_t i = 0; i < coordinatesOnPlane.size(); i++){
                  coordinatesOnPlane[i].y() *= -1;
                  if (coordinatesOnPlane[i].x() > 0) {
                        positiveX.push_back(coordinatesOnPlane[i]);    }
                  if (coordinatesOnPlane[i].x() < 0) {
                        negativeX.push_back(coordinatesOnPlane[i]);   }
            }
            RemoveCopyItem(positiveX);
            RemoveCopyItem(negativeX);
            double minCoordPositiveX = coordinatesOnPlane[coordinatesOnPlane.size() - 1].x();
            for each (auto var in coordinatesOnPlane) {
                  if (var.x() == 0)
                        continue;
                  double temp = var.x();
                  if (var.x() < 0)
                        temp = var.x() * (-1.0);
                  minCoordPositiveX >= temp ? minCoordPositiveX = temp : false;
            }
            double scale = 1;
            do{
                  scale *= 10;
                  minCoordPositiveX = minCoordPositiveX * scale;
            } while (minCoordPositiveX < 100);
            for (size_t i = 0; i < negativeX.size(); i++){
                  negativeX[i].x() *= scale;
                  negativeX[i].y() *= scale;}
            for (size_t i = 0; i < positiveX.size(); i++){
                  positiveX[i].x() *= scale;
                  positiveX[i].y() *= scale;}
            const double y = 1;
            double pr_e_derivative = LagrangePolynomialDerivative(0, positiveX);
            double pl_e_derivative = LagrangePolynomialDerivative(0, negativeX);
            Eigen::Vector2d lOneVector = Eigen::Vector2d(y / VectorLength(Eigen::Vector2d(y,
pl_e_derivative)), pl_e_derivative / VectorLength(Eigen::Vector2d(y, pl_e_derivative)));
            Eigen::Vector2d rOneVector = Eigen::Vector2d(y / VectorLength(Eigen::Vector2d(y,
pr_e_derivative)), pr_e_derivative / VectorLength(Eigen::Vector2d(y, pr_e_derivative)));
            int nMinus = negativeX.size();
            int nPlus = positiveX.size();
            Eigen::MatrixXd A_minus_x(nMinus, 2);
            Eigen::VectorXd b_minus_x(nMinus);
            Eigen::MatrixXd A_plus_x(nPlus, 2);
            Eigen::VectorXd b_plus_x(nPlus);
            for (int i = 0; i < A_minus_x.rows(); i++) {
                  A_minus_x(i, 0) = negativeX[i].x();
                  A_minus_x(i, 1) = 1.0;
                  b_minus_x(i) = negativeX[i].y();}
            for (int i = 0; i < A_plus_x.rows(); i++) {
                  A_plus_x(i, 0) = positiveX[i].x();
```

```
                A_plus_x(i, 1) = 1.0;
                b_plus_x(i) = positiveX[i].y();   }
            Eigen::Vector2d coefficients_minus_x = A_minus_x.colPivHouseholderQr().solve(b_minus_x);
            Eigen::Vector2d coefficients_plus_x = A_plus_x.colPivHouseholderQr().solve(b_plus_x);
            Eigen::Vector2d gradient_minus_x(coefficients_minus_x(0), 1.0);
            Eigen::Vector2d gradient_plus_x(coefficients_plus_x(0), 1.0);
            double angle = AngleBetweenVectors(gradient_minus_x, gradient_plus_x) * (180.0 /
3.14159265359);
            edge->SetWeightEdge(angle);
            return edge->GetWeightEdge();
        }
        std::vector<Edge> EdgeVertexDetection::DetermineAroundN(Edge* edge)
        {   std::vector<int> numVertexListN{ (int)edge->GetNumVertexA(), (int)edge-
>GetNumVertexB() };
            std::vector<int> faceListN(0);
            std::vector<int> temp(0);
            int rememmberPos = 0;
            for (size_t i = 0; i < R; i++){
                for (size_t j = rememmberPos; j < numVertexListN.size(); j++){
                    for (size_t e = 0; e < F.rows(); e++){
                        if (    F(e, 0) == numVertexListN[j] ||
                                F(e, 1) == numVertexListN[j] ||
                                F(e, 2) == numVertexListN[j]){
                                temp.push_back(F(e, 0));
                                temp.push_back(F(e, 1));
                                temp.push_back(F(e, 2));
                        }
                    }
                    rememmberPos++;
                }
                faceListN.insert(faceListN.cend(), temp.cbegin(), temp.cend());
                numVertexListN.insert(numVertexListN.cend(), temp.cbegin(), temp.cend());
                temp.clear();
                RemoveCopyItem(numVertexListN);
            }
            int size = numVertexListN.size();
            Nv.resize(size, 3);
            for (size_t i = 0; i < size; i++)    {
                Nv(i, 0) = V(numVertexListN[i], 0);
                Nv(i, 1) = V(numVertexListN[i], 1);
                Nv(i, 2) = V(numVertexListN[i], 2);
            }
            size = faceListN.size() / 3;
            Nf.resize(size, 3);
            size_t j = 0;
            for (size_t i = 0; i < size; i++){
                Nf(i, 0) = faceListN[j];
                Nf(i, 1) = faceListN[j + 1];
                Nf(i, 2) = faceListN[j + 2];
                j += 3;
            }
            std::vector<Edge> edges;
            int b = 0, e = 1;
            for (size_t i = 0, j = 0; i < Nf.rows(); i++){
                for (size_t t = 0; t < 3; t++){
                    edges.push_back(Edge(
                        Nf(i, b),
```

```
                                        Nf(i, e),
                                        Eigen::Vector3d(V(Nf(i, b), 0), V(Nf(i, b), 1), V(Nf(i, b), 2)),
                                        Eigen::Vector3d(V(Nf(i, e), 0), V(Nf(i, e), 1), V(Nf(i, e), 2))
                                ));
                                b == 2 ? b = 0 : b++;
                                e == 2 ? e = 0 : e++;
                                j++;
                        }
                }
                RemoveCopyItem(edges);
                faceListN = BringingFaceArrayToVertexArray(faceListN, numVertexListN);
                size = faceListN.size() / 3;
                Nf.resize(size, 3);
                j = 0;
                for (size_t i = 0; i < size; i++)    {
                        Nf(i, 0) = faceListN[j];
                        Nf(i, 1) = faceListN[j + 1];
                        Nf(i, 2) = faceListN[j + 2];
                        j += 3;
                }
                return edges;
        }void EdgeVertexDetection::RemoveCopyItem(std::vector<int>& v){
                std::unordered_set<int> s;
                auto end = std::remove_copy_if(v.begin(), v.end(), v.begin(),
                        [&s](int const& i) {       return !s.insert(i).second;});
                v.erase(end, v.end());
        }
        void EdgeVertexDetection::RemoveCopyItem(std::vector<Eigen::Vector3d>& v){
                std::vector<Eigen::Vector3d> uniqueVectors;
                for (const Eigen::Vector3d& vector : v) {
                        if (std::find(uniqueVectors.begin(), uniqueVectors.end(), vector) ==
uniqueVectors.end()) {
                                uniqueVectors.push_back(vector);}
                }
                v = uniqueVectors;
        }
        void EdgeVertexDetection::RemoveCopyItem(std::vector<Eigen::Vector2d>& v){
                std::vector<Eigen::Vector2d> uniqueVectors;
                for (const Eigen::Vector2d& vector : v) {
                        if (std::find(uniqueVectors.begin(), uniqueVectors.end(), vector) ==
uniqueVectors.end()) {
                                uniqueVectors.push_back(vector);
                        }
                }
                v = uniqueVectors;
        }
        void EdgeVertexDetection::RemoveCopyItem(std::vector<Edge>& v){
                std::vector<Edge> uniqueVectors;
                for (const Edge& vector : v) {
                        if (std::find(uniqueVectors.begin(), uniqueVectors.end(), vector) ==
uniqueVectors.end()) {
                                uniqueVectors.push_back(vector);}
                }
                v = uniqueVectors;
        }
        std::vector<int> EdgeVertexDetection::BringingFaceArrayToVertexArray(const
std::vector<int>& v, const std::vector<int>& vertexList){
```

```
		std::vector<int> result(v.size());
		for (size_t i = 0; i < vertexList.size(); i++){
			for (size_t j = 0; j < v.size(); j++){
				if (v[j] == vertexList[i]) {result[j] = i;}
			}
		}
		return result;
	}
	std::vector<Eigen::Vector2d> EdgeVertexDetection::ConversionToPlaneCoordinates(const
std::vector<Eigen::Vector3d> cooedinate,const Eigen::Vector3d center){
		Edge xVector(0, 0, center, cooedinate[cooedinate.size() - 1]);
		vector<Eigen::Vector2d> result(0);
		for (size_t i = 0; i < cooedinate.size(); i++){
			Edge v(0, 0, center, cooedinate[i]);
			double angle = AngleBetweenVectors(xVector.GetVectorAB(), v.GetVectorAB());
			if (!angle || std::isnan(angle)) {
				result.push_back(Eigen::Vector2d(v.GetDistance(), 0));
				continue;
			}
			if(angle >= 90) continue;
			double x = v.GetDistance() * std::cos(angle);
			double y = v.GetDistance() * std::sin(angle);
			result.push_back(Eigen::Vector2d(x, y));
		}
		return result;
	}
	double EdgeVertexDetection::DotProduct(const Eigen::Vector3d& v1, const Eigen::Vector3d& v2){
return v1.x() * v2.x() + v1.y() * v2.y() + v1.z() * v2.z(); }
	double EdgeVertexDetection::DotProduct(const Eigen::Vector2d& v1, const Eigen::Vector2d& v2){
return v1.x() * v2.x() + v1.y() * v2.y(); }
	double EdgeVertexDetection::VectorLength(const Eigen::Vector3d & v){ return std::sqrt(v.x() * v.x() +
v.y() * v.y() + v.z() * v.z()); }
	double EdgeVertexDetection::VectorLength(const Eigen::Vector2d& v){ return std::sqrt(v.x() * v.x() +
v.y() * v.y()); }
	double EdgeVertexDetection::AngleBetweenVectors(const Eigen::Vector3d & v1, const
Eigen::Vector3d & v2){
		double dotProduct = DotProduct(v1, v2);
		double length1 = VectorLength(v1);
		double length2 = VectorLength(v2);
		double cosineTheta = dotProduct / (length1 * length2);
		double angleInRadians = std::acos(cosineTheta);
		return angleInRadians;
	}
	double EdgeVertexDetection::AngleBetweenVectors(const Eigen::Vector2d& v1, const
Eigen::Vector2d& v2){
		double dotProduct = v1.dot(v2); //DotProduct(v1, v2);
		double length1 = v1.norm(); //VectorLength(v1);
		double length2 = v2.norm(); //VectorLength(v2);
		double cosineTheta = dotProduct / (length1 * length2);
		double angleInRadians = std::acos(cosineTheta);
		return angleInRadians;
	}
	void EdgeVertexDetection::MoveToZeroCoordinates(Eigen::MatrixXd v, const Eigen::Vector3d vi){
		for (size_t i = 0; i < v.rows(); i++){
			for (size_t j = 0; j < 3; j++){
				Nv(i, j) -= vi(j);
			}
```

```
            }
        }
        EdgeVertexDetection::~EdgeVertexDetection() {
            // Звільнення пам'яті
            for (int i = 0; i < F.rows(); ++i) {
                delete Edges[i];
            }
            delete[] Edges;
        }
        double EdgeVertexDetection::LagrangeBasis(double x, const std::vector<Eigen::Vector2d>& v,
int i) {
            double result = 1.0;
            for (int j = 0; j < v.size(); j++) {
                if (j != i) {
                    result *= (x - v[j].x()) / (v[i].x() - v[j].x());
                }
            }
            return result;
        }
        double EdgeVertexDetection::LagrangePolynomialDerivative(double x, const
std::vector<Eigen::Vector2d>& v) {
            double result = 0.0;
            for (int i = 0; i < v.size(); i++) {
                double basis = LagrangeBasis(x, v, i);
                double sum = 0.0;
                for (int j = 0; j < v.size(); j++) {
                    if (j != i) {
                        sum += 1.0 / (v[i].x() - v[j].x());
                    }
                }
                result += basis * sum * v[i].y();
            }
            return result;
        }
        Eigen::Vector2d EdgeVertexDetection::NormalizeVector(Eigen::Vector2d vector){
            double norm = 0.0;
            for (double value : vector) {
                norm += value * value;
            }
            if (norm > 0.0) {
                norm = sqrt(norm);
                for (double& value : vector) {
                    value /= norm;
                }
            }
            return vector;
        }
        Plane.h
        #pragma once
        #include <igl/exact_geodesic.h>
        #include <iostream>
        using namespace std;
        #include "Edge.h"
        #include "VertexForEmbedding.h"
        class PlanePointNormal {
        public:
          PlanePointNormal(Eigen::Vector3d originNormalvector, Eigen::Vector3d originPoint);
```

```
            void PrintEquation();
            bool DoesIntersectSegment(Eigen::Vector3d p1, Eigen::Vector3d p2, Eigen::Vector3d&
intersectionPoint);
            bool isPointOnSegment(Eigen::Vector3d A, Eigen::Vector3d B, Eigen::Vector3d P);
            Eigen::Vector3d PlanePointNormal::GetPerpendicular(Eigen::Vector3d externalPoint);
            Eigen::Vector3d PlanePointNormal::ProjectPointOntoPlane(Eigen::Vector3d& pointToProject);
            Eigen::Vector3d GetPoint() { return point; }
            Eigen::Vector3d GetNormalVector() { return normalVector; }
        private:
            Eigen::Vector3d normalVector;
            Eigen::Vector3d point;
        };
        class PlaneThreePoint {
        private:
            double determinant(const PointD& a, const PointD& b, const PointD& c) {
                return a.x * (b.y * c.z - c.y * b.z) - a.y * (b.x * c.z - c.x * b.z) + a.z * (b.x * c.y - c.x * b.y);
            }
            double determinant(const PointD& a, const PointD& b, const Eigen::Vector3d& c) {
                return a.x * (b.y * c.z() - c.y() * b.z) - a.y * (b.x * c.z() - c.x() * b.z) + a.z * (b.x * c.y() - c.x() *
b.y);
            }
        public:
            PlaneThreePoint(const PointD& point1, const PointD& point2, const PointD& point3) : p1(point1),
p2(point2), p3(point3) { }
            PointD p1, p2, p3;
            Eigen::Vector3d intersectionWithLine(const Eigen::Vector3d& linePoint, const Eigen::Vector3d&
lineDirection) {
                Eigen::Vector3d planeNormal;
                planeNormal.x() = determinant(p2, p3, lineDirection);
                planeNormal.y() = determinant(p3, p1, lineDirection);
                planeNormal.z() = determinant(p1, p2, lineDirection);
                double d = -determinant(p1, p2, p3);
                double t = -(determinant(p1, p2, linePoint) + d) / (planeNormal.x() * lineDirection.x() +
planeNormal.y() * lineDirection.y() + planeNormal.z() * lineDirection.z());
                Eigen::Vector3d intersectionPoint;
                intersectionPoint.x() = linePoint.x() + t * lineDirection.x();
                intersectionPoint.y() = linePoint.y() + t * lineDirection.y();
                intersectionPoint.z() = linePoint.z() + t * lineDirection.z();
                return intersectionPoint;
            }
            bool isInsideBounds(const Eigen::Vector3d& point) {
                double determinant1 = determinant(p1, p2, point);
                double determinant2 = determinant(p2, p3, point);
                double determinant3 = determinant(p3, p1, point);
                bool hasSameSign = (determinant1 >= 0 && determinant2 >= 0 && determinant3 >= 0) ||
(determinant1 <= 0 && determinant2 <= 0 && determinant3 <= 0);
                return hasSameSign;
        }
            bool intersectionWithLinAndIsInsideBounds(const Eigen::Vector3d& linePoint, const
Eigen::Vector3d& lineDirection, Eigen::Vector3d& result) {
                Eigen::Vector3d r = intersectionWithLine(linePoint, lineDirection);
                if (!isInsideBounds(r)) return false;
                result = r;
                return true;
            }
            bool operator==(const PlaneThreePoint& other)
            { return (  this->p1.x == other.p1.x && this->p1.y == other.p1.y && this->p1.z == other.p1.z &&
```

```
            this->p2.x == other.p2.x && this->p2.y == other.p2.y && this->p2.z == other.p2.z &&
            this->p3.x == other.p3.x && this->p3.y == other.p3.y && this->p3.z == other.p3.z
            ) ? true : false;
    }
    bool operator!=(const PlaneThreePoint& other){ return (*this == other) ? false : true;}
};
```

Plane.cpp

```
#include "Plane.h"
PlanePointNormal::PlanePointNormal(Eigen::Vector3d originNormalVector, Eigen::Vector3d
originPoint){
    normalVector = originNormalVector;
    point = originPoint;
}
void PlanePointNormal::PrintEquation(){
    cout << "Рівняння площини: ";
    cout << normalVector.x() << "*(X - " << point.x() << ") + "
        << normalVector.y() << "*(Y - " << point.y() << ") + "
        << normalVector.z() << "*(Z - " << point.z() << ") = 0" << endl;
}
bool PlanePointNormal::DoesIntersectSegment(Eigen::Vector3d p1, Eigen::Vector3d p2,
Eigen::Vector3d& intersectionPoint){
    double t = -((normalVector.x() * (p1.x() - point.x()) + normalVector.y() * (p1.y() - point.y())
+ normalVector.z() * (p1.z() - point.z()))
        / (normalVector.x() * (p2.x() - p1.x()) + normalVector.y() * (p2.y() - p1.y()) +
normalVector.z() * (p2.z() - p1.z())));
    if (t >= 0 && t <= 1) {
        intersectionPoint.x() = p1.x() + t * (p2.x() - p1.x());
        intersectionPoint.y() = p1.y() + t * (p2.y() - p1.y());
        intersectionPoint.z() = p1.z() + t * (p2.z() - p1.z());
        return true;
    }
    return false;
}
bool PlanePointNormal::isPointOnSegment(Eigen::Vector3d A, Eigen::Vector3d B,
Eigen::Vector3d P){
    double t = (P.x() - A.x()) / (B.x() - A.x());
    if (t >= 0 && t <= 1) {
        return true;
    }
    else {
        return false;
    }
}
Eigen::Vector3d PlanePointNormal::GetPerpendicular(Eigen::Vector3d externalPoint) {
    Eigen::Vector3d vectorToExternalPoint = externalPoint - point;
    Eigen::Vector3d projection = vectorToExternalPoint.dot(normalVector) /
normalVector.squaredNorm() * normalVector;
    Eigen::Vector3d perpendicularDirection = vectorToExternalPoint - projection;
    return perpendicularDirection;
}
Eigen::Vector3d PlanePointNormal::ProjectPointOntoPlane(Eigen::Vector3d& pointToProject)
{
    Eigen::Vector3d vectorFromPlaneToPoint = pointToProject - point;
    Eigen::Vector3d projection = vectorFromPlaneToPoint.dot(normalVector) /
normalVector.squaredNorm() * normalVector;
    Eigen::Vector3d projectedPoint = pointToProject - projection;
```

```
    return projectedPoint;
}
VertexForEmbedding.h
#pragma once
#include <vector>
#include <igl/exact_geodesic.h>
struct Point {int x, y, z;};
struct PointD {double& x, y, z;};
class VertexWatermark{
public:
        VertexWatermark(Eigen::MatrixXd& V, Eigen::MatrixXi& F) : V(V), F(F) {}
        VertexWatermark operator=(VertexWatermark& othre){return othre;}
        Eigen::MatrixXd& V;
        Eigen::MatrixXi& F;
        unsigned numVertex;
        Eigen::Vector3d vertex;
        Eigen::Vector3d normal;
        double weightEdge = 0;
        std::vector<int> vertexDistrict;
        std::vector<Point> faceList;
        bool DistrictIsIncludedInAnother(VertexWatermark another){
                bool result = false;
                for each (int av in another.vertexDistrict){
                        for each (int v in vertexDistrict) {
                                if (av == v) {
                                        result = true;
                                        break;
                                }
                        }
                        if (result) break;
                }
                return result;
        }
        static void SetN(int i) { i > 0 ? n = i : n = 0; }
        std::vector<std::vector<float>> EmbedWatermark(const std::vector<float>& watermark, float
b);
        std::vector<float> ExtractWatermarc(float b, std::vector<std::vector<float>>&
origTop10DCTValues);
private:
        void CreateImage();
        static int n;
        void CreatePlane();
};
int VertexWatermark::n = 80;

VertexForEmbedding.cpp
#include "VertexForEmbedding.h"
#include "Plane.h"
#include "CoordinateGrid.h"
#include "DistanceImage.h"
#include <igl/exact_geodesic.h>
std::vector<std::vector<float>> VertexWatermark::EmbedWatermark(const std::vector<float>&
watermark, float b){
    PlanePointNormal plane = PlanePointNormal(vertex, normal);
    Eigen::VectorXi VS, FS, VT, FT;
    VS.resize(1);
    VS << numVertex;
```

```
            VT.resize(vertexDistrict.size());
            for (size_t i = 0; i < vertexDistrict.size(); i++) { VT(i) = vertexDistrict[i];}
            Eigen::VectorXd d;
            igl::exact_geodesic(V, F, VS, FS, VT, FT, d);
            int maxDistancePointNum = 0;
            int maxIndex = 0;
            double maxValue = d(maxIndex);
            for (int i = 1; i < d.size(); ++i) {
               if (d(i) > maxValue) {   maxValue = d(i);  maxIndex = i;   }}
            maxDistancePointNum = VT(maxIndex);
            CoordinateGrid grid = CoordinateGrid(plane, Eigen::Vector3d(V(maxDistancePointNum, 0),
V(maxDistancePointNum, 1), V(maxDistancePointNum, 2)));
            Eigen::Vector3d vectorBetweenPoints = grid.GetXAxis() - grid.GetCenter();
            double size = (vectorBetweenPoints.norm()) * 2;
            double step = size / n;
            vector<PlaneThreePoint> facePlane;
            for each (Point point in faceList){
               PointD p1 = PointD{V(point.x, 0), V(point.x, 1), V(point.x, 2)};
               PointD p2 = PointD{V(point.y, 0), V(point.y, 1), V(point.y, 2)};
               PointD p3 = PointD{V(point.z, 0), V(point.z, 1), V(point.z, 2)};
               facePlane.push_back(PlaneThreePoint(p1, p2, p3));}
            DistanceImage distanceImage = DistanceImage(n, n);
            double minValueDist = 0;
            double maxValueDist = 0;
            double startX = -(size / 2);
            double startY = -(size / 2);
            for (size_t i = 0; i < n; i++){
               for (size_t j = 0; j < n; j++) {
                  Eigen::Vector2d coord;
                  coord.x() = i;
                  coord.y() = j;
                  Eigen::Vector2d coordOnGrid;
                  coordOnGrid.x() = startX;
                  coordOnGrid.y() = startY;
                  Eigen::Vector3d pointOnGrid = grid.GetCoordPointOnGrid(coordOnGrid);
                  double distanse;
                  for each (PlaneThreePoint p in facePlane) {
                     Eigen::Vector3d intersectionPoint;
                     if (p.intersectionWithLinAndIsInsideBounds(pointOnGrid,
grid.Plane().GetNormalVector(), intersectionPoint)) {
                        distanse = (intersectionPoint - pointOnGrid).norm();
                        distanceImage.InitializeOriginalDistancesPixels(coord, distanse, coordOnGrid, p);
                        break;
                     }
                  }
                  startY += step;
               }
               startX += step;
            }
            return distanceImage.EmbedWatermark(watermark, b, grid);
         }
         std::vector<float> VertexWatermark::ExtractWatermarc( float b,
std::vector<std::vector<float>>& origTop10DCTValues){
            std::vector<float> watermark;
            PlanePointNormal plane = PlanePointNormal(vertex, normal);
            Eigen::VectorXi VS, FS, VT, FT;
            VS.resize(1);
```

```
        VS << numVertex;
        VT.resize(vertexDistrict.size());
        for (size_t i = 0; i < vertexDistrict.size(); i++){ VT(i) = vertexDistrict[i]; }
        Eigen::VectorXd d;
        igl::exact_geodesic(V, F, VS, FS, VT, FT, d);
        int maxDistancePointNum = 0;
        int maxIndex = 0;
        double maxValue = d(maxIndex);
        for (int i = 1; i < d.size(); ++i) {
          if (d(i) > maxValue) {   maxValue = d(i);  maxIndex = i;} }
        maxDistancePointNum = VT(maxIndex);
        CoordinateGrid grid = CoordinateGrid(plane, Eigen::Vector3d(V(maxDistancePointNum, 0),
V(maxDistancePointNum, 1), V(maxDistancePointNum, 2)));
        Eigen::Vector3d vectorBetweenPoints = grid.GetXAxis() - grid.GetCenter();
        double size = (vectorBetweenPoints.norm()) * 2;
        double step = size / n;
        vector<PlaneThreePoint> facePlane;
        for each (Point point in faceList) {
          PointD p1 = PointD{ V(point.x, 0), V(point.x, 1), V(point.x, 2) };
          PointD p2 = PointD{ V(point.y, 0), V(point.y, 1), V(point.y, 2) };
          PointD p3 = PointD{ V(point.z, 0), V(point.z, 1), V(point.z, 2) };
          facePlane.push_back(PlaneThreePoint(p1, p2, p3));}
        DistanceImage distanceImage = DistanceImage(n, n);
        double minValueDist = 0;
        double maxValueDist = 0;
        double startX = -(size / 2);
        double startY = -(size / 2);
        for (size_t i = 0; i < n; i++){
          for (size_t j = 0; j < n; j++) {
            Eigen::Vector2d coord;
            coord.x() = i;
            coord.y() = j;
            Eigen::Vector2d coordOnGrid;
            coordOnGrid.x() = startX;
            coordOnGrid.y() = startY;
            Eigen::Vector3d pointOnGrid = grid.GetCoordPointOnGrid(coordOnGrid);
            double distanse;
            for each (PlaneThreePoint p in facePlane)  {
              Eigen::Vector3d intersectionPoint;
              if (p.intersectionWithLinAndIsInsideBounds(pointOnGrid, grid.Plane().GetNormalVector(),
intersectionPoint)) {
                distanse = (intersectionPoint - pointOnGrid).norm();
                distanceImage.InitializeOriginalDistancesPixels(coord, distanse, coordOnGrid, p);
                break;
              }
            }
            startY += step; }
          startX += step; }
        distanceImage.ExtractWatermark(origTop10DCTValues, b, watermark);
        return watermark;}

      Watermark.h
      #pragma once
      #include "../EdgeVertexDetection.h"
      #include "VertexForEmbedding.h"
      class Watermark{
      public:
```

```
		Watermark(Eigen::MatrixXd& V, Eigen::MatrixXi& F) : V(V), F(F) {}
		vector<vector<vector<float>>> SetWatermark(EdgeVertexDetection
EdgeVertexDetected, const vector<float>& watermark, const int a = 5, int n = 80, float b = 0.1);
		vector<vector<float>>& GetWatermark(EdgeVertexDetection EdgeVertexDetected,
vector<vector<vector<float>>>& origTop10DCTValues, const int a = 5, int n = 80, float b = 0.1);
		bool DetectionWatermark(const vector<float>&, const vector<float>&, float t = 0.5);
	private:
		Eigen::MatrixXd& V;
		Eigen::MatrixXi& F;
		vector<VertexWatermark>& GetVoronoiPatch(EdgeVertexDetection&
EdgeVertexDetected, Eigen::MatrixXd& V, Eigen::MatrixXi& F, int R);
		void DCT();
		void IDCT();
		void SetCharWatermark(VertexWatermark, char);
		void GetCharWatermark();};

	Watermark.cpp
	#include "Watermark.h"
	void RemoveCopyItem(std::vector<int>& v){
		std::unordered_set<int> s;
		auto end = std::remove_copy_if(v.begin(), v.end(), v.begin(),
			[&s](int const& i) {	return !s.insert(i).second;});
		v.erase(end, v.end());}
	void DetermineAroundOne(Eigen::MatrixXi& F, Edge* e, const int R, std::vector<int>&
numVertexListN,  std::vector<Point>& faceList){
		numVertexListN.push_back((int)e->GetNumVertexA());
		std::vector<int> temp(0);
		int rememmberPos = 0;
		for (size_t i = 0; i < R; i++){
			for (size_t j = rememmberPos; j < numVertexListN.size(); j++){
				for (size_t e = 0; e < F.rows(); e++)	{
					if (F(e, 0) == numVertexListN[j] ||
						F(e, 1) == numVertexListN[j] ||
						F(e, 2) == numVertexListN[j]
						){
						temp.push_back(F(e, 0));
						temp.push_back(F(e, 1));
						temp.push_back(F(e, 2));
						Point p;
						p.x = F(e, 0);
						p.y = F(e, 1);
						p.z = F(e, 2);
						faceList.push_back(p);
					}
				}
				rememmberPos++;
			}
			numVertexListN.insert(numVertexListN.cend(), temp.cbegin(), temp.cend());
			temp.clear();
			RemoveCopyItem(numVertexListN);
		}}
	void BubleSort(vector<VertexWatermark>& arr)
	{	for (int i = 0; i < arr.size() - 1; i++)
			for (int j = arr.size() - 1; j > i; j--)
				if (arr[j].weightEdge < arr[j - 1].weightEdge)
					std::swap(arr[j], arr[j - 1]);}
```

```
	vector<VertexWatermark>& Watermark::GetVoronoiPatch(EdgeVertexDetection& EdgeVertexDetected, Eigen::MatrixXd& V, Eigen::MatrixXi& F, int R){
		vector<VertexWatermark> temp = {};
		vector<VertexWatermark> result = {};
		Eigen::MatrixXd N_vertices;
		igl::per_vertex_normals(V, F, N_vertices);
		for (size_t i = 0; i < F.rows() * 3; i++)
		{	Edge* e = EdgeVertexDetected.Edges[i];
			if (e->GetWeightEdge() < e->thresholdValue) continue;
			VertexWatermark r = VertexWatermark(V, F);
			r.numVertex = e->GetNumVertexA();
			DetermineAroundOne(F, e, R, r.vertexDistrict, r.faceList);
			r.normal.x() = N_vertices(e->GetNumVertexA(), 0);
			r.normal.y() = N_vertices(e->GetNumVertexA(), 1);
			r.normal.z() = N_vertices(e->GetNumVertexA(), 2);
			r.vertex = e->GetVertexA();
			r.weightEdge = e->GetWeightEdge();
			temp.push_back(r);}
		BubleSort(temp);
		result.push_back(temp[0]);
		for (size_t i = 0; i < temp.size() - 1; i++){
			for (size_t j = 0; j < result.size() - 1; j++)
			{				if (!temp[i].DistrictIsIncludedInAnother(result[j]))
					result.push_back(temp[i]);			}		}
		return result;
	}
	vector<vector<vector<float>>> Watermark::SetWatermark(EdgeVertexDetection EdgeVertexDetected, const vector<float>& watermark, const int a = 5, int n = 80, float b = 0.1){
		vector<vector<vector<float>>> result;
		VertexWatermark::SetN(n);
		vector<VertexWatermark>& vertex = GetVoronoiPatch(EdgeVertexDetected, V, F, a);
		for (size_t i = 0; i < vertex.size() - 1; i++){result.push_back(vertex[i].EmbedWatermark(watermark, b));	}
		return result;
	}
	vector<vector<float>>& Watermark::GetWatermark(EdgeVertexDetection EdgeVertexDetected, vector<vector<vector<float>>>& origTop10DCTValues, const int a = 5, int n = 80, float b = 0.1){
		VertexWatermark::SetN(n);
		vector<VertexWatermark>& vertex = GetVoronoiPatch(EdgeVertexDetected, V, F, a);
		vector<vector<float>> result = vector<vector<float>>(vertex.size());
		for (size_t i = 0; i < vertex.size() - 1; i++){
			vector<float> wat = vertex[i].ExtractWatermarc(b, origTop10DCTValues[i]);
			result[i] = wat;  }
		return result;
	}
	bool Watermark::DetectionWatermark(const vector<float>& origWatermark, const vector<float>& otherWatermsrk, float t = 0.5) {
		int L = 0;
		if (origWatermark.size() < otherWatermsrk.size())
			L = origWatermark.size();
		else L = otherWatermsrk.size();
		float result;
		float averageOrigWatermark;
		float averageOtherWatermsrk;
		for (size_t i = 0; i < L; i++)
		{averageOrigWatermark += origWatermark[i];}
		averageOrigWatermark /= origWatermark.size();
```

```
for (size_t i = 0; i < L; i++)
{averageOtherWatermsrk += otherWatermsrk[i];}
averageOtherWatermsrk /= otherWatermsrk.size();
float val1;
float val2;
float val3;
for (size_t i = 0; i < L; i++){
        val1 += (origWatermark[i] - averageOrigWatermark) * (otherWatermsrk[i] - averageOtherWatermsrk);
        val2 += std::powf(origWatermark[i] - averageOrigWatermark, 2.0f);
        val3 += std::powf(otherWatermsrk[i] - averageOtherWatermsrk, 2.0f);}
val2 = std::sqrt(val2);
val3 = std::sqrt(val3);
result = val1 / (val2 * val3);
return result > t ? true : false;}
```

Додаток В. Ілюстративний матеріал

ПІДВИЩЕННЯ СТІЙКОСТІ НЕВИДИМОГО ВОДЯНОГО ЗНАКУ ВІД ГЕОМЕТРИЧНИХ ОПЕРАЦІЙ ПЕРЕТВОРЕННЯ ШЛЯХОМ ВДОСКОНАЛЕННЯ МЕТОДУ ВБУДОВУВАННЯ НА ОСНОВІ 3D-РОЗПІЗНАВАННЯ РЕБЕРНИХ ВЕРШИН ДЛЯ ТРИВИМІРНИХ МОДЕЛЕЙ

Виконала: ст. групи 2КІТС-22м Мовчанюк М.Т.

Керівник: д.ф. (PhD), доц. каф. МБІС Салієва О.В.

Актуальність

Стрімкий ріст популярності використання тривимірних моделей зумовлений швидким технологічним прогресом, однак це також викликає проблеми, такі як порушення авторських прав. Цифрові водяні знаки розглядаються як ефективний захист від піратства. Саме тому вдосконалення алгоритмів вбудовування ЦВЗ від різних типів атак є актуальним для забезпечення ефективного захисту тривимірних моделей.

Мета

Метою роботи є підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей та програмна реалізація удосконаленого методу.

Практична цінність

Розроблено програмний додаток, що реалізує вдосконалений метод вбудовування ЦВЗ у тривимірну модель для збереження цілісності та авторських прав.

Апробація

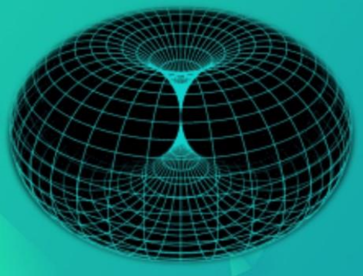
Тези доповідей у даній галузі представленні на міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи».

Алгоритм вбудовування водяних знаків – це процес вбудовування даних в оригінальний контент для підтвердження інформації про право власності або авторські права. Загалом, методи нанесення водяних знаків класифікуються на основі різної області вбудовування водяного знаку на два класи: просторово-трансформаційні та частотні.

Тривимірна модель, або 3D-модель – це математично згенероване або комп'ютерне представлення тривимірного об'єкта, яке описує його форму, розмір та структуру у тривимірному просторі.
Полігональна модель – це тип тривимірних моделей, який використовує полігони (геометричні фігури з трьох або більше вершин), з'єднані ребрами, для утворення поверхні об'єкта.

Ризики для невидимих ЦВЗ, вбудованих у тривимірні моделі:
- **виявлення (несанкціонованим користувачем);**
- **спотворення;**
- **повне видалення.**

Основні підходи до вбудовування водяних знаків у полігональні моделі включають:
- Модифікація геометричних атрибутів;
- Методи текстурного вбудовування;
- Вбудовування в топологічні атрибути.

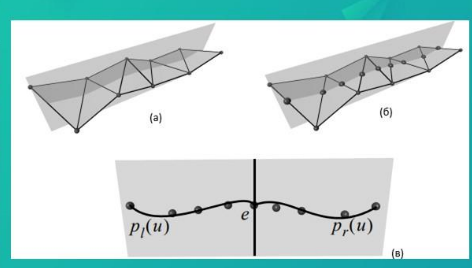


Оригінальний алгоритм пошуку вершин для вбудовування ЦВЗ є досить примітивним та сильно залежить від значення α, що є масштабом округу навколо вершини. До того ж чим більше це значення, тим менше опорних вершин буде знайдено, що є досить сильним недоліком даного алгоритму.

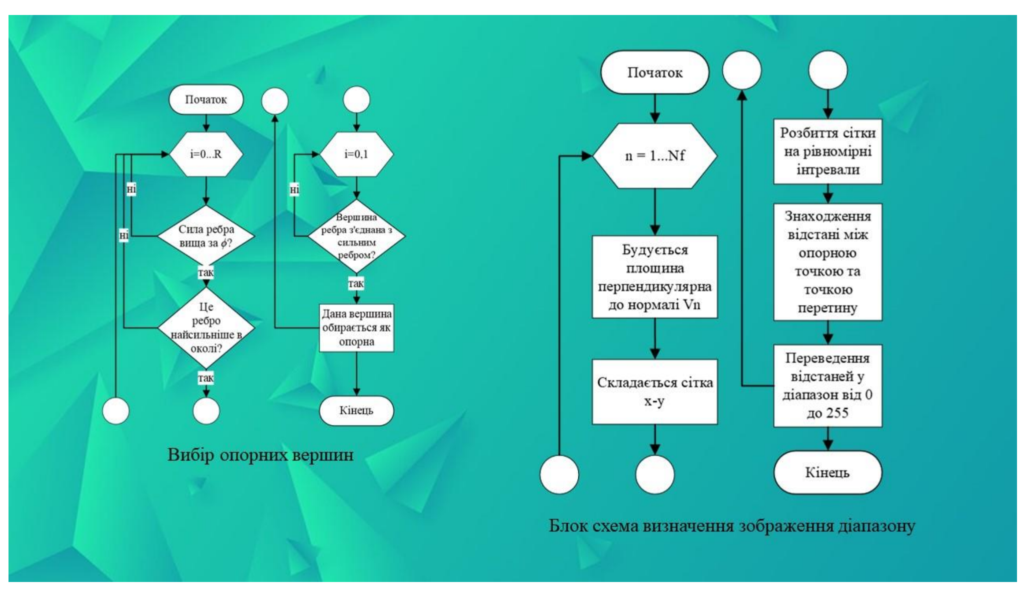
Вдосконалений алгоритм

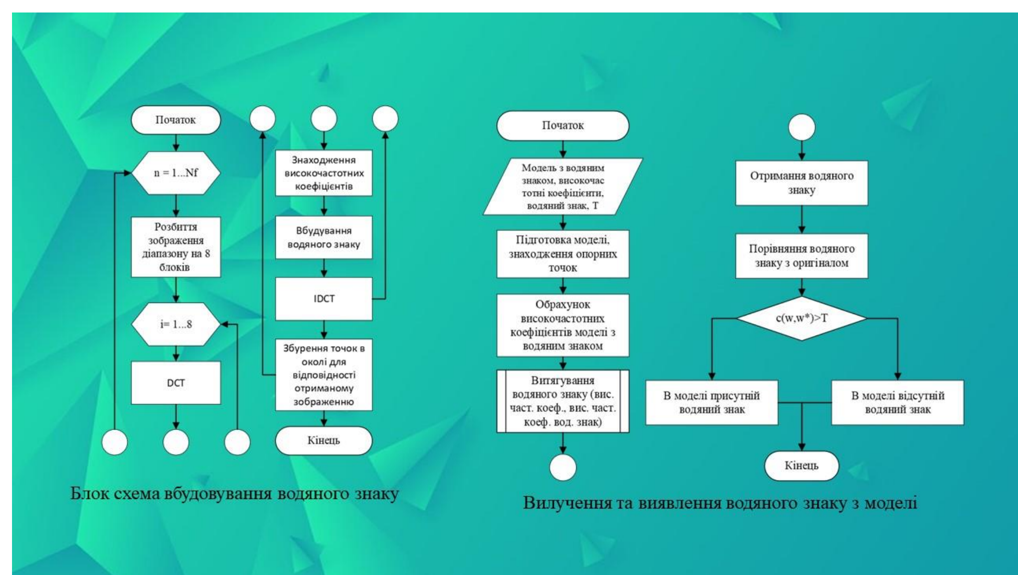
Для усунення такого суттєвого недоліку впроваджено алгоритм 3D-розпізнавання реберних вершин для пошуку опорних точок. Його застосування спрямоване на пошук опорних точок, які виявилися більш стійкими для вбудовування водяного знаку.

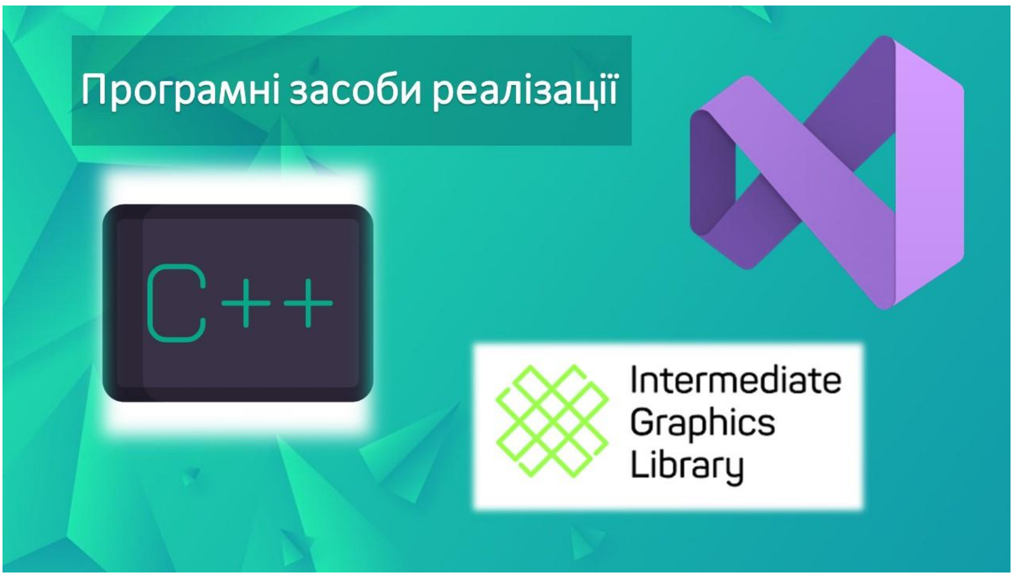
Використання алгоритму 3D-розпізнавання реберних вершин у вдосконаленому методі забезпечує більшу стійкість до геометричних атак, що робить його надійнішим порівняно з початковим методом пошуку опорних точок для вбудовування водяного знаку. Також, він має більше параметрів для пошуку опорних точок.

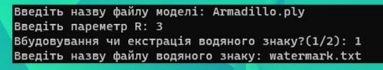


$$w(e_i) = \cos\left(\frac{(1, p_l'(e_i))}{\|(1, p_l'(e_i))\|} \cdot \frac{(1, p_r'(e_i))}{\|(1, p_r'(e_i))\|}\right)^{-1}$$


Початок
i=0...R
ні
ні
Сила ребра вища за ϕ?
так
Це ребро найсильніше в околі?
так
i=0,1
ні
Вершина ребра з'єднана з сильним ребром?
так
Дана вершина обирається як опорна
Кінець
Вибір опорних вершин
Початок
n = 1...Nf
Будується площина перпендикулярна до нормалі Vn
Складається сітка x-y
Розбиття сітки на рівномірні інтревали
Знаходження відстані між опорною точкою та точкою перетину
Переведення відстаней у діапазон від 0 до 255
Кінець
Блок схема визначення зображення діапазону


Початок
n = 1...Nf
Розбиття зображення діапазону на 8 блоків
i= 1...8
DCT
Знаходження високочастотних коефіцієнтів
Вбудування водяного знаку
IDCT
Збурення точок в околі для відповідності отриманому зображенню
Кінець
Блок схема вбудовування водяного знаку
Початок
Модель з водяним знаком, високочастотні коефіцієнти, водяний знак, T
Підготовка моделі, знаходження опорних точок
Обрахунок високочастотних коефіцієнтів моделі з водяним знаком
Витягування водяного знаку (вис. част. коеф., вис. част. коеф. вод. знак)
Отримання водяного знаку
Порівняння водяного знаку з оригіналом
c(w,w*)>T
В моделі присутній водяний знак
В моделі відсутній водяний знак
Кінець
Вилучення та виявлення водяного знаку з моделі


Програмні засоби реалізації
C++
Intermediate Graphics Library



Запуск програми в режимі вбудовування водяного знаку

```
Збереження моделі з водяним знаком
Введіть шлях збереження: C:\
Введіть назву моделі: ArmadilloWatermark.ply
```

Процес збереження тривимірної моделі в яку вбудовано

```
Модель містить даний водяний знак
Збереження зводяного знаку
Введіть шлях збереження: C:\
Введіть назву:  exWatermark.txt
```

Процедура збереження та перевірки водяного знаку тривимірної моделі

Вікно з тривимірною моделлю в яку вбудовано водяний знак

Тестування вдосконаленого алгоритму

Параметри моделей для тестування та алгоритмів

Модель	Вершини	Полігони	Ділянки	$N_x \times N_y$	L	α	β	PSNR (дБ) ориг.	PSNR (дБ) вдоск.
Заєць	362272	725000	20	80×80	20	5	0,1	88.6837	88.3561
Корова	3406	6952	15	36×36	30	3	0,05	87.8950	88.1424
Кіт	418	805	8	20×20	15	2	0,01	86.5972	86.9217

Результати тестування стійкості проти атаки обрізання

Метод	Відсоток моделі	Заєць $c(w,w^*)$	Відсоток моделі	Корова $c(w,w^*)$	Відсоток моделі	Кіт $c(w,w^*)$
Початковий	20%	0.9118	20%	0.8996	5%	0.9411
Вдосконалений		0.9432		0.9253		0.9702
Початковий	50%	0.7021	50%	0.6812	25%	0.7521
Вдосконалений		0.7735		0.7236		0.8062

Результати тестування стійкості проти атаки спрощення

Метод	Відсоток моделі	Заєць $c(w,w^*)$	Відсоток моделі	Корова $c(w,w^*)$	Відсоток моделі	Кіт $c(w,w^*)$
Початковий	10%	0.9732	10%	0.9517	5%	0.9441
Вдосконалений		0.9924		0.9843		0.9524
Початковий	30%	0.8657	30%	0.8433	10%	0.8827
Вдосконалений		0.9135		0.8712		0.9021
Початковий	50%	0.7595	45%	0.6727	20%	0.7993
Вдосконалений		0.7957		0.7004		0.8163

Результати тестування стійкості проти атаки додавання шуму

Метод	Відсоток шуму	Заєць $c(w,w^*)$	Відсоток шуму	Корова $c(w,w^*)$	Відсоток шуму	Кіт $c(w,w^*)$
Початковий	0.15%	0.9833	0.15%	0.9704	0.05%	0.9538
Вдосконалений		0.9892		0.9794		0.9642
Початковий	0.5%	0.9389	0.45%	0.9121	0.1%	0.8689
Вдосконалений		0.9535		0.9203		0.8874

Результати тестування стійкості проти комбінації атаки спрощення з обрізанням

Метод	Відсоток спрощення та обрізання	Заєць $c(w,w^*)$	Відсоток спрощення та обрізання	Корова $c(w,w^*)$	Відсоток спрощення та обрізання	Кіт $c(w,w^*)$
Початковий	80% + 30%	0.6701	80% + 30%	0.6259	95% + 10%	0.7301
Вдосконалений		0.7024		0.6335		0.7394

ВИСНОВОК

У магістерській кваліфікаційній роботі здійснено підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей.

Удосконалення методу вбудовування цифрових водяних знаків включає в себе впровадження алгоритму 3D-розпізнавання реберних вершин для знаходження більш стійких опорних точок, покращуючи таким чином стійкість невидимого водяного знаку до геометричних операцій перетворення у тривимірних моделях. Практично реалізовано програмний додаток, який був протестований шляхом проведення різноманітних атак на різні моделі. Результати тестування, що включають порівняння з початковим методом та визначення непомітності за допомогою PSNR, свідчать про те, що удосконалений алгоритм проявляє більшу стійкість до геометричних атак порівняно з початковим методом. Розробка має економічну обґрунтованість, що підтверджується результатами проведеного аналізу.

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Підвищення стійкості невидимого водяного знаку від геометричних операцій перетворення шляхом вдосконалення методу вбудовування на основі 3D-розпізнавання реберних вершин для тривимірних моделей

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ: Кафедра менеджменту та безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
(кафедра, факультет)

ПОКАЗНИКИ ЗВІТУ ПОДІБНОСТІ UNICHECK

Оригінальність 99 % Схожість 1 %

Аналіз звіту подібності (відмітити потрібне):

1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку ____________ Коваль Н.П.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи ____________ Мовчанюк М.Т.
(підпис) (прізвище, ініціали)

Керівник роботи ____________ Салієва О.В.
(підпис) (прізвище, ініціали)