**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ім. Ю.М. Потебні**
**ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ**

**КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Особливості використання технології Draco для оптимізації графічних даних в контексті Web-розробки**

Виконав: студент 2 курсу, групи 8.1213-ПЗС
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
Освітня програма Інженерія програмного забезпечення
(код і назва освітньої програми)
К. В. Береговой
(ініціали та прізвище )
Керівник доцент, к.т.н. Н. П. Полякова
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ Дісітел
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ім. Ю.М. Потебні**
**ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ**

Кафедра програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

Затверджую

Завідувач кафедри_____Т. В. Критська
"_02_" __вересня_____2024_року

**З А В Д А Н Н Я**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ**

Береговому Кірілу Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи _Особливості використання технології Draco для оптимізації графічних даних в контексті Web-розробки_____

керівник роботи _____Полякова Наталія Петрівна, доцент, к.т.н._____
( прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _28.02.2024р. № 361-е_____

2. Строк подання студентом кваліфікаційної роботи_____
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- дослідження проблеми оптимізації графічних даних в контексті Web-розробки;
- створення програмного продукту та його опис;
- аналіз результатів;
- дослідження закономірностей роботи Draco;
- формування рекомендацій щодо використання технології.

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

6. Консультанти розділів бакалаврської роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата Завдання прийняв |
|---|---|---|
| | | |

7. Дата видачі завдання 02.09.2024 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів магістерської роботи | Строк виконання етапів магістерської роботи | Примітка |
|---|---|---|---|
| 1 | Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником | 18.11.2024 | виконано |
| 2 | Аналіз предметної області | 19.11.2024 | виконано |
| 3 | Дослідження проблематики оптимізації моделей | 22.11.2024 | виконано |
| 4 | Формування вимог до програми | 23.11.2024 | виконано |
| 5 | Розробка програми | 26.11.2024 | виконано |
| 6 | Налаштування виводу програми | 27.11.2024 | виконано |
| 7 | Остаточна перевірка працездатності всіх складових | 28.11.2024 | виконано |
| 8 | Аналіз отриманої програми | 30.11.2024 | виконано |
| 9 | Заповнення таблиці отриманими даними | 01.12.2024 | виконано |
| 10 | Створення діаграм та службових таблиць | 02.12.2024 | виконано |
| 11 | Аналітика отриманих результатів | 04.12.2024 | виконано |
| 12 | Створення презентації | 05.12.2024 | виконано |
| 13 | Підготовка до захисту | 05.12.2024 | виконано |

Студент _____ К.В. Береговой
( підпис ) (прізвище та ініціали)

Керівник роботи _____ Н.П. Полякова
( підпис ) (прізвище та ініціали)

**Нормоконтроль пройдено**

Нормоконтролер _____ І.А. Скрипник
( підпис ) (прізвище та ініціали)

# АНОТАЦІЯ

Сторінок: 92

Рисунків: 15

Таблиць: 2

Джерел: 30

Береговой К. В. Особливості використання технології Draco для оптимізації графічних даних в контексті Web-розробки: кваліфікаційна робота бакалавра спеціальності 121 «Програмне забезпечення систем» / наук. керівник Н. П. Полякова. Запоріжжя : ЗНУ, 2024. 92 с.

Мета і завдання дослідження полягають у вивченні особливостей використання технології Draco для стиснення та оптимізації графічних даних у контексті Web-розробки. Основний акцент зроблено на аналізі ефективності алгоритмів стиснення та інтеграцію моделей у веб-застосунки.

У ході роботи досліджено принципи роботи технології Draco, включаючи її алгоритми кодування та декодування, а також параметри стиснення, такі як квантизація. Особливу увагу приділено аналізу впливу параметрів на швидкість стиснення та декодування моделей, а також їх інтеграцію в веб-застосунки.

Для оцінки ефективності Draco було створено бенчмарк. За його допомогою проаналізовано стиснення великої кількості 3D моделей. Отримані результати дозволили розробити практичні рекомендації, які допоможуть оптимізувати графіку у веб-застосунках. Це призведе до швидшого завантаження моделей, зменшення витрат трафіку та покращення взаємодії користувача з застосунком.

Ключові слова: *технологія Draco, стиснення графічних даних, 3D-моделі, оптимізація, веб-розробка, квантизація, декодування, продуктивність, інтернет-трафік.*

# ABSTRACT

Pages: 92

Drawings: 15

Tables: 2

Sources: 30

Berehovoi K. V. Peculiarities of using Draco technology for optimizing graphic data in the context of Web development: bachelor's qualification work in specialty 121 «Systems software» / scientific supervisor N. P. Polyakova. Zaporizhzhia: ZNU, 2024. 92 p.

The purpose and objectives of the study are to study the features of using Draco technology for compressing and optimizing graphic data in the context of Web development. The main emphasis is on analyzing the efficiency of compression algorithms, their impact on the size of 3D models and the integration of such models into web applications.

In the course of the work, the principles of the Draco technology were studied, including its encoding and decoding algorithms, as well as compression parameters, such as quantization. Special attention was paid to the analysis of the impact of parameters on the speed of compression and decoding of models, as well as their integration into web applications.

A benchmark was created to evaluate Draco’s performance. It analyzed the compression of a large number of 3D models. The results provided practical recommendations that will help optimize graphics in web applications. This will lead to faster model loading, reduced traffic costs, and improved user interaction with the application.

Keywords: *Draco technology, graphic data compression, 3D models, optimization, web development, quantization, decoding, performance, Internet traffic.*

# ЗМІСТ

ВСТУП ..... 9
1. ДОСЛІДЖЕННЯ ПРОБЛЕМИ ОПТИМІЗАЦІЇ 3D МОДЕЛЕЙ У ВЕБ-ЗАСТОСУНКАХ ..... 15
1.1 Актуальність проблеми оптимізації тривимірних моделей у веб-застосунках ..... 15
1.2 Вплив тривимірних моделей на ефективність роботи веб-застосунків ..... 16
1.3 Проблеми обробки тривимірної графіки в реальному часі ..... 17
1.4 Необхідність зменшення розміру тривимірних моделей для покращення ефективності роботи веб-застосунків ..... 17
1.5 Забезпечення сумісності тривимірних моделей із різними браузерами та платформами ..... 18
1.6 Можливості оптимізації 3D моделей в контексті мобільних пристроїв ..... 19
1.7 Висновки з розділу 1 ..... 20
2. ДОСЛІДЖЕННЯ ЗАСОБІВ ОПТИМІЗАЦІЇ ТРИВИМІРНИХ МОДЕЛЕЙ У ВЕБ-ЗАСТОСУНКАХ ..... 22
2.1 Основні засоби оптимізації тривимірних моделей ..... 22
2.1.1 Методи стиснення тримивірних моделей з втратами ..... 22
2.1.2 Методи стиснення тривимірних моделей без втрат ..... 24
2.2. Основні інструменти для оптимізації тривимірних моделей ..... 25
2.2.1 Бібліотека тривимірних моделей MeshOpt ..... 26
2.2.2. Бібліотека оптимізації тривимірних моделей Simplygon ..... 27
2.2.3 Бібліотека оптимізації тривимірних моделей glTF-Transform ..... 28
2.2.4 Бібліотека оптимізації тривимірних моделей MeshLab ..... 29
2.2.5 Бібліотека оптимізації тривимірних моделей Draco ..... 30

2.2.6 Порівняння основних інструментів стиснення тривимірних моделей ..... 31
2.3 Переваги використання бібліотеки Draco для веб-застосунків ..... 34
2.4 Алгоритми стиснення геометрії Draco ..... 35
2.4.1 Алгоритм Sequential ..... 36
2.4.2 Алгоритм Edgebreaker ..... 37
2.4.3 Порівняння алгоритмів бібліотеки Draco ..... 38
2.5 Основні параметри стиснення тривимірних моделей в бібліотеці Draco ..... 39
2.5.1 Квантизація позицій як параметр стиснення у бібліотеці Draco ..... 40
2.5.2 Квантизація нормалей як параметр стиснення у бібліотеці Draco ..... 42
2.5.3 Квантизація кольорів як параметр стиснення у бібліотеці Draco ..... 43
2.6 Формати збереження тривимірної графіки бібліотеки Draco ..... 44
2.6.1 Формат збереження тривимірної графіки glTF ..... 45
2.6.2 Формат збереження тривимірної графіки PLY ..... 47
2.6.3 Формат збереження тривимірної графіки OBJ ..... 48
2.7 Висновки до розділу 2 ..... 50
3. РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ОТРИМАННЯ СТАТИСТИКИ ВИКОРИСТАННЯ DRACO ..... 52
3.1 Формування вимог для майбутньої програми ..... 52
3.2 Огляд архітектури комп'ютерної системи ..... 53
3.2.1 Вибір платформи для розробки ..... 53
3.2.2 Визначення алгоритму роботи програми ..... 53
3.3 Реалізація комп'ютерної системи ..... 54
3.3.1 Планування файлової системи ..... 55
3.3.2 Реалізація класу TestSuite ..... 56
3.3.3 Реалізація класу Test ..... 59
3.3.4 Утиліти для генерації 3D моделі ..... 61
3.3.5 Генерація виводу отриманих даних ..... 62

3.3.6 Логіка стиснення та декомпресії ..... 63
3.3.7 Створення набору тестів ..... 65
3.3.8 Налаштування виводу ..... 68
3.4 Налаштування таблиці для порівняння результатів ..... 69
3.5 Висновки з розділу 3 ..... 70
4. ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ КОМП'ЮТЕРНОЇ СИСТЕМИ ТА ПОБУДОВА РЕКОМЕНДАЦІЙ ..... 71
4.1 Прогон програми з подальшим занесенням результатів в таблицю 71
4.2 Аналіз ефективності стиснення тривимірних моделей за допомогою Draco ..... 73
4.2.1 Аналіз ефективності стиснення Edgebreaker ..... 73
4.2.2 Аналіз ефективності стиснення Sequential ..... 77
4.2.3 Порівняння алгоритмів в контексті ефективності стиснення 79
4.2.4 Порівняння параметрів квантування в контексті ефективності стиснення ..... 79
4.2.5 Оцінка часу кодування та декодування тривимірних моделей залежно від розміру моделі ..... 81
4.2.6 Оцінка часу кодування та декодування 3D моделей залежно від параметрів квантування ..... 82
4.3 Формування рекомендацій щодо використання технології Draco .. 85
4.4 Висновки з розділу 4 ..... 85
ВИСНОВКИ ..... 87
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 88

# ВСТУП

## Актуальність теми

Сучасна веб-розробка характеризується зростанням популярності інтерактивних елементів, зокрема 3D-графіки, яка знаходить застосування у візуалізації даних, електронній комерції, розробці ігор, освітніх програмах і багатьох інших сферах. Використання 3D-моделей у веб-застосунках підвищує їх привабливість і функціональність, проте також створює низку технічних викликів, пов'язаних із великою вагою графічних даних, що може значно вплинути на продуктивність веб-застосунків і швидкість їхнього завантаження.

Технологія Draco, яка пропонує ефективне стиснення 3D-моделей, дозволяє суттєво зменшити їхній розмір без втрати якості. Це особливо важливо для забезпечення швидкого завантаження веб-застосунків і зниження обсягу трафіку, що є критичним для користувачів із повільними або обмеженими інтернет-з’єднаннями. Зростання кількості користувачів мобільних пристроїв також підкреслює важливість мінімізації ресурсів, необхідних для обробки графічних даних.

Окрім того, впровадження технології Draco може спростити обробку складних 3D-сцен та інтеграцію моделей у веб-середовище, що сприяє розширенню можливостей сучасних веб-застосунків. Актуальність дослідження полягає в необхідності детального аналізу можливостей Draco для оптимізації графічних даних і вироблення рекомендацій для її ефективного застосування у веб-розробці.

## Мета дослідження

Мета — дослідження особливостей використання технології Draco для стиснення графічних даних у веб-застосунках та розробити рекомендації для її ефективного застосування.

## Завдання дослідження

Завдання полягає в аналізі принципів роботи технології Draco, її алгоритмів кодування та декодування, в дослідженні впливу параметрів стиснення на продуктивність і швидкість обробки 3D-моделей, в розробці рекомендацій щодо застосування Draco у веб-застосунках.

**Об’єкт дослідження**

Об’єктом дослідження є процеси стиснення та розтиснення 3D-графіки у веб-застосунках.

**Предмет дослідження**

Предметом дослідження є технологія Draco. Її особливості та особливості її алгоритмів у контексті оптимізації графічних даних.

**Методи дослідження**

Аналіз технічної документації та програмних рішень, експериментальні дослідження параметрів стиснення, порівняння результатів із використанням тестових моделей.

**Наукова новизна одержаних результатів**

Наукова новизна дослідження полягає у:

1. Систематизації особливостей використання технології Draco для стиснення графічних даних у веб-застосунках.
2. Виявленні впливу параметрів квантизації на продуктивність і розмір 3D-моделей при забезпеченні збереження їхньої якості після декодування.
3. Розробці практичних рекомендацій для інтеграції стиснених 3D-моделей у веб-середовище з урахуванням особливостей сучасних технологій веб-розробки.
4. Вивченні специфіки використання алгоритмів Draco для оптимізації складних 3D-сцен у контексті зниження ресурсних витрат і покращення користувацького досвіду.

**Практичне значення одержаних результатів**

Розроблені рекомендації сприятимуть оптимізації 3D-графіки у веб-застосунках, зменшенню часу завантаження та підвищенню продуктивності, що важливо для розробників інтерактивних веб-систем.

**Апробація одержаних результатів**

Результати дослідження були представлені на XVII науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2024» [31], а також на I Міжнародній науково-практичній конференції «Інженерні інновації та розбудова національної економіки» [32].

**Глосарій**

*Draco* — технологія для стиснення та декодування 3D-графіки, розроблена компанією Google. Вона дозволяє значно зменшити обсяг 3D-моделей без втрати якості, зберігаючи їхню геометрію, топологію та атрибути. Draco широко використовується у веб-застосунках для прискорення завантаження та зменшення обсягу переданих даних.

*Sequential* — алгоритм стиснення 3D-геометрії, який виконує обробку вершин і граней моделі у визначеній послідовності. Завдяки цьому забезпечується передбачення даних для подальшого стиснення. Sequential добре підходить для моделей із регулярною структурою, що дозволяє досягти високого рівня оптимізації.

*Edgebreaker* — алгоритм для стиснення тріангуляційних сіток, який кодує топологію моделі за допомогою серії п’яти можливих операцій (CLERS). Він дозволяє ефективно зберігати структуру з'єднань між вершинами та полігонами, мінімізуючи обсяг даних.

*Квантизація* — метод зменшення точності числових даних, який використовується для стиснення геометрії моделей. Під час квантизації координати

вершин округлюються до фіксованої точності, що дозволяє зменшити обсяг даних.

*3D-модель* — цифрове представлення тривимірного об’єкта у вигляді множини вершин, граней і полігонів. Такі моделі використовуються у комп’ютерній графіці, віртуальній реальності та іграх.

*Стиснення (кодування)* — процес зменшення обсягу даних за допомогою алгоритмів оптимізації. У графіці стиснення застосовується для зменшення розміру 3D-моделей, текстур та інших графічних ресурсів.

*Розтиснення (декодування)* — відновлення даних до їх початкового вигляду після стиснення. У випадку з Draco цей процес є зворотним до стиснення і забезпечує отримання моделі, ідентичної оригіналу.

*Графічні дані* — цифрова інформація, яка представляє об’єкти, сцени або інші візуальні компоненти у вигляді координат, текстур, кольорів тощо.

*Алгоритм стиснення* — набір математичних і програмних методів, які використовуються для зменшення обсягу графічних даних, таких як вершини, текстури або атрибути моделі.

*Вершина (Vertex)* — точка у тривимірному просторі, яка визначає основні координати для побудови полігонів у моделі.

*Полігон* — багатокутник, що є базовою одиницею 3D-графіки. Полігони складаються з вершин і утворюють поверхню тривимірного об’єкта.

*Мережа багатокутників (Mesh)* — структура 3D-моделі, що складається з множини полігонів, які з’єднуються для формування поверхні об’єкта.

*Текстура* — цифрове зображення, що накладається на поверхню 3D-об’єкта для надання йому кольору, деталей або візуальних ефектів.

*Сцена (Scene)* — сукупність 3D-об’єктів, розташованих у тривимірному просторі. Сцени можуть включати моделі, текстури, джерела світла та інші елементи.

*Бітрейт* — величина, яка визначає кількість бітів, що використовується для зберігання або передачі графічних даних за одиницю часу.

*Об'єм даних* — розмір файлу, що містить 3D-модель, текстуру чи інший графічний ресурс, вимірюється у байтах або мегабайтах.

*Оптимізація* — процес покращення характеристик моделі, таких як розмір або швидкість обробки, для досягнення більшої ефективності.

*Топологія моделі* — структура з'єднань між вершинами, ребрами та гранями у тривимірному об'єкті, яка визначає його форму та взаємозв'язки.

*Рендеринг* — процес візуалізації 3D-сцени, який перетворює дані про об'єкти у зображення, що відображається на екрані.

*JSON* — текстовий формат для зберігання та передачі структурованих даних. У технології Draco використовується для опису атрибутів стиснутих моделей.

*Фрагментація даних* — поділ великих 3D-моделей на менші частини для покращення ефективності стиснення та передачі.

*Індексований рендеринг* — метод обробки 3D-графіки, що використовує індекси вершин для зменшення обсягу даних, що передаються між GPU та процесором.

*Асети (Assets)* — будь-які ресурси, що використовуються у веб-застосунках, зокрема моделі, текстури, анімації та шейдери.

*GPU (Graphics Processing Unit)* — спеціалізований процесор, призначений для швидкого оброблення графічних даних і виконання рендерингу.

*Воксель (Voxel)* — об'ємний аналог пікселя, який представляє елемент тривимірного простору в моделі або сцені.

*Кешування* — збереження даних у тимчасовому сховищі для прискорення доступу до них під час повторного використання.

*Бібліотека* — набір готових функцій і компонентів, які спрощують роботу з графічними чи іншими типами даних у програмуванні.

*Топологічна оптимізація* — модифікація структури моделі, що дозволяє зменшити кількість полігонів при мінімальному впливі на її вигляд.

*Двійковий формат (Binary Format)* — спосіб зберігання даних, що використовує компактне представлення, зменшуючи обсяг файлів та прискорюючи їх обробку.

*Лод (Level of Detail)* — метод зменшення деталізації моделі залежно від її відстані до камери, що зменшує навантаження на систему рендерингу.

*Координати моделі* — значення, що визначають розташування вершин у локальній системі координат 3D-об’єкта.

*Глобальні координати* — розташування об’єкта у загальній системі координат сцени, враховуючи трансформації моделі.

*Трикутник (Triangle)* — базовий полігон у 3D-графіці, який складається з трьох вершин і використовується для побудови складніших форм.

*Ребро (Edge)* — лінія, яка з’єднує дві вершини в тривимірній моделі, формуючи каркас полігонів.

*Декартові координати* — система координат для визначення положення точок у просторі за допомогою трьох чисел: x, y та z.

# 1. ДОСЛІДЖЕННЯ ПРОБЛЕМИ ОПТИМІЗАЦІЇ 3D МОДЕЛЕЙ У ВЕБ-ЗАСТОСУНКАХ

## 1.1 Актуальність проблеми оптимізації тривимірних моделей у веб-застосунках

Проблема оптимізації тривимірних моделей для веб-застосунків є актуальним напрямком досліджень у галузі комп'ютерної графіки. Зростаюча популярність веб-застосунків, що використовують тривимірну графіку, стимулює пошук нових методів стиснення, спрощення та адаптивної передачі тривимірних моделей. Результати дослідження [1] підтверджують необхідність подальших розробок у цій галузі.

Р. Скопіньо, М. Каллієрі, М. Деллепіане та інші [2] відзначають, що одна з основних труднощів полягає у досягненні балансу між високою якістю моделей і їхнім розміром, оскільки великі файли значно сповільнюють завантаження та рендеринг на веб-сторінках. Для розв’язання цієї проблеми застосовують методи стиснення, зокрема алгоритми, що дозволяють зменшити кількість полігонів у моделі та спростити текстури, зберігаючи при цьому високий рівень візуальної якості.

Застосування сучасних методів стиснення, таких як Draco, дозволяє значно зменшити кількість полігонів та текстурних даних 3D-моделей без помітної втрати візуальної якості. Це робить можливим ефективну передачу великих обсягів 3D-контенту в умовах обмеженої пропускної здатності мережі, що є необхідною умовою для  правильної роботи веб-ігор та AR-застосунків.

Окрім технічних аспектів, важливим є також підвищення інтерактивності веб-застосунків. Наприклад, в дослідженні з веб-рендерінгу для AR [1] запропоновано багаторівневі моделі оптимізації, що забезпечують зменшення часу завантаження моделей на 46%, зберігаючи їх якість навіть у середовищах з низькою пропускною здатністю мережі.

Отже, вдосконалення технологій оптимізації 3D-моделей є невід’ємною частиною розвитку сучасних веб-застосунків. Це дозволяє не лише підтримувати високий рівень якості, але й забезпечує ефективне використання ресурсів та покращує взаємодію користувачів з застосунком.

## 1.2 Вплив тривимірних моделей на ефективність роботи веб-застосунків

Якість та ефективність роботи веб-застосунків значною мірою залежить від того, наскільки добре оптимізовані 3D-моделі, які в них використовуються. Від цього залежить швидкість завантаження сторінки, плавність анімації та загальне враження користувача. Особливо важливою є оптимізація для мобільних пристроїв, де обмежені обчислювальні ресурси можуть негативно вплинути на роботу застосунку.

Однією з основних проблем є великий обсяг даних, який передається під час завантаження 3D-моделей. Оптимізація моделі дозволяє зменшити цей обсяг без значних втрат у якості візуалізації, що, своєю чергою, зменшує час завантаження і знижує вимоги до пропускної здатності мережі. Наприклад, оптимізовані моделі використовують менше пам'яті та процесорних ресурсів, що дозволяє забезпечити стабільну роботу навіть на менш потужних пристроях.

Використання динамічного завантаження 3D-моделей є ефективним способом оптимізації веб-застосунків. Цей метод дозволяє зменшити початковий розмір файлу, що пришвидшує завантаження сторінки. Крім того, динамічне завантаження та ефективне управління ресурсами забезпечують плавну роботу 3D-графіки в реальному часі, особливо на мобільних пристроях з обмеженими ресурсами.

Завдяки цим стратегіям вдається забезпечити високу якість візуалізації при одночасному збереженні високої продуктивності, що дозволяє адаптувати веб-застосунки до різних пристроїв та мережевих умов.

## 1.3 Проблеми обробки тривимірної графіки в реальному часі

Обробка 3D графіки в реальному часі вимагає вирішення комплексу технічних проблем, пов'язаних з необхідністю забезпечити високу швидкість рендерингу, деталізовану візуалізацію та інтерактивність. Для веб-застосунків особливо важливо оптимізувати процес рендерингу, щоб забезпечити плавну роботу на різних пристроях, навіть при використанні складних 3D моделей. Це включає в себе вибір ефективних алгоритмів рендерингу, оптимізацію геометрії моделей, використання ефективних текстур та інші методи.

Одним з найскладніших завдань при роботі з 3D-сценами є забезпечення плавної відображення великої кількості деталей. Для розв'язання цієї проблеми, як показано в дослідженні [3], ефективно використовувати методи динамічного змінення рівня деталізації (LOD). Цей підхід дозволяє зменшувати кількість полігонів у віддалених об'єктах, зменшуючи таким чином навантаження на систему і забезпечуючи плавну роботу.

Як у своєму дослідженні [4] зазначають, Н. Бунсаванг, К. Мотонобу і П. Чанпум іншими критичними аспектами є обробка текстур і тіней, що надають реалістичний вигляд 3D сценам. Однак високоякісні текстури й складні методи рендерингу вимагають великих обчислювальних ресурсів і можуть спричинити затримки у рендерингу, що знижує загальну ефективність. Тому необхідна постійна оптимізація процесу рендерингу для забезпечення плавності взаємодії в реальному часі.

## 1.4 Необхідність зменшення розміру тривимірних моделей для покращення ефективності роботи веб-застосунків

Швидкість завантаження, рендерингу та взаємодії з 3D-моделями в веб-застосунках безпосередньо залежить від їх розміру та складності. Для розв'язання цієї проблеми використовують алгоритми спрощення сітки, які дозволя-

ють зменшити кількість полігонів в моделі без помітного погіршення її зовнішнього вигляду. Як показано в дослідженні [5], методи кластеризації вершин та згортання ребер є ефективними інструментами для оптимізації 3D-моделей. Застосування цих алгоритмів дозволяє знизити навантаження на процесор та відеокарту, що особливо важливо для інтерактивних застосунків, таких як онлайн-ігри.

Окрім спрощення геометрії моделей, для підвищення продуктивності веб-застосунків з 3D-графікою застосовують алгоритми оптимізації текстур. Ці алгоритми дозволяють стиснути текстури без помітних втрат якості, що значно зменшує обсяг переданих даних між клієнтом та сервером, що особливо важливо для масштабних веб-застосунків.

Таким чином, для створення сучасних веб-застосунків з високою продуктивністю та привабливим інтерфейсом необхідно проводити ретельну оптимізацію 3D-моделей, використовуючи спрощення геометрії та стиснення текстур.

## 1.5 Забезпечення сумісності тривимірних моделей із різними браузерами та платформами

Сумісність 3D-моделей у різних браузерах і платформах залишається критичною проблемою через різні підтримки технологій візуалізації. Дослідження [6] підкреслює, що різні веб-браузери по-різному реалізують WebGL і пов’язані технології 3D-рендерінгу, що може призвести до розбіжностей у відображенні моделі, швидкості роботи застосунку та інтерактивності. Хоча Chrome, Firefox і Safari, можуть мати схожі можливості, ефективність роботи може значно відрізнятися залежно від пристрою, операційної системи та специфікацій обладнання. Це безпосередньо впливає на час і якість візуалізації, потенційно призводячи до неоптимальної взаємодії з користувачем.

Крім того, забезпечення сумісності 3D-моделей з різноманітними пристроями є складним завданням. Різні пристрої мають неоднакові характеристики

апаратного забезпечення, що ускладнює підтримання стабільної якості візуалізації та швидкодії. Для розв'язання цієї проблеми часто доводиться вдаватися до оптимізації моделей, адаптуючи їх до менш потужних пристроїв без втрати зручності використання.

Щоб забезпечити сумісність, дослідники [6] підкреслюють важливість стандартизації форматів файлів і підходів до відтворення. Це може допомогти досягти більшої узгодженості між платформами та підвищити доступність і ефективність роботи 3D-моделей у веб-застосунках.

## 1.6 Можливості оптимізації 3D моделей в контексті мобільних пристроїв

На відміну від потужних настільних систем, мобільні пристрої мають обмежені можливості для відображення 3D-графіки. Дослідження [7] підкреслює необхідність оптимізації 3D-моделей для мобільних пристроїв. Обмежені обчислювальні потужності та пам’ять мобільних платформ вимагають застосування спеціальних методів.

Децимація та технологія рівнів деталізації (LOD) є основними методами оптимізації 3D-моделей для мобільних пристроїв. Децимація дозволяє зменшити кількість полігонів у моделі, а LOD динамічно змінює рівень деталізації залежно від відстані до об'єкта та кута огляду, що дозволяє зберегти високу деталізацію на близьких відстанях і спростити модель на дальніх відстанях.

Крім того, одним з ефективних способів оптимізації 3D-моделей для мобільних пристроїв є стиснення текстур. Цей процес дозволяє значно зменшити обсяг пам'яті, необхідної для зберігання моделі, і прискорити її завантаження, що особливо важливо для веб-застосунків. Впровадження передових алгоритмів стиснення може допомогти мінімізувати дані, необхідні для високоякісних текстур, зберігаючи візуальну цілісність.

Інший шлях до покращення ефективності роботи 3D-моделей на мобільних пристроях лежить через використання мобільних графічних процесорів.

Незважаючи на меншу потужність порівняно з настільними аналогами, сучасні мобільні GPU дозволяють реалізувати складні алгоритми візуалізації. Оптимізуючи моделі під специфіку мобільних API та використовуючи ефективні конвеєри рендерингу, ми можемо досягти значного підвищення швидкодії та якості графіки в мобільних застосунках.

Таким чином, для забезпечення максимальної продуктивності 3D-моделей на мобільних пристроях необхідний комплексний підхід, що включає спрощення геометрії, стиснення текстур та використання оптимізованих для графічних процесорів методів візуалізації. Такий підхід дозволяє адаптувати 3D-графіку до обмежень мобільного обладнання без втрати якості зображення.

### 1.7 Висновки з розділу 1

1. Проблема оптимізації тривимірних моделей для веб-застосунків є надзвичайно актуальною, оскільки саме від її вирішення залежить ефективність роботи сучасних застосунків, що активно використовують 3D-графіку.

2. Одним із ключових напрямів розв'язання цієї проблеми є застосування алгоритмів стиснення, які дозволяють значно зменшити кількість полігонів у моделі та стиснути текстури, зберігаючи при цьому високий рівень візуальної якості. Важливим доповненням до цього є динамічне завантаження 3D-моделей, яке дозволяє оптимізувати початковий розмір файлів і забезпечити плавність роботи графіки в реальному часі.

3. Для того, щоб підвищити продуктивність і одночасно зберегти візуальну якість, широко використовуються методи рівнів деталізації (LOD). Цей підхід дозволяє адаптивно змінювати кількість полігонів у моделі залежно від відстані до об’єкта. Разом із цим застосовуються техніки текстурної оптимізації, які дають змогу стиснути текстури без помітної втрати їхньої якості.

4. Різні веб-браузери по-різному реалізують підтримку WebGL і технологій 3D-рендерингу, що може спричинити відмінності у відображенні моде-

лей та швидкодії застосунків. Для мінімізації таких проблем важливо стандартизувати формати файлів і використовувати уніфіковані підходи до відтворення, що забезпечить узгодженість роботи застосунків на різних пристроях і операційних системах.

5. Водночас оптимізація моделей для мобільних пристроїв вимагає особливої уваги. Використання технологій децимації, рівнів деталізації та стиснення текстур дозволяє значно зменшити обсяг пам’яті та обчислювальні ресурси, необхідні для роботи з 3D-графікою, забезпечуючи плавність взаємодії навіть на пристроях із низькою продуктивністю.

# 2. ДОСЛІДЖЕННЯ ЗАСОБІВ ОПТИМІЗАЦІЇ ТРИВИМІРНИХ МОДЕЛЕЙ У ВЕБ-ЗАСТОСУНКАХ

## 2.1 Основні засоби оптимізації тривимірних моделей

Оптимізація тривимірних моделей є необхідною умовою для ефективного функціонування веб-застосунків, оскільки безпосередньо впливає на результативність роботи та швидкість завантаження. Для зменшення обсягу даних тривимірних моделей без втрати візуальної якості та функціональності застосовують різноманітні алгоритми та методи.

Як вказує Р. Грицай [8], методи стиснення даних можна розділити методи з втратами і методи без втрат.

1. Методи стиснення з втратами — це методи, що досягають вищого ступеня стиснення за рахунок видалення надлишкової або малозначущої інформації з даних. В результаті, відновлений файл відрізняється від оригіналу, але при цьому зберігає достатню якість для сприйняття. Застосовується переважно для стиснення мультимедійних даних (зображення, аудіо, відео), де невеликі втрати якості можуть бути непомітними для користувача.

2. Методи стиснення без втрат — це методи, що забезпечують точну реконструкцію вихідних даних після декомпресії. Іншими словами, відновлений файл є бітовою копією оригіналу. Такий підхід використовується, коли будь-яка втрата інформації є неприпустимою, наприклад, для стиснення текстових документів, виконуваних кодів або наукових даних.

### 2.1.1 Методи стиснення тримивірних моделей з втратами

Алгоритми стиснення з втратами обмінюють певний рівень якості на більші коефіцієнти стиснення, що робить їх придатними для застосунків, де незначне зниження якості суттєво не впливає на роботу користувача. Ці алгоритми зосереджуються на зменшенні даних вершин і деталей текстури, що забезпечує швидшу візуалізацію та менші розміри файлів.

Автор зазначає [9], що одним з найпоширеніших методів втратного стиснення тривимірних моделей є метод мінімізації квадратичної помилки (Quadric Error Metrics). Суть методу полягає в оптимізації геометрії моделі шляхом зменшення кількості вершин за умови збереження загальної форми. QEM широко застосовується в застосунках реального часу, наприклад, відеоіграх та середовищах віртуальної/доповненої реальності (VR/AR), де пріоритет віддається забезпеченню плавної взаємодії користувача за рахунок деякої втрати деталізації моделі.

В. Відаль та інші [10] стверджують, що прогресивні сітки є ефективним методом втратного стиснення тривимірних моделей. Ця методика передбачає поетапне спрощення геометрії моделі, починаючи з низькодеталізованого представлення. За необхідності, модель може бути поступово деталізована шляхом додавання нової геометричної інформації. Прогресивні сітки широко застосовуються в інтерактивних веб-застосунках, де рівень деталізації моделі динамічно адаптується до обчислювальних ресурсів користувача та вимог до візуалізації

Г. Лаву та інші [11] зазначає, що в тривимірній графіці широко застосовується метод стиснення текстур. Найпоширенішими алгоритмами стиснення текстур є DXT та ETC. Вони базуються на блоковому поділі текстури, де кожен блок замінюється набором кольорів та індексів, що дозволяє досягти значного зменшення розміру файлу без суттєвої втрати візуальної якості. Інші методи стиснення текстур, такі як ASTC та BC7, пропонують більш адаптивний підхід, застосовуючи різні схеми стиснення до різних блоків текстури залежно від їх особливостей. Це дозволяє оптимізувати співвідношення між розміром файлу та рівнем деталізації зображення. Техніки стиснення текстур знайшли широке застосування в ігровій індустрії та інших сферах, де критично важлива ефективна обробка графічних даних.

### 2.1.2 Методи стиснення тривимірних моделей без втрат

Стиснення тривимірних моделей без втрат має на меті зменшити обсяг даних, необхідний для збереження та передавання 3D моделей, при цьому зберігаючи всі важливі деталі, включаючи геометрію, текстуру та кольори. Такі методи потрібні в ситуаціях, коли критично важливо зберегти всі деталі та точність 3D-моделі, наприклад, при створенні архітектурних візуалізацій, медичних зображень або прототипів продуктів.

Один із основних методів для досягнення стиснення без втрат — це кодування сіток. Як вказують Ю. Лі, Х. Чжэн, Л. Цзоу та інші [12], метод Vertex List Encoding, який зберігає точні координати вершин моделі, дозволяє зберегти всі деталі без будь-яких змін. У цьому підході кожна точка сітки моделі зберігається в її первісному вигляді, без використання квантування, що забезпечує високу точність відтворення. Для збереження топологічної структури моделі використовують Face Connectivity Encoding, яке зберігає інформацію про з'єднання між вершинами у вигляді індексів. Це дозволяє зменшити обсяг даних без втрати суттєвих деталей, зберігаючи всю топологічну інформацію.

Ще одним важливим методом є кодування об'ємів, яке дозволяє зберігати кожен осередок об'єму моделі без втрат. Використання детального опису кожного voxеля зберігає всі характеристики, включаючи щільність, кольори та текстури, у їх первісному вигляді. Це забезпечує точність відтворення моделі без будь-яких змін. Як зазначають З. Тан, Х. Є, Х. Чжан [13] Octree Encoding — це ще один підхід, який поділяє модель на октави, зберігаючи інформацію про кожну октаву. Кожна октава представляє певний об'єм у просторі, зберігаючи координати, нормалі та текстуру. Цей метод дозволяє зберегти точність геометрії моделі, зменшуючи обсяг даних шляхом збереження важливих частин об'єму.

Для стиснення поверхні тримірних моделей без втрат використовуються методи, які зберігають інформацію про геометрію без жодної втрати деталей. Vertex Clustering є одним із таких методів, де групи подібних вершин об'єднуються, що зменшує обсяг даних, не впливаючи на точність моделі. Кодування

Progressive Meshes також дозволяє зберігати моделі з різними рівнями деталізації, зберігаючи всі важливі деталі без жодних втрат. К. Лю [14] підкреслює, що використання LOD (Level of Detail) Compression дозволяє зберігати кілька версій моделі з різними рівнями деталізації, що дозволяє оптимізувати передавання даних відповідно до доступної пропускної здатності.

Сучасні нейронні мережі також активно використовуються для стиснення тримірних моделей без втрат. Автоенкодери, наприклад, дозволяють зменшити обсяг даних, зберігаючи всі суттєві характеристики моделі. Автоенкодер навчається знаходити та зберігати важливі деталі моделі у меншому просторі, використовуючи методи квантування та кодування. Генеративні моделі, такі як GANs (Generative Adversarial Networks), створюють зменшені версії моделей, які потім можуть бути відновлені з високою точністю. Ці методи забезпечують високу ефективність стискання, зберігаючи деталі без втрат.

## 2.2. Основні інструменти для оптимізації тривимірних моделей

Інструменти оптимізації тривимірних моделей — це програмні комплекси, що застосовуються для зменшення складності та обсягу геометричних даних тривимірних об’єктів без суттєвого погіршення візуальної якості. Їхнє основне призначення полягає в оптимізації обчислювальних ресурсів, необхідних для обробки великомасштабних тривимірних сцен, зокрема, пришвидшенні процесів візуалізації та зменшенні вимог до об’єму оперативної пам’яті та обчислювальної потужності. Типові методи оптимізації включають: редукцію полігональної сітки, генерацію рівнів деталізації (LOD), стиснення геометричних даних та оптимізацію топології.

Аналізуючи технічну документацію, ми виділили основні інструменти, які використовуються для оптимізації тривимірних моделей, що будуть розглянуті більш детально у наступних підрозділах:

- MeshOpt (спеціалізується на оптимізації геометрії моделей, забезпечуючи високу продуктивність рендерингу в реальному часі завдяки ефективним алгоритмам стиснення та упорядкування вершин).
- Draco (пропонує ефективне стиснення 3D-даних, що дозволяє значно зменшити розмір файлів без помітної втрати якості).
- Simplygon (дозволяє генерувати моделі з різними рівнями деталізації (LOD) та оптимізувати геометрію).
- glTF-Transform (оптимізує моделі для швидкого завантаження на веб-ресурсах та в мобільних застосунках).
- MeshLab (надає широкий спектр фільтрів для обробки 3D-моделей, спрощення геометрії та покращення якості).

Вибір конкретного інструменту залежить від специфічних вимог проекту, таких як зменшення розміру файлів, підвищення швидкості рендерингу або створення моделей з адаптивною деталізацією.

### 2.2.1 Бібліотека тривимірних моделей MeshOpt

MeshOptimizer (MeshOpt) — це бібліотека, розроблена Капулькіним А. А., яка фокусується на оптимізації стиснення даних 3D-сітки. Ця бібліотека реалізує низку алгоритмів, спрямованих на оптимізацію геометричних даних (мешів) для підвищення ефективності рендерингу. Зокрема, meshOptimizer мфстить алгоритм генерації мешлетів, який дозволяє розбивати складні геометричні моделі на менші, більш керовані підмножини.

Алгоритм генерації мешлетів в meshOptimizer базується на побудові структури даних, що відображає топологію меша (суміжність трикутників і вершин). Для кожного трикутника обчислюються такі характеристики як центроїд, нормаль і площа. Отримані дані використовуються для побудови дерева kd, яке слугує для ефективного пошуку найближчих трикутників під час формування мешлетів.

У своєму дослідженні [15] М. Дженсен, Я. Фрісвад та Й. Берентцен зазначають, що критерії включення трикутника до мешлета у MeshOpt визначаються на основі кількох факторів:

- повторне використання вершин (трикутники, які мають спільні вершини з вже включеними до мешлета, отримують вищу оцінку);
- збільшення площі мешлета (трикутники, додавання яких призводить до меншого збільшення площі мешлета, вважаються більш пріоритетними);
- нормаль трикутника (алгоритм дозволяє задавати вагу для нормалі трикутника, що впливає на вибір наступного трикутника для включення до мешлета).

Бібліотека MeshOpt знаходить широке застосування у веб-розробці, інтегруючись через розширення EXT_meshopt_compression в стандартний формат обміну тривимірними даними glTF. Це дозволяє досягти значного підвищення ефективності обробки та візуалізації тривимірних моделей у веб-застосунках та інших програмних продуктах, що орієнтовані на використання у реальному часі.

### 2.2.2. Бібліотека оптимізації тривимірних моделей Simplygon

Simplygon — це програмна бібліотека, створена Donya Labs в 2009 році у Швеції, яка забезпечує автоматизовану оптимізацію геометричних даних тривимірних моделей. Завдяки використанню алгоритмів спрощення сіток, Simplygon дозволяє знизити полігонаж моделей без значної втрати візуальної якості, що є необхідною умовою для забезпечення ефективного рендерингу в інтерактивних 3D-застосунках.

Ю. Арво, А. Еуранто, Л. Ярвенпяя та інші дослідники [16] відзначають, що програмне забезпечення Simplygon є інноваційним рішенням у галузі 3D-комп’ютерної графіки. Воно реалізує власні алгоритми автоматичної оптимізації 3D-моделей шляхом створення ієрархічних структур даних, відомих як

сітки рівня деталізації (LOD). Цей підхід дозволяє динамічно змінювати деталізацію моделі залежно від відстані до спостерігача, що є важливим для забезпечення плавної роботи ігор та інших інтерактивних застосунків.

Відмінною рисою Simplygon є хмарна архітектура, яка надає користувачам гнучкий доступ до обчислювальних ресурсів. Також, Software Development Kit (SDK) дозволяє інтегрувати технології Simplygon в інші програмні продукти за допомогою плагінів, розширюючи таким чином їх функціональність.

Крім того, для підвищення якості автоматичної оптимізації, Simplygon дозволяє художникам задавати додаткові параметри, такі як ваги вершин, що дозволяє керувати процесом спрощення моделі з урахуванням її семантичних характеристик.

Інтеграція бібліотеки Simplygon з різноманітними програмними платформами, включаючи продукти корпорації Microsoft, значно розширила сферу її застосування. Завдяки цьому, Simplygon стала доступною для широкого кола розробників, надаючи їм ефективний інструментарій для створення та обробки тривимірних графічних ресурсів, що відповідають високим вимогам сучасних цифрових проектів.

### 2.2.3 Бібліотека оптимізації тривимірних моделей glTF-Transform

glTF-Transform — це програмна бібліотека, що надає розширений набір інструментів для оптимізації та модифікації тривимірних моделей, представлених у форматі glTF. Формат glTF (GL Transmission Format) став одним з найбільш популярних для передачі 3D-даних в Інтернеті. У дослідженні [17] зазначається, що glTF має високу ефективність, не лише завдяки тому, що якісно стискає розмір самого файлу, а ще й тому, що має вищу середню швидкість завантаження, ніж інші формати.

Бібліотека glTF-Transform дозволяє застосовувати алгоритми стиснення із втратами, що значно зменшує розмір файлів. Паралельно з оптимізацією текстур glTF-Transform пропонує інструменти для спрощення геометрії моделі

шляхом скорочення кількості полігонів та об’єднання вершин. Така оптимізація дозволяє адаптувати рівень деталізації моделі під обчислювальні потужності різних пристроїв. Для подальшого зменшення розміру файлу бібліотека автоматично видаляє елементи моделі, які не використовуються, такі як матеріали, шейдери та анімації. Бібліотека також забезпечує перетворення моделей у формат, що підтримується більшістю сучасних рендерингових движків. Це дозволяє уникнути проблем, пов’язаних із різними форматами моделей, та гарантує стабільну роботу в різних середовищах.

На відміну від універсальних інструментів для маніпуляцій з 3D-моделями або спеціалізованих програмних рішень, *glTF-Transform* пропонує інтегровану платформу для швидкої, беззбиткової оптимізації і створення скриптів, що підходить для пакетної обробки моделей у веб та офлайн середовищах.

### 2.2.4 Бібліотека оптимізації тривимірних моделей MeshLab

MeshLab — це бібліотека оптимізації з багатофункціональними інструментами, що була спеціально розроблена Visual Computing Lab для обробки та аналізу тривимірних геометричних моделей. Як зауважують дослідники у своїй роботі [18] MeshLab надає широкий набір інструментів, що дозволяють оптимізувати, редагувати, аналізувати та перетворювати геометричні дані. Бібліотека пропонує різні алгоритми згладжування та видалення шумів. Крім цього, програма дозволяє коригувати напрямки нормалей граней, що є важливою умовою для реалістичної візуалізації.

Однією з ключових особливостей MeshLab є можливість редукції полігонів, тобто зменшення кількості граней у моделі без значної втрати деталізації. Крім роботи з геометрією, MeshLab також надає інструменти для роботи з текстурами, дозволяючи створювати UV-розгортки, перевіряти та виправляти накладання текстурних координат, а також генерувати карти нормалей та інші карти, необхідні для реалістичного відображення матеріалів.

Аналіз геометрії — ще одна важлива функція MeshLab. Програма дозволяє виявляти різні дефекти моделі — самоперетинання, не валідні елементи

геометрії та інші, а також розраховувати геометричні характеристики, такі як площа поверхні та об’єм. MeshLab дозволяє здійснювати конвертацію моделей між різними системами САПР та програмними пакетами.

MeshLab є безкоштовним програмне забезпечення з відкритим вихідним кодом, що робить його доступним для широкого кола користувачів.

### 2.2.5 Бібліотека оптимізації тривимірних моделей Draco

Draco — це бібліотека оптимізації тривимірних графічних даних, розроблена компанією Google, основною метою якої є стиснення 3D-моделей для ефективнішого зберігання, передачі через мережу та швидшого рендерингу в інтерактивних застосунках.

Draco працює з геометричними та топологічними даними моделей, оптимізуючи зберігання координат вершин, нормалей, кольорів і текстурних координат, а також зв’язків між вершинами, гранями й ребрами. Використовуючи гібридні алгоритми кодування, бібліотека поєднує побітове та ентропійне кодування, що дозволяє зменшити обсяг файлів при збереженні важливих деталей. Завдяки використанню сучасних алгоритмів стиснення, навіть складні тривимірні моделі у форматі glTF можуть бути ефективно компресовані, що дозволяє значно зменшити обсяг переданих даних та оптимізувати їх транспортування через мережу.

У своєму дослідженні [19] С. Пачеко-Гутьєррес, І. Калісканеллі та Р. Скілтон аналізують ефективність бібліотеки Draco для стиснення даних тривимірних моделей. Зокрема, вони досліджують здатність бібліотеки зменшувати розмір файлів без значної втрати якості, вплив стиснення на час рендерингу моделей та баланс між рівнем компресії та деталізацією. Результати дослідження підтверджують, що використання Draco істотно покращує ефективність роботи, особливо в умовах обмеженої пропускної здатності мережі, і дозволяє забезпечити інтерактивну візуалізацію тривимірного контенту в реальному часі.

Draco також демонструє свою ефективність у галузі зберігання та обробки тривимірних даних завдяки підтримці роботи з різноманітними типами моделей, включаючи сітчасті структури й хмари точок. Це забезпечує широку сферу застосування, від архітектурного моделювання до створення високоякісних ігрових середовищ. Унікальність бібліотеки полягає у збереженні текстурної деталізації та топологічної точності навіть за значного зменшення розміру файлів, що дозволяє створювати більш реалістичні сцени без зайвого навантаження на систему. Завдяки відкритому вихідному коду Draco інтегрується в різні платформи й фреймворки, що робить її доступною для спільнот розробників і стимулює подальші дослідження й оптимізації в галузі компресії тривимірних даних.

### 2.2.6 Порівняння основних інструментів стиснення тривимірних моделей

Для детального порівняння основних інструментів оптимізації тривимірних моделей — MeshOpt, Simplygon, glTF-Transform, MeshLab і Draco — необхідно розглянути їх ключові особливості, сфери застосування та підходи до оптимізації. Незважаючи на різницю в підходах, кожна з цих бібліотек вирішує спільні завдання — покращення ефективності рендерингу, зменшення розміру файлів і спрощення обробки тривимірних даних.

MeshOpt зосереджується на стисканні даних і оптимізації геометричних даних для підвищення ефективності рендерингу. Він використовує алгоритми, які допомагають розділяти складні моделі на більш керовані мешлети, що спрощує їх обробку. Завдяки інтеграції через EXT_meshopt_compression у формат glTF, MeshOpt знаходить широке застосування у веб-розробці.

Simplygon пропонує автоматизовану оптимізацію з використанням ієрархічних структур LOD і хмарної архітектури, що забезпечує гнучкість і доступ до обчислювальних ресурсів. Simplygon також дозволяє художникам задавати додаткові параметри для тонкого налаштування процесу спрощення моделі.

Тому цей інструмент ідеально підходить для створення і обробки моделей у реальному часі.

glTF-Transform орієнтований на формат glTF і надає широкий спектр інструментів для оптимізації. Його алгоритми дозволяють адаптувати рівень деталізації моделей під різні пристрої, що робить його чудовим вибором для вебдизайну та використання в різних рендерингових движках. glTF-Transform допомагає забезпечити швидке завантаження моделей і стабільність роботи, завдяки автоматичному видаленню не використаних елементів — матеріалів та анімацій.

MeshLab відрізняється своєю багатофункціональністю, пропонуючи інструменти для згладжування, видалення шумів, редукції полігонів і створення карт нормалей. Цей інструмент корисний для аналізу геометрії моделей, виявлення дефектів і підготовки моделей до рендерингу. MeshLab особливо цінний для фахівців, які працюють з високодеталізованими моделями, завдяки своїй можливості редагувати і оптимізувати тривимірні дані зі збереженням високого рівня деталізації.

Draco, розроблена компанією Google, пропонує унікальні можливості стиснення, використовуючи гібридні алгоритми кодування для зменшення обсягу файлів. Цей інструмент забезпечує високу швидкість передачі даних і ефективне зберігання тривимірних моделей. Draco підтримує різноманітні типи моделей, включаючи сітчасті структури і хмари точок, що розширює його застосування від архітектурного моделювання до створення якісних ігрових середовищ та завдяки відкритим вихідним кодам легко інтегрується в вебзастосунки.

З метою об'єктивного порівняння функціональних можливостей кожного з розглянутих інструментів оптимізації тривимірних моделей було розроблено порівняльну таблицю (Таблиця 1), яка надає візуальне представлення їхніх ключових характеристик.

Таблиця 1 — Порівняння основних інструментів оптимізації тривимірних моделей

| **Інструмент оптимізації** | **Основні особливості** | **Застосування** |
|---|---|---|
| MeshOpt | Оптимізація даних тривимірних сіток, використання алгоритмів для розбиття моделей на мешлети, підтримка формату glTF | Веб-розробка, підвищення ефективності рендерингу, інтеграція в застосунки, орієнтовані на реальне часі |
| Simplygon | Автоматична оптимізація з використанням ієрархічних структур LOD і хмарної архітектури | Ігри, інтерактивні застосунки, створення високоякісних тривимірних графічних ресурсів |
| glTF-Transform | Стиснення з втратами, скорочення полігонів, підтримка формату glTF | Веб-дизайн, оптимізація моделей для різних пристроїв, стабільна робота в рендерингових движках |
| MeshLab | Багатофункціональні інструменти для аналізу, редукції полігонів, створення карт нормалей і UV-розкладок | Обробка і редагування тривимірних моделей, підготовка до рендерингу, аналіз геометрії |
| Draco | Гібридні алгоритми кодування для стиснення тривимірних моделей, підтримка різноманітних моделей | Архітектурне моделювання, створення якісних ігрових середовищ, висока швидкість передачі даних |

Таким чином, розглянуті інструменти пропонують широкий спектр методів оптимізації та обробки тривимірних моделей, кожен з яких має свої особливості та сфери застосування. Вибір конкретного інструментального засобу визначається специфічними вимогами проекту та поставленими завданнями.

## 2.3 Переваги використання бібліотеки Draco для веб-застосунків

На підставі порівняння різних бібліотек оптимізації тривимірних моделей, що було наведено вище можна зробити висновок, що Draco вирізняється своєю унікальністю та перевагами у сфері застосування для створення веб-застосунків. Вона використовує сучасні гібридні алгоритми кодування, які забезпечують ефективне зменшення обсягу файлів без значної втрати якості, що робить її ідеальним вибором для швидкої передачі і високоякісної візуалізації тривимірного контенту в реальному часі.

Як зазначають П. Марті, К. Лін, С. Брандт та інші [20], Draco є перспективним інструментом для підвищення ефективності веб-застосунків завдяки своїм унікальним характеристикам. Ключовою перевагою Draco є висока ефективність стиснення 3D-моделей, що призводить до суттєвого зменшення розміру файлів. Це, в свою чергу, позитивно впливає на швидкість завантаження контенту та знижує вимоги до пропускної здатності мережі. Таким чином, Draco є оптимальним рішенням для веб-застосунків, орієнтованих на користувачів з різноманітних географічних регіонів, включаючи ті, де спостерігається низька швидкість Інтернету.

Ще одним важливим критерієм оцінки веб-застосунків, що використовують 3D-графіку є час відгуку системи. Метод стиснення Draco пропонує ефективне рішення для зменшення часу відгуку таких застосунків завдяки своїм алгоритмам стиснення. Зменшення обсягу даних, необхідних для представлення 3D-моделей, досягається за рахунок застосування передових методів стиснення. Це призводить до зменшення обчислювальних витрат на рендеринг графіки, що, в свою чергу, забезпечує більш швидкий відгук системи на дії користувача.

Також Draco демонструє високий рівень сумісності з широко використовуваним стандартом обміну тривимірними моделями glTF. Це забезпечує

безшовну інтеграцію 3D-моделей, стиснутих за допомогою Draco, у різноманітні веб-програми без необхідності додаткових перетворень або розробки власних форматів даних. Підтримка glTF робить Draco універсальним рішенням для оптимізації 3D-контенту в веб-застосунках. Оскільки формат glTF отримує все більш широке розповсюдження та вбудовується в сучасні веб-браузери, розробники можуть легко інтегрувати 3D-моделі, стиснуті за допомогою Draco, у свої проекти. Це суттєво спрощує процес розробки та підвищує ефективність використання 3D-графіки в веб-застосунках.

Таким чином, висока ефективність стиснення Draco дозволяє значно скоротити розмір файлів, що позитивно впливає на швидкість завантаження та відображення 3D-контенту. Завдяки сумісності з поширеним стандартом glTF, Draco легко інтегрується в різні веб-застосунки, спрощуючи процес розробки. Крім того, Draco сприяє зменшенню часу відгуку веб-застосунків за рахунок зниження навантаження на процесор, що особливо важливо для пристроїв з обмеженими ресурсами. В результаті використання Draco дозволяє створювати інтерактивні та візуально привабливі веб-застосунки, які можуть працювати на широкому спектрі пристроїв і в різних мережевих умовах.

## 2.4 Алгоритми стиснення геометрії Draco

Алгоритми стиснення геометрії — це математичні та програмні методи, які дозволяють зменшити обсяг даних, що описують тривимірні об’єкти, без суттєвого погіршення їхньої структури чи візуальної якості. Такі алгоритми використовуються для оптимізації зберігання та передачі 3D-моделей у різних застосунках, зокрема в іграх, віртуальній реальності, доповненій реальності та веб-застосунках.

М. Фінлі та Т. Белл у своїй роботі [21] вказують, що у випадку 3D-геометрії компресія зазвичай включає:

- стиснення геометричних координат (зменшення кількості інформації, яка описує положення вершин у просторі);

- стиснення топології (оптимізація даних, що описують зв’язки між вершинами, гранями та іншими елементами сітки);
- стиснення атрибутів (компресія додаткових даних, таких як кольори, текстурні координати, нормалі тощо).

Метою цих алгоритмів є знаходження балансу між ступенем стиснення та точністю відновлення оригінальної моделі під час декомпресії. Ефективні алгоритми забезпечують високу щільність даних із мінімальними втратами, що особливо важливо для передачі моделей через мережу.

Draco використовує спеціалізовані підходи, такі як Sequential і Edgebreaker, для стиснення геометрії, дозволяючи суттєво скоротити обсяг даних при збереженні точності та якості 3D-моделей.

### 2.4.1 Алгоритм Sequential

Sequential — це метод обробки та стиснення геометрії 3D-моделей, який використовується у бібліотеці Draco. Sequential обробляє геометрію 3D-моделей у заздалегідь визначеному порядку, зазвичай послідовно обробляючи вершини та грані. Основна ідея цього алгоритму полягає у використанні прогнозного кодування. За цим підходом положення кожної вершини прогнозується на основі координат попередньо закодованих вершин. Замість зберігання абсолютних значень координат зберігається залишок — тобто різниця між прогнозованим і фактичним положенням. Завдяки цьому обсяг необхідних для передачі або зберігання даних суттєво зменшується, особливо у випадках, коли модель має плавні поверхні.

Як зазначають І. Дешко та В. Цвєтков у своїй роботі [22], Sequential демонструє високу ефективність у стисненні завдяки тому, що координати вершин зазвичай мають значну кореляцію. Ця властивість забезпечує менші похибки прогнозу та, відповідно, знижує обсяг залишкових значень, які підлягають кодуванню.

Крім того, алгоритм добре адаптується до різних форматів геометрії. У випадках, коли необхідно досягти максимальної компресії, він підтримує режим стиснення з регульованим рівнем втрат. Це дозволяє налаштовувати баланс між якістю відображення моделі та її розміром, що є особливо корисним у застосунках з обмеженими ресурсами, таких як мобільні або вебплатформи.

Sequential також може інтегрувати додаткові методи оптимізації, наприклад, кластеризацію вершин, що дає змогу підвищити точність прогнозування та зменшити залишкові похибки, що покращує загальну ефективність алгоритму. Як відзначають дослідники, наступними кроками у розвитку алгоритму можуть стати адаптація до складних топологій та розширення можливостей для роботи з багатокомпонентними 3D-об’єктами.

### 2.4.2 Алгоритм Edgebreaker

Алгоритм Edgebreaker спрямований на кодування зв’язності трикутної сітки за допомогою компактної послідовності символів. Топологія сітки, тобто інформація про те, як трикутники з’єднані один з одним, представляється п’ятьма символами: C (з’єднання); L (ліворуч); E (кінець); R (праворуч); S (розділ).

За словами Дж. Россіньяка [23], автора цього алгоритму, середня довжина коду складає лише два біти на трикутник, що забезпечує високий ступінь стискання.

Ключова ідея полягає у послідовному обході трикутників сітки та записі їх зв’язків у вигляді символів. Завдяки передбачуваності зв’язків між трикутниками у більшості 3D-моделей алгоритм значно зменшує обсяг необхідної для зберігання інформації.

Алгоритм також оптимізований для швидкого декодування: записана послідовність символів дозволяє ефективно відновлювати топологію сітки. Алгоритм також дозволяє суттєво зменшити обсяг топологічної інформації,

необхідної для передачі або зберігання. Edgebreaker добре працює із трикутними сітками різного типу, включаючи ті, які містять отвори або складні топологічні особливості.

У бібліотеці Draco алгоритм Edgebreaker використовується для стискання топології трикутних сіток у поєднанні з іншими методами компресії, такими як прогнозне кодування координат вершин. Це дозволяє забезпечити комплексну оптимізацію 3D-моделей, охоплюючи як геометрію, так і топологію. Завдяки цьому підходу Draco дозволяє суттєво зменшити розмір 3D-файлів без втрати якості.

### 2.4.3 Порівняння алгоритмів бібліотеки Draco

В рамках бібліотеки Draco алгоритми Sequential та Edgebreaker є ключовими методами стиснення геометричних 3D-моделей. Алгоритм Sequential фокусується на оптимізації представлення координат вершин, тоді як Edgebreaker розроблений для ефективного кодування топологічних відношень між трикутниками. Обидва підходи є невід'ємною частиною загальної стратегії стиснення 3D-даних, кожен з яких вносить свій унікальний внесок у досягнення високого ступеня компресії.

Алгоритми Sequential та Edgebreaker пропонують різні підходи до стиснення 3D-моделей. Sequential фокусується на оптимізації представлення геометричних даних за допомогою прогнозного кодування, тоді як Edgebreaker забезпечує ефективне кодування топологічної інформації. Використання п'ятисимвольного алфавіту в Edgebreaker дозволяє досягти середньої довжини коду в два біти на трикутник, що є значно менше, ніж при використанні традиційних методів. Однак, ефективність Sequential безпосередньо залежить від локальних характеристик моделі, тоді як Edgebreaker забезпечує більш стабільні результати для різних типів моделей. Основні відмінності ми також відобразили у таблиці 2.

Таблиця 2 — Порівняння алгоритмів бібліотеки Draco

| Характеристика | Sequential | Edgebreaker |
|---|---|---|
| Основна мета | Стиснення геометрії | Стиснення топології |
| Методологія | Прогнозне кодування координат вершин | Кодування зв’язності трикутних сіток |
| Формат даних | Координати вершин | Трикутна сітка |
| Середній обсяг даних | Залежить від кореляції координат | 2 біти на трикутник |
| Адаптація до складних топологій | Потребує додаткових методів | Добре працює з отворами та складними топологіями |
| Швидкість декодування | Висока | Дуже висока |
| Застосування в Draco | Компресія геометрії | Компресія топології у трикутних сітках |

Таким чином, бібліотека Draco пропонує комплексне рішення для стиснення 3D-моделей, засноване на синергії алгоритмів Sequential та Edgebreaker. Алгоритм Sequential забезпечує ефективне стиснення геометричних атрибутів моделі за допомогою прогнозного кодування, тоді як алгоритм Edgebreaker оптимізує представлення топологічних відношень між елементами моделі. Комбіноване застосування цих методів дозволяє досягти високого ступеня компресії без суттєвої втрати якості геометрії та топології.

## 2.5 Основні параметри стиснення тривимірних моделей в бібліотеці Draco

Параметри стиснення тривимірних моделей — це сукупність налаштувань, що визначають алгоритми та рівень компресії даних, які описують геометричні, матеріальні та топологічні властивості тривимірного об’єкта. Мета стиснення полягає в оптимізації розміру файлу без суттєвої втрати візуальної

якості, необхідної для подальшого використання моделі в інтерактивних середовищах, візуалізації чи 3D-друку.

Як зазначають А. Бутсі, Х. Іоаннідіс та С. Верикоку [1] у бібліотеці Draco стиснення досягається за допомогою трьох основних параметрів: квантизації позицій**,** квантизації нормалей та квантизації кольорів**.**

1. Квантизація позицій — це параметр, що визначає точність координат вершин 3D-моделі. Менша кількість рівнів дискретизації знижує розмір файлу, але може спричинити втрату геометричних деталей.

2. Квантизація нормалей — це параметр для стиснення орієнтації поверхонь моделі, що впливає на освітлення та текстурування. Їхня квантизація дозволяє зберегти важливу інформацію про напрямок поверхонь при мінімальних візуальних змінах.

3. Квантизація кольорів — це параметр, що застосовується до даних кольору та текстури моделі. Він дозволяє зменшити розмір файлу шляхом зниження кількості бітів, виділених на кожен колірний канал (R, G, B, A).

Усі три параметри налаштовуються залежно від конкретних вимог до моделі.

### 2.5.1 Квантизація позицій як параметр стиснення у бібліотеці Draco

Квантизація позицій є ключовим параметром алгоритму стиснення тривимірних моделей у форматі Draco, що дозволяє знизити бітову глибину представлення координат вершин. Шляхом обмеження точності координат, які зазвичай кодуються 32-бітними числами з плаваючою комою, алгоритм досягає значного зменшення розміру файлу. Квантизація полягає у дискретизації неперервного простору координат, що еквівалентно мапуванню дійсних чисел на дискретний набір значень, визначаючи таким чином рівень деталізації геометрії моделі.

Як вказують Д. Кім, К. Ю та Н. Квак [24], принцип роботи квантизації полягає в дискретизації простору, у якому розташована модель. Спочатку ви-

значається діапазон координат [min, max], який займає модель. Потім цей діапазон поділяється на кількість рівнів, що залежить від заданої кількості бітів. Наприклад, при використанні 10 бітів діапазон розбивається на $2^{10} = 1024$ рівні, а координати кожної вершини округлюються до найближчого значення серед цих рівнів. У результаті втрачається частина точності, проте розмір файлу значно зменшується.

Математично процес квантизації описується формулою:

$$quantized_value = round\left(\frac{coordinate-min}{max-min} \times (levels - 1)\right),$$

де levels — кількість рівнів дискретизації, що дорівнює $2^{bits}$, а bits — кількість бітів, виділених для зберігання координат. Це значення згодом відновлюється у реальному просторі під час декодування.

Вибір кількості бітів є ключовим моментом, адже він визначає компроміс між розміром файлу та точністю геометрії. Наприклад, 10-бітна квантизація підходить для великих об’єктів або низькодеталізованих моделей, тоді як 14 біт забезпечують високу точність і мінімальні спотворення, зберігаючи при цьому значну економію обсягу даних. Якщо ж використовується 16 і більше бітів, модель майже не відрізняється від оригіналу, але це зменшує ефективність стиснення.

Недостатня кількість бітів може призвести до візуальних спотворень, таких як «сходинки» на поверхнях або розриви в місцях, де вершини повинні з'єднуватися. Однак правильно підібрана кількість рівнів забезпечує високу візуальну якість без зайвого навантаження на обчислювальні ресурси. Для більшості задач у веб-розробці або мобільних застосунках 12—14 біт є оптимальним вибором.

У бібліотеці Draco квантизація позицій налаштовується за допомогою параметра $-qp$. Наприклад, команда [$draco_encoder -i\ input.obj -o\ output.drc\ -qp$ 12] встановлює 12-бітну квантизацію для координат позицій, що дозволяє суттєво зменшити розмір файлу з мінімальною втратою якості, що особливо важливо при передачі моделей через мережу.

### 2.5.2 Квантизація нормалей як параметр стиснення у бібліотеці Draco

Квантизація нормалей є параметром у процесі стиснення тривимірних моделей за допомогою бібліотеки Draco. Вона дозволяє суттєво зменшити бітову глибину представлення нормальних векторів, які визначають орієнтацію поверхонь моделі у просторі. Оскільки нормалі є одиничними векторами, їх координати обмежені одиничною сферою, що надає можливість застосовувати спеціальні методи квантизації, оптимізовані для представлення даних на сфері

М. Дірінг [25] пояснює, що нормалі зазвичай представлені тривимірними координатами $(x, y, z)$ у плаваючій точці. Квантизація нормалей передбачає округлення їх значень до дискретних рівнів на одиничній сфері. Це досягається шляхом перетворення нормалей у сферичну координатну систему $(\phi, \theta)$, де $\phi$ — азимутальний кут, а $\theta$ — полярний кут. Потім ці кути дискретизуються залежно від заданої кількості рівнів, після чого нормалі реконструюються на основі нових значень.

У Draco квантизація нормалей налаштовується через кількість бітів, виділених на їх зберігання. Більша кількість бітів забезпечує вищу точність орієнтації, тоді як менша — значно зменшує розмір моделі. Наприклад, 6 бітів — це мінімальний рівень, який може створювати помітні артефакти в освітленні, 10 бітів забезпечує достатню точність для більшості застосунків, а 16 бітів і більше — це практично ідентична точність оригіналу.

Для управління квантизацією нормалей у Draco використовується параметр $-qn$ (quantization for normals). Команда стиснення з квантизацією нормалей може виглядати так: [$draco_encoder\ -i\ input.obj\ -o\ output.drc\ -qn\ 10$].

Для зменшення похибок квантизації нормальних векторів бібліотека Draco використовує метод проектування на одиничну сферу з дискретним набором точок. Такий підхід забезпечує рівномірний розподіл квантів на сфері, що є особливо важливим для складних геометричних об’єктів. Дискретизація

кутів дозволяє оптимізувати баланс між точністю представлення нормалей та обчислювальною складністю.

### 2.5.3 Квантизація кольорів як параметр стиснення у бібліотеці Draco

Квантизація кольорів — це параметр для оптимізації розміру 3D-моделей у бібліотеці Draco, що полягає у зменшенні обсягу даних, необхідних для зберігання колірної інформації. У 3D-моделях кольори часто застосовуються для текстурування, візуалізації матеріалів або як частина візуального дизайну. Вони зазвичай представлені у форматі RGB або RGBA, де кожен канал (червоний, зелений, синій, альфа) використовує 8 бітів для зберігання значень, що забезпечує до 256 градацій на канал. Це є запорукою високої точності, але значно збільшує розмір файлу.

М. Перес-Дельгадо та М. Селебі [26] вказують, що квантизація кольорів працює за принципом зменшення кількості бітів, виділених на кожен колірний канал. Наприклад, якщо замість 8 бітів використовується 4 біти, то кількість можливих значень для кожного каналу зменшується до 16, що суттєво скорочує загальний обсяг даних. Це означає, що кольори округлюються до найближчого значення у новому, менш деталізованому колірному просторі.

Бібліотека Draco дозволяє налаштовувати квантизацію кольорів за допомогою параметра $-qc$ (quantization for colors). Значення цього параметра визначає кількість бітів, що виділяються на кожен канал таким чином, що 8 бітів зберігає вихідну якість кольорів, 6 бітів виявляє вже деяке зменшення точності з мінімально помітними змінами для людського ока, 4 біти вказує вже на значне скорочення обсягу даних, і може спричинити втрату плавності градацій. Команда для встановлення рівня квантизації кольорів виглядає так:

[$draco_encoder\ -i\ input.obj\ -o\ output.drc\ -qc$ 6]

Квантизація кольорів дозволяє суттєво зменшити розмір файлу, однак ступінь втрати точності залежить від кількості використаних бітів. Для моде-

лей із невеликою кількістю унікальних кольорів (наприклад, технічні креслення) можна використовувати низькі значення $-qc$. У випадках, коли модель має детальні текстури або градієнти, необхідно застосовувати більшу кількість бітів, щоб уникнути помітної деградації кольорів.

Основним недоліком квантизації кольорів є те, що зменшення бітової глибини представлення кольорів шляхом квантизації неминуче призводить до втрати частини візуальної інформації, що проявляється у вигляді появи візуальних артефактів. Однак шляхом підбору оптимальних параметрів квантизатора можна досягти компромісу між ступенем стиснення та якістю відновленого зображення.

## 2.6 Формати збереження тривимірної графіки бібліотеки Draco

Формати збереження тривимірної графіки — це стандарти, які використовуються для зберігання, передачі та обробки тривимірних даних у цифровому форматі. Вони дозволяють зберігати інформацію про геометрію, текстури, матеріали, освітлення та інші атрибути тривимірних моделей. Кожен формат має свої особливості, переваги та недоліки, що робить їх придатними для різних застосунків і використання в різних галузях.

Бібліотека Draco підтримує стиснення даних тривимірних моделей, інтегруючись із різними форматами. Серед найбільш популярних форматів, які забезпечують ефективне використання можливостей Draco, виділяються glTF (GL Transmission Format), PLY (Polygon File Format) та OBJ (Wavefront Object). У навчальному посібнику [27] надаються визначення цих форматів збереження тривимірної графіки:

- glTF — це сучасний стандарт передачі 3D-даних, оптимізований для стиснення геометрії й текстур із підтримкою бібліотеки Draco. Цей формат дозволяє стискати як окремі компоненти сітки (позиції, нормалі, текстурні координати), так і анімації, що робить його ідеальним для інтерактивних веб- і мобільних застосунків.

- PLY — це формат, який здебільшого використовується для зберігання хмар точок та інших точних тривимірних даних. Хоча PLY не має нативної підтримки стиснення, його дані можуть бути стиснуті Draco шляхом перетворення у сумісні структури. Це дозволяє значно зменшити розмір файлів, які використовуються у високоточних наукових і промислових застосунках.
- OBJ — це один із найпоширеніших форматів для обміну тривимірними моделями, особливо для статичних сіток. Стиснення OBJ через Draco зазвичай реалізується через конвертацію у формат glTF, після чого застосовується компресія геометрії та топології. Цей підхід ефективний для роботи з великими бібліотеками моделей.

Використання бібліотеки Draco разом із glTF, PLY та OBJ забезпечує широкий спектр можливостей для оптимізації зберігання й передачі тривимірних даних.

### 2.6.1 Формат збереження тривимірної графіки glTF

Формат glTF (GL Transmission Format), розроблений консорціумом Khronos Group, є стандартом для передачі 3D-активів, оптимізованих для ефективного зберігання, завантаження та передачі. Його ключовою метою є мінімізація розміру файлів без суттєвих втрат якості. В. Лемуан і М. Війнантс [28] підкреслюють, що головною перевагою цього формату є підтримка різноманітних методів стиснення, які охоплюють як геометрію, так і текстури, а також додаткові атрибути.

Стиснення геометрії в glTF реалізується через інтеграцію бібліотеки Draco, яка дозволяє ефективно стискати позиції вершин, нормалі, текстурні координати, кольори та індекси сіток. Використання Draco дозволяє зменшити розмір даних у кілька разів завдяки комбінуванню технік квантування та прогнозування. У glTF-файлах інформація про стиснення Draco зберігається у JSON-структурі, яка описує, які саме дані було стиснуто, що забезпечує швидке декодування клієнтськими бібліотеками, такими як WebGL. Це сприяє

збереженню якості візуалізації при значному зменшенні обсягу переданих даних.

Текстури в glTF також підтримують багаторівневе стиснення, яке включає популярні формати JPEG і PNG, що забезпечують базову компресію. Для сучасних задач часто використовують формат Basis Universal (KTX2), який підтримує режими ефективного втратного стиснення (ETC1S) і високоякісного компресування (UASTC). Цей формат є надзвичайно ефективним для зменшення розміру текстур із підтримкою апаратного декодування та інтеграції різнорівневих текстурних карт (міп-мапів). Використання стиснених текстур дозволяє економити пам’ять, прискорює завантаження 3D-моделей і забезпечує візуальну деталізацію навіть за обмеженого обсягу ресурсів.

Окрім цього, glTF передбачає механізми стиснення анімацій, які базуються на квантилюванні ключових кадрів та предикативному кодуванні. Ці методи дозволяють оптимізувати зберігання анімаційних даних без суттєвої втрати плавності руху.

Формат glTF існує у кількох варіантах. Найбільш поширеним є glTF (.gltf +.bin), де JSON-файл описує структуру моделі, а геометрія та текстури зберігаються у окремих бінарних файлах. Іншою формою є glTF-бінарний (.glb), який об’єднує JSON, геометрію та текстури в єдиному файлі, що спрощує передачу. Використання стиснення Draco інтегрується в будь-який із цих форматів, роблячи їх ще більш компактними та зручними для зберігання.

Інструменти для оптимізації glTF-моделей включають glTF-Pipeline для автоматизації процесу стиснення, плагіни для Blender, які дозволяють експортувати стислий glTF, та інші утиліти для роботи з текстурами, такі як конвертери Basis або KTX2-компресори. Завдяки цьому екосистема glTF забезпечує гнучкість і адаптивність до різних технічних вимог.

### 2.6.2 Формат збереження тривимірної графіки PLY

Формат PLY (Polygon File Format), також відомий як Stanford Triangle Format, був розроблений для зберігання тривимірних даних, зокрема полігональних моделей, з акцентом на їхню простоту і гнучкість. Його основне призначення — забезпечення ефективного зберігання геометричної та атрибутивної інформації, яка використовується для опису 3D-об’єктів. PLY підтримує як прості трикутникові сітки, так і хмари точок, з можливістю додавання довільної кількості метаінформації у вигляді властивостей вершин або граней.

Стиснення у форматі PLY реалізується в основному через оптимізацію представлення геометрії. Дані можуть зберігатися в текстовому або бінарному вигляді, що безпосередньо впливає на їхній обсяг. Дж. Джонстон [29] зазначає, що текстовий формат є зручним для читання та редагування, однак значно збільшує розмір файлу, тоді як бінарний формат зменшує обсяг даних і прискорює завантаження. У текстовому варіанті точність числових значень може бути обмежена кількістю знаків після коми, що дозволяє зменшити розмір, але може призводити до втрат точності. Бінарні PLY-файли забезпечують збереження повної точності представлення даних, використовуючи менший обсяг пам’яті.

Особливістю формату є підтримка довільних атрибутів, що дозволяє зберігати додаткові властивості, такі як кольори, нормалі, текстурні координати або інші характеристики вершин і граней, але його пряме використання з алгоритмом стиснення Draco обмежене. Для цього потрібна попередня конвертація даних.

Щодо текстур, PLY не має інтегрованих механізмів для їх зберігання, але підтримує вказівку текстурних координат у вигляді атрибутів вершин. Це дозволяє використовувати текстури, збережені окремо, однак для стиснення таких даних зазвичай застосовуються зовнішні формати, як-от JPEG, PNG або KTX2, залежно від конкретного сценарію використання.

Анімаційні дані, на відміну від таких форматів, як glTF, не є частиною специфікації PLY. Це обмежує застосування формату переважно статичними

моделями або сканованими хмарами точок, які не потребують динамічного відображення. Проте завдяки його простоті та підтримці точних геометричних даних формат широко використовується в комп'ютерному баченні, зокрема для роботи з лазерним скануванням, реконструкцією поверхонь та іншими задачами, що вимагають збереження максимальної точності вихідних даних.

Формат PLY має два основних варіанти зберігання — ASCII та бінарний. ASCII забезпечує легку читабельність і можливість редагування, однак значно збільшує розмір файлів. Бінарний формат, у свою чергу, компактний і придатний для зберігання великих обсягів даних, але вимагає спеціалізованого програмного забезпечення для перегляду та редагування.

Незважаючи на відсутність власних алгоритмів стиснення, формат файлів PLY залишається широко використовуваним у наукових дослідженнях та інженерних розробках завдяки своїй гнучкості у визначенні атрибутів вершин та граней, що дозволяє зберігати детальну геометричну інформацію. Простота структури формату сприяє його легкій обробці та інтеграції в різноманітні програмні середовища. Однак, для досягнення високих ступенів стиснення даних, файли PLY часто потребують додаткової обробки та конвертації в інші, більш спеціалізовані формати.

### 2.6.3 Формат збереження тривимірної графіки OBJ

Формат OBJ (Wavefront Object) є одним із найпоширеніших стандартів для зберігання тривимірних моделей. Е. Байрактар, Ю. Ван та А. ДельБ'ю [30] пояснюють, що цей формат забезпечує збереження геометричних даних: вершин, нормалей, текстурних координат, а також описує топологію поверхонь через полігони, зазвичай трикутники або чотирикутники. Попри свою простоту і широке використання, цей формат сам по собі не включає вбудованих механізмів стиснення, однак підтримує кілька підходів до оптимізації, які дозволяють зменшити розмір файлів без втрати функціональності.

Стиснення геометричних даних у форматі OBJ реалізується шляхом застосування зовнішніх алгоритмів і технік оптимізації. Одним із найбільш поширених методів є квантування даних, яке дозволяє зменшити точність чисел до потрібного рівня, зберігаючи при цьому візуальну якість моделі. Наприклад, зменшення кількості десяткових знаків у координатах вершин може істотно скоротити розмір текстових файлів OBJ, оскільки кожна координата зберігається у вигляді тексту. Крім того, дублікати вершин і граней можуть бути видалені або об’єднані за допомогою інструментів оптимізації сіток, що зменшує обсяг даних і підвищує ефективність зберігання.

Бінаризація є ще одним підходом до стиснення OBJ. Хоча сам формат є текстовим і орієнтованим на читабельність, його можна конвертувати у бінарний формат за допомогою сторонніх утиліт. Це значно скорочує розмір файлів і покращує швидкість завантаження, однак робить файли менш зручними для ручного редагування.

Для текстур, які часто супроводжують OBJ-файли, стиснення здійснюється через формати зображень, наприклад, JPEG, PNG або KTX2. OBJ дозволяє включати посилання на зовнішні файли текстур у своїх матеріалах через файли MTL (Material Template Library). Використання сучасних методів стиснення текстур, дозволяє значно зменшити обсяг переданих даних, особливо для моделей з великими текстурними картами.

Попри свою простоту, OBJ не підтримує механізмів стиснення анімаційних даних, оскільки вони не є частиною його специфікації. Це обмежує його застосування переважно статичними моделями. Для стискання складніших структур використовуються інші формати, наприклад, glTF.

OBJ залишається популярним завдяки своїй універсальності і простоті, але для зменшення розміру файлів і підвищення ефективності він зазвичай використовується у поєднанні з зовнішніми інструментами. Вони дозволяють зменшити кількість вершин, застосовувати алгоритми спрощення сіток, вида-

ляти зайві дані і здійснювати бінаризацію. Крім того, для зберігання та стиснення геометрії OBJ-файли можуть бути інтегровані з бібліотеками на зразок Draco, хоча це вимагає попередньої конвертації у відповідні формати.

## 2.7 Висновки до розділу 2

1. Для зменшення обсягу тривимірних даних використовують методи з втратами та без втрат**.** Перші спрощують модель, знижуючи деталізацію та колірну глибину, що дозволяє досягти значної компресії, але може вплинути на візуальну якість. Другі зберігають геометрію без втрат, але забезпечують меншу компресію.

2. Вибір інструмента для оптимізації тривимірних моделей обумовлений необхідністю виконання певних функцій. Порівняння цих методів показало, що бібліотека Draco є універсальним інструментом для оптимізації тривимірних моделей у веб-застосунках. Застосування технології Draco сприяє значному скороченню часу завантаження моделей та зниженню вимог до обчислювальних ресурсів, забезпечуючи швидке завантаження моделей навіть за умов низької швидкості інтернету.

3. Алгоритми Sequential та Edgebreaker у бібліотеці Draco забезпечують ефективне стиснення 3D моделей. Sequential оптимізує координати вершин, а Edgebreaker мінімізує дані про структуру сітки. Завдяки цьому досягається високий ступінь стиснення без втрати якості.

4. Стиснення тривимірних моделей у бібліотеці Draco базується на квантизації позицій, нормалей та кольорів. Квантизація позицій знижує точність координат вершин, нормалей — орієнтацію поверхонь, а кольорів — деталізацію колірної інформації, забезпечуючи баланс між ступенем компресії та візуальними вимогами.

5. Draco підтримує три основні формати збереження тривимірної графіки — glTF, PLY та OBJ. Вони забезпечують широкий спектр можливостей для роботи з 3D-даними, кожен із них має свої унікальні переваги. Інтеграція

цих форматів із бібліотекою Draco розширює їхні можливості, дозволяючи ефективно зменшувати розмір файлів і забезпечувати якісну передачу даних для різноманітних сфер застосування.

# 3. РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ОТРИМАННЯ СТАТИСТИКИ ВИКОРИСТАННЯ DRACO

## 3.1 Формування вимог для майбутньої програми

Розробка комп'ютерної системи для отримання статистичних даних про ефективність стиснення Draco вимагає чіткого формулювання вимог. Програма має забезпечувати ефективний аналіз процесів кодування та декодування, підтримувати різні конфігурації та забезпечувати точні результати порівняльного аналізу.

Однією з головних цілей є полегшення як кодування, так і декодування 3D-моделей за допомогою бібліотеки Draco, дозволяючи при цьому налаштовувати параметри стиснення, такі як рівні квантування для позицій вершин, координат текстури та нормалей. Він також має дозволяти користувачам перемикатися між двома основними методами стиснення, які пропонує Draco — EdgeBreaker і Sequential. Така гнучкість забезпечує комплексний аналіз можливостей бібліотеки.

Щоб зробити систему практичною та надійною, вона повинна обробляти синтетичні 3D-дані. Синтетичні моделі мають бути створені з різною складністю, включаючи різну кількість вершин, індекси та атрибути. У реальних сценаріях система повинна обробляти широко використовувані 3D-формати, забезпечуючи значущу оцінку продуктивності в реалістичних умовах.

Програма має містити модуль порівняльного аналізу, який вимірює час виконання для кодування та декодування, оцінює коефіцієнти стиснення та аналізує співвідношення між якістю та швидкістю стиснення. Ці результати мають бути експортовані в структурованих форматах, таких як CSV, щоб полегшити зовнішній аналіз і візуалізацію.

Важливою метою проектування є забезпечення модульності в архітектурі системи. Кожна основна функція, така як кодування, декодування, порівняльний аналіз і експорт результатів, повинна бути інкапсульована в своєму

модулі для спрощення розробки, обслуговування та майбутніх оновлень. Крім того, система повинна бути доступна через інтерфейс командного рядка (CLI), який дозволяє користувачам вказувати параметри та виконувати завдання без використання графічного інтерфейсу.

## 3.2 Огляд архітектури комп'ютерної системи

### 3.2.1 Вибір платформи для розробки

Платформа для розробки системи була обрана на основі таких критеріїв, як сумісність з бібліотекою Draco, кросплатформенність і простота розробки. JavaScript і TypeScript були обрані як основні мови через їх широке використання, сильну підтримку асинхронних операцій та інтеграцію з модулями Draco Encoder і Decoder. Ці мови також надають доступ до сучасних інструментів для створення інтерфейсів командного рядка та обробки 3D-даних.

Крім того, проект включає бібліотеку draco3d для стиснення та декомпресії, а також допоміжні бібліотеки, такі як json-2-csv для експорту результатів і аргументи командного рядка для аналізу параметрів. Ця комбінація гарантує, що система є водночас легкою та універсальною, що робить її придатною для досліджень і практичних застосувань.

### 3.2.2 Визначення алгоритму роботи програми

Програма працює за структурованим алгоритмом, розробленим для забезпечення надійної обробки даних і точних вимірювань продуктивності. Ключові кроки алгоритму такі:

1. Ініціалізація. Програма починається з аналізу заданих користувачем параметрів через інтерфейс командного рядка. Значення за замовчуванням використовуються для параметрів, які не встановлені явно.

2. Генерація синтетичних даних. 3D-моделі генеруються на основі вхідних параметрів, включаючи різну кількість вершин, індексів і атрибутів. Крім того, можна завантажити моделі, надані користувачем.

3. Компресія. Кожна модель стискається за допомогою модуля кодування Draco із застосуванням вибраного методу (EdgeBreaker або Sequential) і параметрів квантування. Розмір стисненого виходу та тривалість стиснення записуються.

4. Декомпресія. Стиснуті дані декодуються за допомогою модуля Draco Decoder для забезпечення цілісності та вимірювання часу декомпресії.

5. Порівняльний аналіз. Програма повторює кроки стиснення та розпакування для всіх тестів, зберігаючи результати кожної ітерації. Розраховуються такі показники, як час виконання та ступінь стиснення.

6. Експорт результатів. Зведені результати форматуються та експортуються у файл CSV, що забезпечує подальший аналіз. Також програма формує графіки на кожен з наборів тестів, які можна переглянути за посиланням з консолі у браузері. Ще під час виконання програми в консоль виводиться стан проходження тесту, за яким можна приблизно відстежувати коли програма закінчить виконання, та після проходження набору тестів формується табличка з попередніми результатами яка також записується у консоль.

7. Обробка помилок. Програма відстежує винятки під час виконання, забезпечуючи реєстрацію помилок і інформування користувача про будь-які проблеми.

Алгоритм успішно вирішує поставлене завдання, надаючи об’єктивні статистичні дані щодо продуктивності алгоритму Draco, що відповідають загальноприйнятим стандартам оцінки. Результати роботи алгоритму підтверджують його відповідність функціональним та технічним вимогам, зазначеним у технічному завданні.

### 3.3 Реалізація комп’ютерної системи

Реалізація комп’ютерної системи передбачала створення конкретних алгоритмів і модулів, що відповідають сформульованим вимогам. Вони включають створення 3D-моделей для тестування, вимірювання ключових параметрів

під час стиснення та декомпресії та виведення результатів для подальшого аналізу.

### 3.3.1 Планування файлової системи

Програма буде складатися з двох частин — це бенчмарк в якому будуть всі необхідні класи для тестування моделей з подальшим виводом результатів, та головний файл який буде використовувати цей бенчмарк: динамічно створювати 3D моделі за різними параметрами, записувати налаштування для тестування та для виводу результатів, запускати бенчмарк за цими налаштуваннями.

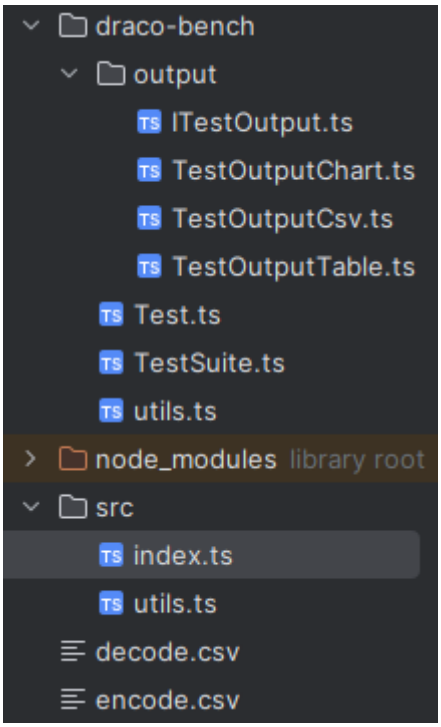



Рисунок 1 — Файлова структура програми

В папці draco-bench міститься логіка порівняльного аналізу та різні вихідні формати для результатів тестування:

- ITestOutput.ts (інтерфейс, що визначає структуру для реалізацій виводу).

- TestOutputChart.ts (забезпечує створення візуальних результатів у формі діаграм).
- TestOutputCsv.ts (створює файли CSV із результатами тестування для подальшого аналізу).
- TestOutputTable.ts (генерує таблиці для представлення результатів тестування).
- Test.ts (визначає логіку та життєвий цикл для окремих порівняльних тестів).
- TestSuite.ts (керує групами тестів і їх виконанням у єдиному наборі).
- utils.ts (містить службові функції для спільних операцій у модулі порівняльного аналізу).

Папка src містить основну програму, яка інтегрується з бібліотекою draco-bench, та службові функції в файлі utils.ts. Програма відповідає за:

1. Динамічне створення 3D-моделей із певними параметрами за кількістю вершин.
2. Налаштування параметрів для стиснення та для подальшого виводу.
3. Подача створених моделей з необхідними параметрами у TestSuite для проведення бенчмарку та порівняльного аналізу.

Така структура забезпечує чіткий поділ проблем: draco-bench, призначений для порівняльного аналізу логіки та результатів, а src зосереджується на основних функціях програми.

### 3.3.2 Реалізація класу TestSuite

Клас TestSuite керує виконанням тестів, керує їхнім життєвим циклом і агрегує результати. Він надає механізми для одночасного виконання, відстеження прогресу та гнучкого визначення тесту.

Лістинг 1 — Службові методу класу набору тестів TestSuite

```
public add(testOptions: Omit<TestOptions<TCustomProps>,
'suite'>) {
    if (this.running) return this;
    const test = new Test({ ...testOptions, suite: this });
    this.tests.push(test);
    if (testOptions.group) {
      this.testGroups.push(testOptions.group);
    }
    return this;
  }


  public addTest(test: Test<TCustomProps>) {
    if (this.running) return this;
    test.suite = this;
    this.tests.push(test);
    return this;
  }

  public addOutputGenerator(generator: ITestOutput) {
    this.outputGenerators.push(generator);
    return this;
  }

  public removeOutputGenerator(generator: ITestOutput) {
    const index = this.outputGenerators.indexOf(generator);
    if (index !== -1) this.outputGenerators.splice(index,
1);
    return this;
  }

  public output() {
    this.outputGenerators.forEach((generator) => genera-
tor.output(this, this.tests));
  }

  public testsCount() : number {
    return this.tests.length;
  }

  public get groups() {
    return this.testGroups;
  }
```

Дані методи з лістингу 1 забезпечують коректну роботи основної частини класу, в якої ці методи й використовуються.

Лістинг 2 — Основний метод запуску роботи класу набору тестів TestSuite

```
public async run() {
  this.running = true;
  this.done = false;
  const pool = new PromisePool();
  const bar = new SingleBar({
    hideCursor: true,
    format: `Process [{bar}] | {value}/{total} |
{duration_formatted},`,
    barCompleteChar: "=",
    barIncompleteChar: " ",
  });
  bar.start(this.tests.length * this.executeTimes, 0)
  const iterable: Iterable<number> = {
    [Symbol.iterator]: () => numberGenerator(this.exe-
cuteTimes),
  };
  await pool
    .withConcurrency(this.concurrency)
    .for(iterable)
    .onTaskFinished(_ => {
      bar.increment(this.tests.length);
    })
    .process(async (_, index, pool) => {
      for (let i = 0; i < this.tests.length; i++) {
        const test = this.tests[i];
        if (test.skip) return;
        try {
          const preReturn = test.preRun({});
          const runReturn = await test.run(preReturn);
          test.postRun(runReturn);
        } catch (err) {console.log(err);}
      }
    });
  this.done = true;
  this.running = false;
  bar.stop();
  console.log(kleur.green(" ✓"), ` Test suite
${kleur.bold(`"${this.name}"`)} completed`);
}
```

Даний клас являє собою модуль, що відповідає за організацію процесу тестування та надає інтерфейси для управління виконанням тестів, збору даних та формування звітів. Виконання тестування починається під час виклику команди run() з лістингу 2.

### 3.3.3 Реалізація класу Test

Клас Test з лістингу 3 представляє окремі тестові випадки, інкапсулюючи логіку для виконання та вимірювання продуктивності. Кожен тест відстежує свої показники виконання, включаючи тривалість і середній час, і забезпечує логіку до і після запуску.

Лістинг 3 — Клас Test

```
export interface TestOptions<TCustomProps extends TestCus-
tomPropsType> {
  name: string;
  description?: string;
  group?: string;
  suite: TestSuite<TCustomProps>;
  callback: TestFunctionType;
  callbackPre?: TestFunctionType;
  callbackPost?: TestFunctionType;
  customProps?: TCustomProps;
}
const TestOptionsDefaults: TestOptionsType<{}> = {
  name: '',
  description: '',
}
export type TestFunctionType = (test?: Test, chainParams?:
object) => Promise<object | void> | object | void;
type TestOptionsTypePure = 'callback' | 'callbackPre' |
'callbackPost' | 'suite'
type TestOptionsType<T extends TestCustomPropsType> =
Omit<TestOptions<T>, TestOptionsTypePure>;
export interface TestMetrics {
  duration: number;
  runs: number;
  average: number;
  skipped: number;
}
const TestMetricsDefaults = {
  duration: 0,
```

```
  runs: 0,
  average: 0,
  skipped: 0
}
export class Test<TCustomProps extends TestCustomPropsType
= any> implements TestOptionsType<TCustomProps> {
  public name!: string;
  public description!: string;
  public group?: string;
  public suite!: TestSuite<TCustomProps>;
  public skip: boolean = false;
  public customProps!: TCustomProps;
  protected callback!: TestFunctionType;
  protected callbackPre?: TestFunctionType;
  protected callbackPost?: TestFunctionType;
  protected metrics: TestMetrics;
  constructor(options: TestOptions<TCustomProps>) {
    this.metrics = { ...TestMetricsDefaults };
    const selfParams = { ...TestOptionsDefaults, ...options
};
    Object.assign(this, selfParams);
  }
  public run(chainParams: object): Promise<object> {
    return new Promise(async (resolve, reject) => {
      let result: object | void = {};
      try {
        const startAt = performance.now();
        result = await this.callback(this, chainParams);
        const endAt = performance.now();
        const testDuration = endAt - startAt;
        this.metrics.runs += 1;
        this.metrics.duration += testDuration;
        this.metrics.average += testDuration;

        if (this.metrics.runs > 1) this.metrics.average/=2;
      } catch (err) {
        this.skip = true;
        this.metrics.skipped = this.suite.executeTimes -
this.metrics.runs;
        reject(`Test ${this.name} execution failed, skip-
ping: ${err}`);
      }
      resolve(result ?? {});
    });
  }
  public preRun(chainParams: object) : object {
    if (this.callbackPre) return this.callbackPre(this,
chainParams) ?? {};
    return {}
```

```
  }

  public postRun(chainParams: object) : object {
    if (this.callbackPost) return this.callbackPost(this,
chainParams) ?? {};
    return {}
  }
  public getMetrics(): TestMetrics {
    return this.metrics;
  }

}
```

### 3.3.4 Утиліти для генерації 3D моделі

Файл utils.ts для головної логіки надає основні функції для створення синтетичних 3D-моделей і обробки тестових даних, код яких розташовано на лістингу 4. Ці утиліти генерують рандомізовані вершини, індекси та нормалі для імітації різноманітної складності моделі.

Лістинг 4 — Утилітарні функції для головного коду

```
export function vertexBytesLength(attributes: Rec-
ord<string, number>) : number {
  const keys = Object.keys(attributes);
  const attributeValues = keys.map((key) => attrib-
utes[key]);
  return attributeValues.reduce((former, current) => former
+ current);
}
export function generateVerticesBuffer(vertexCount: number,
attributes: Record<string, number> = { position: 3 }) :
Float32Array {
  if (!vertexCount || vertexCount % 3 !== 0) throw Er-
ror('generateRawVerticesBuffer - vertexCount must be deriv-
ative of 3');
  const vertices: Float32Array = new Float32Array(vertex-
Count * vertexBytesLength(attributes)).map(_ => getRandom-
Int(-500, 500) + Math.random());
  return vertices;
}
export function generateVertexIndices(vertexCount: number)
: Uint32Array {
  if (!vertexCount || vertexCount % 3 !== 0) throw Er-
ror('generateVertexIndices - vertexCount must be derivative
```

```
of 3');
  const indices = new Uint32Array(vertexCount);
  for (let i = 0; i < vertexCount; i++) indices[i] = i;
  return indices;
}
export function convertBytesToHumanReadable(bytes: number):
string {
  const sizes = ['Bytes', 'KB', 'MB', 'GB', 'TB'];
  if (bytes === 0) return '0 Byte';
  const i = parseInt(Math.floor(Math.log(bytes) /
Math.log(1024)).toString());
  return Math.round((bytes / Math.pow(1024, i)) * 100) /
100 + ' ' + sizes[i];
}
```

Цей код надає допоміжні функції, необхідні для створення та керування даними 3D моделей, а також для перетворення розмірів байтів у формати, зрозумілі людині. Ці утиліти є важливими для створення синтетичних даних та інтерпретації результатів тестів на стиснення та розтиснення.

### 3.3.5 Генерація виводу отриманих даних

Генератори вихідних даних форматують і зберігають результати тестів для аналізу. Інтерфейс ITestOutput і такі реалізації, як TestOutputCsv, обробляють структуровані звіти про результати. Вивід завдяки TestOutputCsv з лістингу 5 є найголовнішим в рамках цієї роботи, тому що зберігає інформацію всю інформацію в файлах таблиці, а не лише виводить проміжні результати в консолі в якості таблиці як TestOutputTable, або в якості графіку як TestOutputChart.

Лістинг 5 — Клас виводу результатів у csv файл

```
export class TestOutputCsv extends ITestOutput {
  protected header: Array<Record<string, string>> = [];
  protected rowGenerator!: (test: Test) => Record<string,
string | number | boolean> | undefined;
  protected filename!: string;
  constructor(
    filename: string,
    header: Array<Record<string, string>>,
```

```
    rowBuilder: (test: Test) => Record<string, string |
number | boolean> | undefined
  ) {
    super();
    this.filename = filename;
    this.header = header;
    this.rowGenerator = rowBuilder;
  }
  public output(suite: TestSuite<any>, tests: Test<any>[]):
void {
    const csvWriter = createObjectCsvWriter({
      append: true,
      path: this.filename,
      header: this.header
    });
    const records = new Array<Record<string, string | num-
ber | boolean>>();
    tests.forEach(test => {
      const row = this.rowGenerator(test);
      if (row) records.push(row);
    });
    csvWriter.writeRecords(records);
  }
}
```

TestOutputCsv дозволяє експортувати результати структурованих тестів, роблячи їх доступними для подальшого аналізу в таких інструментах, як електронні таблиці або бази даних. Він забезпечує гнучкість завдяки підтримці спеціальних заголовків і форматів рядків для конкретних потреб у звітах.

### 3.3.6 Логіка стиснення та декомпресії

Ця частина основної програми обробляє основні функції кодування та декодування за допомогою бібліотеки Draco. Він дозволяє налаштовувати такі параметри, як рівні квантування та методи стиснення.

Лістинг 6 — Функції компресії та декомпресії

```
export async function compression(
  mesh: { indices: Uint32Array; vertices: Float32Array;
texcoords: Float32Array },
  encoderModule: draco3d.EncoderModule,
  options: Record<string, any>
```

```
): Promise<number> {
  const encoder = new encoderModule.Encoder();
  const meshBuilder = new encoderModule.MeshBuilder();
  const dracoMesh = new encoderModule.Mesh();
  const numFaces = mesh.vertices.length / 9;
  const numPoints = mesh.vertices.length / 3;
  meshBuilder.AddFacesToMesh(dracoMesh, numFaces, mesh.in-
dices);
  mesh.vertices && meshBuilder.AddFloatAttribute(dracoMesh,
encoderModule.POSITION, numPoints, 3, mesh.vertices);
  mesh.texcoords && meshBuilder.AddFloatAttribute(dra-
coMesh, encoderModule.TEX_COORD, numPoints, 2, mesh.tex-
coords);
  encoder.SetSpeedOptions(options.encodeSpeed, options.de-
codeSpeed);
  encoder.SetAttributeQuantization(encoderModule.POSITION,
options.quantizePosition);
  encoder.SetAttributeQuantization(encoderModule.TEX_COORD,
options.quantizeTexcoord);
  encoder.SetEncodingMethod(options.method ===
"edgebreaker" ? encoderModule.MESH_EDGEBREAKER_ENCODING :
encoderModule.MESH_SEQUENTIAL_ENCODING);
  const encodedData = new encoderModule.DracoInt8Array();
  const encodedLen = encoder.EncodeMeshToDracoBuffer(dra-
coMesh, encodedData);
  encoderModule.destroy(dracoMesh);
  encoderModule.destroy(encoder);
  encoderModule.destroy(meshBuilder);
  encoderModule.destroy(encodedData);
  return encodedLen;
}
export async function decompression(
  outputData: Int8Array,
  decoderModule: draco3d.DecoderModule
): Promise<void> {
  const buffer = new decoderModule.DecoderBuffer();
  buffer.Init(outputData, outputData.length);
  const decoder = new decoderModule.Decoder();
  const geometryType = decoder.GetEncodedGeome-
tryType(buffer);
  const outputGeometry =
    geometryType === decoderModule.TRIANGULAR_MESH
      ? new decoderModule.Mesh()
      : new decoderModule.PointCloud();
  if (geometryType === decoderModule.TRIANGULAR_MESH) {
    decoder.DecodeBufferToMesh(buffer,
<Mesh>outputGeometry);
  } else {
```

```
    decoder.DecodeBufferToPointCloud(buffer, outputGeome-
try);
  }
  decoderModule.destroy(outputGeometry);
  decoderModule.destroy(decoder);
  decoderModule.destroy(buffer);
}
```

Функція compression, яка описана в лістингу 6, кодує 3D-сітку в стиснутий формат за допомогою бібліотеки Draco, а decompression з лістингу 6 декодує стиснуті дані назад у тривимірну геометрію (сітка або хмара точок). Вони дозволяють налаштовувати параметри, такі як:

Метод стиснення — EdgeBreaker або Sequential.

Рівні квантування — визначає компроміс між точністю та розміром.

Параметри швидкості — балансує швидкість кодування та якість стиснення.

Така гнучкість робить їх придатними для порівняльного аналізу та оптимізації робочих процесів обробки 3D-даних.

### 3.3.7 Створення набору тестів

Частина коду головної програми яка відповідає за наповнення набору тестів по заданих заздалегідь параметрах 3D моделей зображена на лістингу 7.

Лістинг 7 — Створення набору тестів в головному файлі

```
const testSuite = new TestSuite({
  name: 'Draco bench suite\n' +
    `quantize | position ${cliOptions.quantizePosition}, uv
${cliOptions.quantizeTexcoord}, generic
${cliOptions.quantizeGeneric}\n` +
    `speed | encode ${cliOptions.encodeSpeed}, decode
${cliOptions.decodeSpeed}  `,
  description: "Compression and decompression using
draco3d",
  executeTimes: 200,
  concurrency: 1
});
const encoderModule = await draco3d.createEncoderModule();
const decoderModule = await draco3d.createDecoderModule();
```

```
testSuite.add({ name: 'none', callback: () => {} });
tests.forEach(test => {
  testSuite
    .add({
      name: test.name,
      group: "encode",
      callback: () => compression(test.mesh, encoderMod-
ule),
      callbackPost: (t) => t.customProps.sizeEncoded =
test.mesh?.length,
      customProps: test.customProps
    })
    .add({
      name: test.name,
      group: "decode",
      callback: () => decompression(decoderModule),
      callbackPost: (t) => t.customProps.sizeDecoded = (
        (test.mesh.vertices.length * 4) +
        (test.mesh.texcoords.length * 4) +
        (test.mesh.indices.length * 4)),
      customProps: test.customProps
    })
})
```

Як показано на лістингу 8, заздалегідь повинні бути задані такі параметри як cliOptions які за замовчуванням використовують дані з optionDefinitions, та tests.

Лістинг 8 — Значення для стискання за замовчуванням

```
export const optionDefinitions: Array<OptionDefinition> = [
  { name: 'method', type: String, defaultValue:
'edgebreaker' },
  { name: 'encodeSpeed', type: Number, defaultValue: 0 },
  { name: 'decodeSpeed', type: Number, defaultValue: 10 },
  { name: 'quantizePosition', type: Number, defaultValue:
12 },
  { name: 'quantizeNormal', type: Number, defaultValue: 10
},
  { name: 'quantizeColor', type: Number, defaultValue: 8 },
  { name: 'quantizeTexcoord', type: Number, defaultValue:
12 },
  { name: 'quantizeGeneric', type: Number,defaultValue:12},
];
```

На лістингу 9 зображена структура як виглядають параметри 3D моделі для подальшого тестування. Ім’я задається залежно від кількості вершин, сітка формується завдяки утилітарної функції та вказується фактична кількість вершин окремим параметром.

Лістинг 9 — Перелік тестів для проходження

```
const tests = [
  { name: `${formatBigNumber(3333 * 3)} vert`, mesh:
generateMash(3333), customProps: { vertices: 3333 * 3 } },
// 10k or 9 999
  { name: `${formatBigNumber(5000 * 3)} vert`, mesh:
generateMash(5000), customProps: { vertices: 5000 * 3 } },
// 15k or 15 000
  { name: `${formatBigNumber(6666 * 3)} vert`, mesh:
generateMash(6666), customProps: { vertices: 6666 * 3 } },
// 20k or 19 998
  { name: `${formatBigNumber(8333 * 3)} vert`, mesh:
generateMash(8333), customProps: { vertices: 8333 * 3 } },
// 25k or 24 999
  { name: `${formatBigNumber(10000 * 3)} vert`, mesh:
generateMash(10000), customProps: { vertices: 10000 * 3 }
}, // 30k or 30 000
  { name: `${formatBigNumber(11666 * 3)} vert`, mesh:
generateMash(11666), customProps: { vertices: 11666 * 3 }
}, // 35k or 34 998
  { name: `${formatBigNumber(13333 * 3)} vert`, mesh:
generateMash(13333), customProps: { vertices: 13333 * 3 }
}, // 40k or 39 999
  { name: `${formatBigNumber(15000 * 3)} vert`, mesh:
generateMash(15000), customProps: { vertices: 15000 * 3 }
}, // 45k or 45 000
  { name: `${formatBigNumber(16666 * 3)} vert`, mesh:
generateMash(16666), customProps: { vertices: 16666 * 3 }
}, // 50k or 39 998
  { name: `${formatBigNumber(18333 * 3)} vert`, mesh:
generateMash(18333), customProps: { vertices: 18333 * 3 }
}, // 55k or 39 998
  { name: `${formatBigNumber(20000 * 3)} vert`, mesh:
generateMash(20000), customProps: { vertices: 20000 * 3 }
}, // 60k or 60 000
]
```

### 3.3.8 Налаштування виводу

Виведення даних, яке можна побачити в лістингу 10, також налаштовується в головному файлі програми. Можна налаштувати, наприклад, таблицю csv задавши ім’я колонок та указати як їх значення будуть розраховуватися. Виводяться дані окремо в файли encode та decode.

Лістинг 10 — Налаштування виводу у csv файл в головному файлі

```
const csvHeader = [
  { id: 'vertices', title: 'Vertices' },
  { id: 'time', title: 'Time (ms)' },
  { id: 'sizeBytesDecoded', title: 'Size decoded (bytes)'
},
  { id: 'sizeReadableDecoded', title: 'Size decoded' },
  { id: 'sizeBytesEncoded', title: 'Size encoded (bytes)'
},
  { id: 'sizeReadableEncoded', title: 'Size encoded' },
  { id: 'encodeSpeed', title: 'Encode speed' },
  { id: 'decodeSpeed', title: 'Decode speed' },
  { id: 'quantizePosition', title: 'Quantize position' },
  { id: 'quantizeTexcoord', title: 'Quantize texcoord' },
  { id: 'quantizeNormal', title: 'Quantize normal' },
];
const csvEncodeOutputGenerator = new
TestOutputCsv('./encode.csv', csvHeader,
  (test) => (test.group === 'decode' ? undefined : {
    name: test.name,
    group: test.group ?? '',
    vertices: test.customProps?.vertices ?? 0,
    time: test.getMetrics().average,
    sizeBytesEncoded: test.customProps?.sizeEncoded ?? 0,
    sizeBytesDecoded: test.customProps?.sizeDecoded ?? 0,
    sizeReadableEncoded:
convertBytesToHumanReadable(test.customProps?.sizeEncoded
?? 0),
    sizeReadableDecoded:
convertBytesToHumanReadable(test.customProps?.sizeDecoded
?? 0),
    encodeSpeed: cliOptions.encodeSpeed,
    decodeSpeed: cliOptions.decodeSpeed,
    quantizePosition: cliOptions.quantizePosition,
    quantizeTexcoord: cliOptions.quantizeTexcoord,
    quantizeGeneric: cliOptions.quantizeGeneric,
    quantizeNormal: cliOptions.quantizeNormal,
  })
```

```
);
const csvDecodeOutputGenerator = new
TestOutputCsv('./decode.csv', csvHeader,
  (test) => (test.group === 'encode' ? undefined : {
    name: test.name,
    group: test.group ?? '',
    vertices: test.customProps?.vertices ?? 0,
    time: test.getMetrics().average,
    sizeBytesEncoded: test.customProps?.sizeEncoded ?? 0,
    sizeBytesDecoded: test.customProps?.sizeDecoded ?? 0,
    sizeReadableEncoded:
convertBytesToHumanReadable(test.customProps?.sizeEncoded
?? 0),
    sizeReadableDecoded:
convertBytesToHumanReadable(test.customProps?.sizeDecoded
?? 0),
    encodeSpeed: cliOptions.encodeSpeed,
    decodeSpeed: cliOptions.decodeSpeed,
    quantizePosition: cliOptions.quantizePosition,
    quantizeTexcoord: cliOptions.quantizeTexcoord,
    quantizeGeneric: cliOptions.quantizeGeneric,
    quantizeNormal: cliOptions.quantizeNormal,
  })
);
testSuite.addOutputGenerator(csvEncodeOutputGenerator);
testSuite.addOutputGenerator(csvDecodeOutputGenerator);
```

### 3.4 Налаштування таблиці для порівняння результатів

Для полегшення аналізу ефективності стиснення Draco було створено порівняльну таблицю за допомогою Google Sheets. Ця таблиця служить централізованим сховищем для запису та порівняння показників із різних тестів, виконаних за допомогою системи порівняльного аналізу. Структура таблиці організована таким чином, щоб включати стовпці для основних параметрів і результатів:

1. Назва моделі (визначає 3D-модель, яка використовується в тесті)
2. Кількість вершин та індексів (фіксує складність моделі).
3. Параметри квантування (записує значення, застосовані до таких атрибутів, як положення та координати текстури).
4. Час стиснення (вказує, скільки часу потрібно для кодування моделі).

5. Час декомпресії (відображає час, необхідний для декодування моделі).
6. Закодований розмір (представляє розмір стиснутих даних).
7. Ступінь стиснення (підкреслює ефективність стиснення порівняно з вихідним розміром).

Використання Google Таблиць гарантує, що дані доступні, ними легко ділитися та редагувати, що сприяє спільній оцінці результатів. Це також дозволяє застосовувати формули для автоматизованих розрахунків, наприклад коефіцієнтів стиснення або побудову графіків за цими результатами для наочного порівняння. Цей структурований підхід спрощує визначення тенденцій, аномалій або закономірностей у різних тестах, роблячи результати більш зручними для інтерпретації та застосування.

### 3.5 Висновки з розділу 3

Впровадження комп’ютерної системи для оцінки ефективності стиснення Draco призвело до створення гнучкої модульної структури. Розділення логіки порівняльного аналізу та основної функціональності програми забезпечує ясність і зручність обслуговування, одночасно полегшуючи інтеграцію нових функцій або коригування за потреби. Використовуючи бібліотеку Draco, система забезпечує ефективне стиснення та розпакування 3D-моделей, дозволяючи детально аналізувати продуктивність за різних налаштувань квантування та методів кодування.

Здатність виводити результати в різних форматах, включаючи файли CSV і графіки, робить програму універсальною та практичною для дослідницьких робочих процесів. Його дизайн не тільки відповідає вимогам для аналізу ефективності стиснення, але й забезпечує основу для майбутніх розширень або адаптації до нових контекстів. Система ефективно збалансовує продуктивність, зручність використання та розширюваність, підкреслюючи її придатність для детальних завдань порівняльного аналізу.

# 4. ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ КОМП'ЮТЕРНОЇ СИСТЕМИ ТА ПОБУДОВА РЕКОМЕНДАЦІЙ

## 4.1 Прогон програми з подальшим занесенням результатів в таблицю

Для заповнення таблиці результатів було виконано прогон програми із варіюванням параметрів тестування. Було визначено такі змінні: параметри квантування (позицій, текстурних координат, нормалей), метод стиснення (EdgeBreaker або Sequential), а також кількість вершин тестових моделей.

Draco bench suite
quantize | position 12, uv 12, generic 12
speed | encode 0, decode 10

| Name | Group | Runs | Avg. time (ms) | Vertices | SizeEncoded | SizeDecoded |
|---|---|---|---|---|---|---|
| none | | 200 | 0.0085 | | | |
| 9 999 vert | encode | 200 | 6.5868 | 9999.00 | 103364.00 | 239976.00 |
| 9 999 vert | decode | 200 | 1.9346 | 9999.00 | 103364.00 | 239976.00 |
| 15 000 vert | encode | 200 | 9.4721 | 15000.00 | 155005.00 | 360000.00 |
| 15 000 vert | decode | 200 | 2.8889 | 15000.00 | 155005.00 | 360000.00 |
| 19 998 vert | encode | 200 | 12.4477 | 19998.00 | 206617.00 | 479952.00 |
| 19 998 vert | decode | 200 | 3.7285 | 19998.00 | 206617.00 | 479952.00 |
| 24 999 vert | encode | 200 | 14.9631 | 24999.00 | 258256.00 | 599976.00 |
| 24 999 vert | decode | 200 | 4.8024 | 24999.00 | 258256.00 | 599976.00 |
| 30 000 vert | encode | 200 | 18.3202 | 30000.00 | 309895.00 | 720000.00 |
| 30 000 vert | decode | 200 | 5.7484 | 30000.00 | 309895.00 | 720000.00 |
| 34 998 vert | encode | 200 | 20.7920 | 34998.00 | 361504.00 | 839952.00 |
| 34 998 vert | decode | 200 | 6.6658 | 34998.00 | 361504.00 | 839952.00 |
| 39 999 vert | encode | 200 | 23.9525 | 39999.00 | 413149.00 | 959976.00 |

Рисунок 2 — Проміжна таблиця з консолі для попередньої візуалізації користувачу

Для кожного тестового випадку виконувалося стиснення моделі з подальшим вимірюванням часу виконання операцій кодування та декодування, а також визначенням розмірів файлів після стиснення й розпакування.

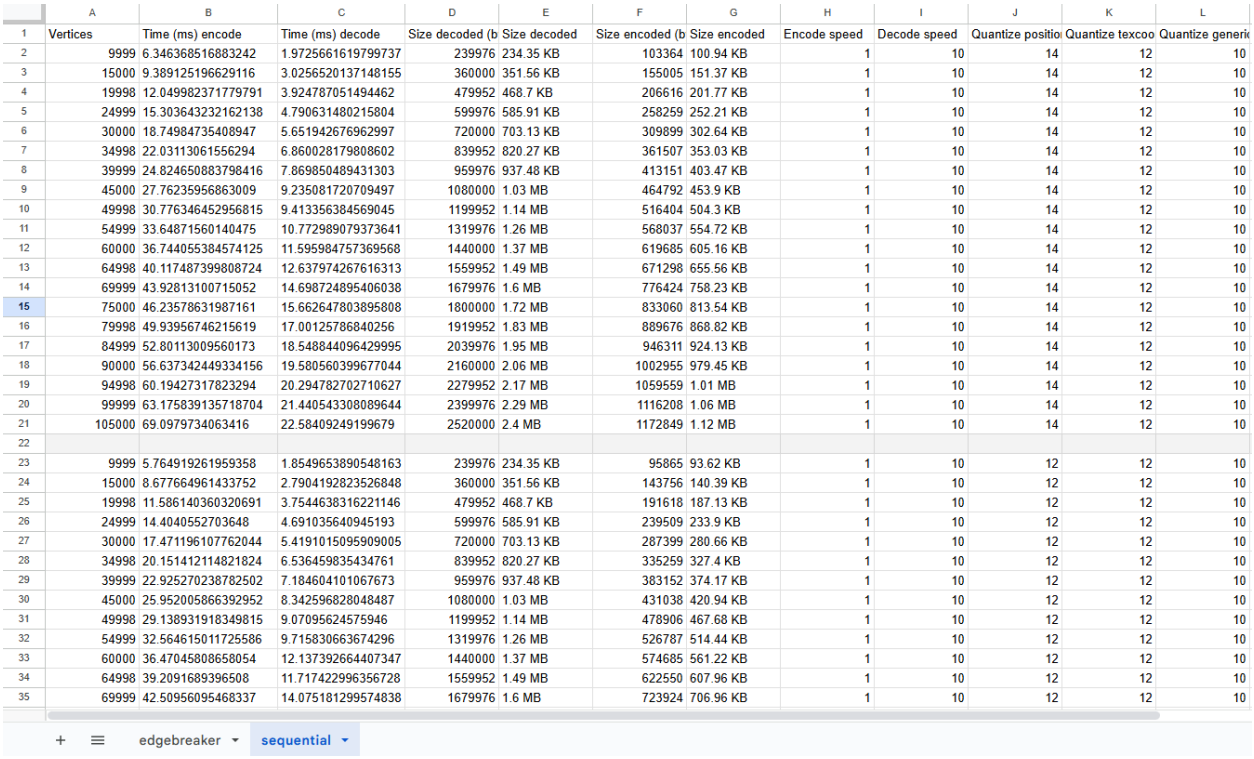
Результати були систематично занесені в таблицю, яка включала ключові показники: кількість вершин у моделі, параметри квантування, час кодування й декодування, розміри файлів і метод стиснення. Така структура дозволила організувати дані та підготувати їх до подальшого аналізу, сприяючи виявленню закономірностей і аномалій у роботі алгоритмів.

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Vertices | Time (ms) encode | Time (ms) decode | Size decoded (b | Size decoded | Size encoded (b | Size encoded | Encode speed | Decode speed | Quantize positio | Quantize texcoo | Quantize generic |
| 2 | 9999 | 6.586845170701012 | 1.934575026744550 | 239976 | 234.35 KB | 103364 | 100.94 KB | 1 | 10 | 14 | 12 | 10 |
| 3 | 15000 | 9.472080396510929 | 2.888879183419817 | 360000 | 351.56 KB | 155005 | 151.37 KB | 1 | 10 | 14 | 12 | 10 |
| 4 | 19998 | 12.447722097837225 | 3.728546665944790 | 479952 | 468.7 KB | 206617 | 201.77 KB | 1 | 10 | 14 | 12 | 10 |
| 5 | 24999 | 14.963072248412008 | 4.802366333837593 | 599976 | 585.91 KB | 258256 | 252.2 KB | 1 | 10 | 14 | 12 | 10 |
| 6 | 30000 | 18.320159346754107 | 5.748362571944506 | 720000 | 703.13 KB | 309895 | 302.63 KB | 1 | 10 | 14 | 12 | 10 |
| 7 | 34998 | 20.792041009857464 | 6.665805643834383 | 839952 | 820.27 KB | 361504 | 353.03 KB | 1 | 10 | 14 | 12 | 10 |
| 8 | 39999 | 23.95253636965459 | 7.562059685767450 | 959976 | 937.48 KB | 413149 | 403.47 KB | 1 | 10 | 14 | 12 | 10 |
| 9 | 45000 | 26.944518229150713 | 8.832358541337417 | 1080000 | 1.03 MB | 464793 | 453.9 KB | 1 | 10 | 14 | 12 | 10 |
| 10 | 49998 | 30.61569228666632 | 9.460306163091493 | 1199952 | 1.14 MB | 516403 | 504.3 KB | 1 | 10 | 14 | 12 | 10 |
| 11 | 54999 | 33.964071717967 | 10.33712369039866 | 1319976 | 1.26 MB | 568038 | 554.72 KB | 1 | 10 | 14 | 12 | 10 |
| 12 | 60000 | 37.00671082627277 | 11.40057154875896 | 1440000 | 1.37 MB | 619683 | 605.16 KB | 1 | 10 | 14 | 12 | 10 |
| 13 | 64998 | 39.78518922277639 | 12.22346428133423 | 1559952 | 1.49 MB | 671290 | 655.56 KB | 1 | 10 | 14 | 12 | 10 |
| 14 | 69999 | 43.983534099936705 | 14.72670398454633 | 1679976 | 1.6 MB | 776421 | 758.22 KB | 1 | 10 | 14 | 12 | 10 |
| 15 | 75000 | 47.75406268930843 | 15.73831272308775 | 1800000 | 1.72 MB | 833063 | 813.54 KB | 1 | 10 | 14 | 12 | 10 |
| 16 | 79998 | 50.103459053222906 | 16.87257111507188 | 1919952 | 1.83 MB | 889676 | 868.82 KB | 1 | 10 | 14 | 12 | 10 |
| 17 | 84999 | 54.79759095678724 | 18.01029509127525 | 2039976 | 1.95 MB | 946307 | 924.13 KB | 1 | 10 | 14 | 12 | 10 |
| 18 | 90000 | 57.18168566939623 | 19.21075976422676 | 2160000 | 2.06 MB | 1002956 | 979.45 KB | 1 | 10 | 14 | 12 | 10 |
| 19 | 94998 | 60.86241552877054 | 20.0302358351569 | 2279952 | 2.17 MB | 1059562 | 1.01 MB | 1 | 10 | 14 | 12 | 10 |
| 20 | 99999 | 63.41756250100832 | 21.50860210576316 | 2399976 | 2.29 MB | 1116214 | 1.06 MB | 1 | 10 | 14 | 12 | 10 |
| 21 | 105000 | 68.75093576210135 | 22.2165326188681 | 2520000 | 2.4 MB | 1172847 | 1.12 MB | 1 | 10 | 14 | 12 | 10 |
| 22 | | | | | | | | | | | | |
| 23 | 9999 | 5.904762551713402 | 1.798203007118917 | 239976 | 234.35 KB | 95865 | 93.62 KB | 1 | 10 | 12 | 12 | 10 |
| 24 | 15000 | 8.614844852562454 | 2.86002003557359 | 360000 | 351.56 KB | 143756 | 140.39 KB | 1 | 10 | 12 | 12 | 10 |
| 25 | 19998 | 11.703158868055816 | 3.701711722561916 | 479952 | 468.7 KB | 191619 | 187.13 KB | 1 | 10 | 12 | 12 | 10 |
| 26 | 24999 | 14.473720314636696 | 4.494103339316396 | 599976 | 585.91 KB | 239507 | 233.89 KB | 1 | 10 | 12 | 12 | 10 |
| 27 | 30000 | 18.0498931895241 | 5.458119248691520 | 720000 | 703.13 KB | 287394 | 280.66 KB | 1 | 10 | 12 | 12 | 10 |
| 28 | 34998 | 20.208834780821118 | 6.537367828967820 | 839952 | 820.27 KB | 335256 | 327.4 KB | 1 | 10 | 12 | 12 | 10 |
| 29 | 39999 | 23.32808235978238 | 7.091701709602677 | 959976 | 937.48 KB | 383150 | 374.17 KB | 1 | 10 | 12 | 12 | 10 |
| 30 | 45000 | 25.995894369956122 | 7.961211129518231 | 1080000 | 1.03 MB | 431045 | 420.94 KB | 1 | 10 | 12 | 12 | 10 |
| 31 | 49998 | 28.855504618343836 | 8.976451086370233 | 1199952 | 1.14 MB | 478904 | 467.68 KB | 1 | 10 | 12 | 12 | 10 |
| 32 | 54999 | 33.074261528173835 | 9.872824927247265 | 1319976 | 1.26 MB | 526789 | 514.44 KB | 1 | 10 | 12 | 12 | 10 |
| 33 | 60000 | 35.88415542520062 | 10.91698589005923 | 1440000 | 1.37 MB | 574682 | 561.21 KB | 1 | 10 | 12 | 12 | 10 |
| 34 | 64998 | 39.14748148085232 | 12.46208087154345 | 1559952 | 1.49 MB | 622541 | 607.95 KB | 1 | 10 | 12 | 12 | 10 |
| 35 | 69999 | 42.875362258039324 | 14.02711652452045 | 1679976 | 1.6 MB | 723922 | 706.96 KB | 1 | 10 | 12 | 12 | 10 |

+ ≡ edgebreaker sequential

Рисунок 3 — Частина отриманих результатів з використанням алгоритму edgebreaker

Моделі створювались з кроком в 5000 вершин: з 9999 вершин по 105000. Також були задані такі налаштування квантування з кроком 2: квантування позиції з 14 по 6, квантування текстурних координат з 12 по 6, квантування нормалей з 10 до 6. Подальший аналіз буде організовано на основі отриманої таблиці. Усього таблиць було отримано 2 залежно від методів стиснення edgebreaker або sequential.



| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Vertices | Time (ms) encode | Time (ms) decode | Size decoded (b | Size decoded | Size encoded (b | Size encoded | Encode speed | Decode speed | Quantize positio | Quantize texcoo | Quantize generic |
| 2 | 9999 | 6.346368516883242 | 1.9725661619799737 | 239976 | 234.35 KB | 103364 | 100.94 KB | 1 | 10 | 14 | 12 | 10 |
| 3 | 15000 | 9.389125196629116 | 3.0256520137148155 | 360000 | 351.56 KB | 155005 | 151.37 KB | 1 | 10 | 14 | 12 | 10 |
| 4 | 19998 | 12.049982371779791 | 3.924787051494462 | 479952 | 468.7 KB | 206616 | 201.77 KB | 1 | 10 | 14 | 12 | 10 |
| 5 | 24999 | 15.303643232162138 | 4.790631480215804 | 599976 | 585.91 KB | 258259 | 252.21 KB | 1 | 10 | 14 | 12 | 10 |
| 6 | 30000 | 18.74984735408947 | 5.65194267962997 | 720000 | 703.13 KB | 309899 | 302.64 KB | 1 | 10 | 14 | 12 | 10 |
| 7 | 34998 | 22.03113061556294 | 6.860028179808602 | 839952 | 820.27 KB | 361507 | 353.03 KB | 1 | 10 | 14 | 12 | 10 |
| 8 | 39999 | 24.824650883798416 | 7.869850489431303 | 959976 | 937.48 KB | 413151 | 403.47 KB | 1 | 10 | 14 | 12 | 10 |
| 9 | 45000 | 27.76235956863009 | 9.235081720709497 | 1080000 | 1.03 MB | 464792 | 453.9 KB | 1 | 10 | 14 | 12 | 10 |
| 10 | 49998 | 30.776346452956815 | 9.413356384569045 | 1199952 | 1.14 MB | 516404 | 504.3 KB | 1 | 10 | 14 | 12 | 10 |
| 11 | 54999 | 33.64871560140475 | 10.772989079373641 | 1319976 | 1.26 MB | 568037 | 554.72 KB | 1 | 10 | 14 | 12 | 10 |
| 12 | 60000 | 36.744055384574125 | 11.595984757369568 | 1440000 | 1.37 MB | 619685 | 605.16 KB | 1 | 10 | 14 | 12 | 10 |
| 13 | 64998 | 40.117487399808724 | 12.637974267616313 | 1559952 | 1.49 MB | 671298 | 655.56 KB | 1 | 10 | 14 | 12 | 10 |
| 14 | 69999 | 43.92813100715052 | 14.698724895406038 | 1679976 | 1.6 MB | 776424 | 758.23 KB | 1 | 10 | 14 | 12 | 10 |
| 15 | 75000 | 46.23578631987161 | 15.662647803895808 | 1800000 | 1.72 MB | 833060 | 813.54 KB | 1 | 10 | 14 | 12 | 10 |
| 16 | 79998 | 49.93956746215619 | 17.00125786840256 | 1919952 | 1.83 MB | 889676 | 868.82 KB | 1 | 10 | 14 | 12 | 10 |
| 17 | 84999 | 52.80113009560173 | 18.548844096429995 | 2039976 | 1.95 MB | 946311 | 924.13 KB | 1 | 10 | 14 | 12 | 10 |
| 18 | 90000 | 56.637342449334156 | 19.580560399677044 | 2160000 | 2.06 MB | 1002955 | 979.45 KB | 1 | 10 | 14 | 12 | 10 |
| 19 | 94998 | 60.19427317823294 | 20.294782702710627 | 2279952 | 2.17 MB | 1059559 | 1.01 MB | 1 | 10 | 14 | 12 | 10 |
| 20 | 99999 | 63.175839135718704 | 21.440543308089644 | 2399976 | 2.29 MB | 1116208 | 1.06 MB | 1 | 10 | 14 | 12 | 10 |
| 21 | 105000 | 69.0979734063416 | 22.58409249199679 | 2520000 | 2.4 MB | 1172849 | 1.12 MB | 1 | 10 | 14 | 12 | 10 |
| 22 | | | | | | | | | | | | |
| 23 | 9999 | 5.764919261959358 | 1.8549653890548163 | 239976 | 234.35 KB | 95865 | 93.62 KB | 1 | 10 | 12 | 12 | 10 |
| 24 | 15000 | 8.677664961433752 | 2.7904192823526848 | 360000 | 351.56 KB | 143756 | 140.39 KB | 1 | 10 | 12 | 12 | 10 |
| 25 | 19998 | 11.586140360320691 | 3.7544638316221146 | 479952 | 468.7 KB | 191618 | 187.13 KB | 1 | 10 | 12 | 12 | 10 |
| 26 | 24999 | 14.4040552703648 | 4.691035640945193 | 599976 | 585.91 KB | 239509 | 233.9 KB | 1 | 10 | 12 | 12 | 10 |
| 27 | 30000 | 17.471196107762044 | 5.4191015095909005 | 720000 | 703.13 KB | 287399 | 280.66 KB | 1 | 10 | 12 | 12 | 10 |
| 28 | 34998 | 20.151412114821824 | 6.536459835434761 | 839952 | 820.27 KB | 335259 | 327.4 KB | 1 | 10 | 12 | 12 | 10 |
| 29 | 39999 | 22.925270238782502 | 7.184604101067673 | 959976 | 937.48 KB | 383152 | 374.17 KB | 1 | 10 | 12 | 12 | 10 |
| 30 | 45000 | 25.952005866392952 | 8.342596828048487 | 1080000 | 1.03 MB | 431038 | 420.94 KB | 1 | 10 | 12 | 12 | 10 |
| 31 | 49998 | 29.138931918349815 | 9.07095624575946 | 1199952 | 1.14 MB | 478906 | 467.68 KB | 1 | 10 | 12 | 12 | 10 |
| 32 | 54999 | 32.564615011725586 | 9.715830663674296 | 1319976 | 1.26 MB | 526787 | 514.44 KB | 1 | 10 | 12 | 12 | 10 |
| 33 | 60000 | 36.47045808658054 | 12.137392664407347 | 1440000 | 1.37 MB | 574685 | 561.22 KB | 1 | 10 | 12 | 12 | 10 |
| 34 | 64998 | 39.2091689396508 | 11.717422996356728 | 1559952 | 1.49 MB | 622550 | 607.96 KB | 1 | 10 | 12 | 12 | 10 |
| 35 | 69999 | 42.50956095468337 | 14.075181299574838 | 1679976 | 1.6 MB | 723924 | 706.96 KB | 1 | 10 | 12 | 12 | 10 |

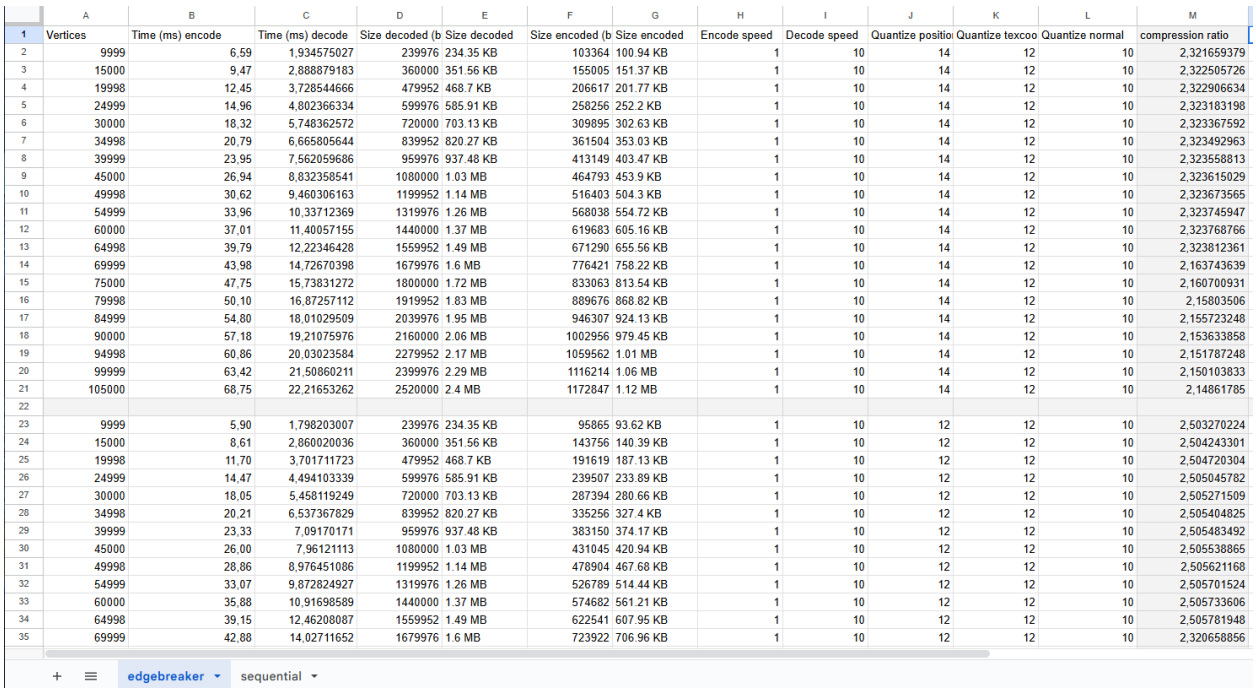


Рисунок 4 — Частина отриманих результатів з використанням алгоритму sequential

## 4.2 Аналіз ефективності стиснення тривимірних моделей за допомогою Draco

Зібрані дані дають змогу зрозуміти ефективність бібліотеки Draco у стисненні 3D-моделей. Для цього аналізу було введено додатковий стовпчик з коефіцієнтом стиснення. Коефіцієнт стиснення розраховується шляхом порівняння ваги моделі до та після компресії.

### 4.2.1 Аналіз ефективності стиснення Edgebreaker



| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Vertices | Time (ms) encode | Time (ms) decode | Size decoded (b | Size decoded | Size encoded (b | Size encoded | Encode speed | Decode speed | Quantize positio | Quantize texcoo | Quantize normal | compression ratio |
| 2 | 9999 | 6,59 | 1,934575027 | 239976 | 234.35 KB | 103364 | 100.94 KB | 1 | 10 | 14 | 12 | 10 | 2,321659379 |
| 3 | 15000 | 9,47 | 2,888879183 | 360000 | 351.56 KB | 155005 | 151.37 KB | 1 | 10 | 14 | 12 | 10 | 2,322505726 |
| 4 | 19998 | 12,45 | 3,728544666 | 479952 | 468.7 KB | 206617 | 201.77 KB | 1 | 10 | 14 | 12 | 10 | 2,322906634 |
| 5 | 24999 | 14,96 | 4,802366334 | 599976 | 585.91 KB | 258256 | 252.2 KB | 1 | 10 | 14 | 12 | 10 | 2,323183198 |
| 6 | 30000 | 18,32 | 5,748362572 | 720000 | 703.13 KB | 309895 | 302.63 KB | 1 | 10 | 14 | 12 | 10 | 2,323367592 |
| 7 | 34998 | 20,79 | 6,665805644 | 839952 | 820.27 KB | 361504 | 353.03 KB | 1 | 10 | 14 | 12 | 10 | 2,323492963 |
| 8 | 39999 | 23,95 | 7,562059686 | 959976 | 937.48 KB | 413149 | 403.47 KB | 1 | 10 | 14 | 12 | 10 | 2,323558813 |
| 9 | 45000 | 26,94 | 8,832358541 | 1080000 | 1.03 MB | 464793 | 453.9 KB | 1 | 10 | 14 | 12 | 10 | 2,323615029 |
| 10 | 49998 | 30,62 | 9,460306163 | 1199952 | 1.14 MB | 516403 | 504.3 KB | 1 | 10 | 14 | 12 | 10 | 2,323673565 |
| 11 | 54999 | 33,96 | 10,33712369 | 1319976 | 1.26 MB | 568038 | 554.72 KB | 1 | 10 | 14 | 12 | 10 | 2,323745947 |
| 12 | 60000 | 37,01 | 11,40057155 | 1440000 | 1.37 MB | 619683 | 605.16 KB | 1 | 10 | 14 | 12 | 10 | 2,323768766 |
| 13 | 64998 | 39,79 | 12,22346428 | 1559952 | 1.49 MB | 671290 | 655.56 KB | 1 | 10 | 14 | 12 | 10 | 2,323812361 |
| 14 | 69999 | 43,98 | 14,72670398 | 1679976 | 1.6 MB | 776421 | 758.22 KB | 1 | 10 | 14 | 12 | 10 | 2,163743639 |
| 15 | 75000 | 47,75 | 15,73831272 | 1800000 | 1.72 MB | 833063 | 813.54 KB | 1 | 10 | 14 | 12 | 10 | 2,160700931 |
| 16 | 79998 | 50,10 | 16,87257112 | 1919952 | 1.83 MB | 889676 | 868.82 KB | 1 | 10 | 14 | 12 | 10 | 2,15803506 |
| 17 | 84999 | 54,80 | 18,01029509 | 2039976 | 1.95 MB | 946307 | 924.13 KB | 1 | 10 | 14 | 12 | 10 | 2,155723248 |
| 18 | 90000 | 57,18 | 19,21075976 | 2160000 | 2.06 MB | 1002956 | 979.45 KB | 1 | 10 | 14 | 12 | 10 | 2,153633858 |
| 19 | 94998 | 60,86 | 20,03023584 | 2279952 | 2.17 MB | 1059562 | 1.01 MB | 1 | 10 | 14 | 12 | 10 | 2,151787248 |
| 20 | 99999 | 63,42 | 21,50860211 | 2399976 | 2.29 MB | 1116214 | 1.06 MB | 1 | 10 | 14 | 12 | 10 | 2,150103833 |
| 21 | 105000 | 68,75 | 22,21653262 | 2520000 | 2.4 MB | 1172847 | 1.12 MB | 1 | 10 | 14 | 12 | 10 | 2,14861785 |
| 22 | | | | | | | | | | | | | |
| 23 | 9999 | 5,90 | 1,798203007 | 239976 | 234.35 KB | 95865 | 93.62 KB | 1 | 10 | 12 | 12 | 10 | 2,503270224 |
| 24 | 15000 | 8,61 | 2,860020036 | 360000 | 351.56 KB | 143756 | 140.39 KB | 1 | 10 | 12 | 12 | 10 | 2,504243301 |
| 25 | 19998 | 11,70 | 3,701711723 | 479952 | 468.7 KB | 191619 | 187.13 KB | 1 | 10 | 12 | 12 | 10 | 2,504720304 |
| 26 | 24999 | 14,47 | 4,494103339 | 599976 | 585.91 KB | 239507 | 233.89 KB | 1 | 10 | 12 | 12 | 10 | 2,505045782 |
| 27 | 30000 | 18,05 | 5,458119249 | 720000 | 703.13 KB | 287394 | 280.66 KB | 1 | 10 | 12 | 12 | 10 | 2,505271509 |
| 28 | 34998 | 20,21 | 6,537367829 | 839952 | 820.27 KB | 335256 | 327.4 KB | 1 | 10 | 12 | 12 | 10 | 2,505404825 |
| 29 | 39999 | 23,33 | 7,09170171 | 959976 | 937.48 KB | 383150 | 374.17 KB | 1 | 10 | 12 | 12 | 10 | 2,505483492 |
| 30 | 45000 | 26,00 | 7,96121113 | 1080000 | 1.03 MB | 431045 | 420.94 KB | 1 | 10 | 12 | 12 | 10 | 2,505538865 |
| 31 | 49998 | 28,86 | 8,976451086 | 1199952 | 1.14 MB | 478904 | 467.68 KB | 1 | 10 | 12 | 12 | 10 | 2,505621168 |
| 32 | 54999 | 33,07 | 9,872824927 | 1319976 | 1.26 MB | 526789 | 514.44 KB | 1 | 10 | 12 | 12 | 10 | 2,505701524 |
| 33 | 60000 | 35,88 | 10,91698589 | 1440000 | 1.37 MB | 574682 | 561.21 KB | 1 | 10 | 12 | 12 | 10 | 2,505733606 |
| 34 | 64998 | 39,15 | 12,46208087 | 1559952 | 1.49 MB | 622541 | 607.95 KB | 1 | 10 | 12 | 12 | 10 | 2,505781948 |
| 35 | 69999 | 42,88 | 14,02711652 | 1679976 | 1.6 MB | 723922 | 706.96 KB | 1 | 10 | 12 | 12 | 10 | 2,320658856 |



Рисунок 5 — Додано розрахування коефіцієнту стиснення у Edgebreaker

Від кількості вершин дані різняться, але не суттєво. Важливим є то що коефіцієнт змінюється не лінійно, а скоріше ламано. Наприклад при квантуванні позиції з параметром 14 у 9999 вершинної моделі коефіцієнт стискання становить 2,321 і зі збільшенням кількості вершин коефіцієнт зростає до 64998 вершинної моделі й становить 2,323. Після цього відбувається переламний момент та вже з 69999 де коефіцієнт різко став 2,163 та зі збільшенням вершин тільки зменшився. Так на 105000 вершинній моделі коефіцієнт вже становив 2.148.

Створені графіки цю різницю демонструють наочніше. Слід зауважити, що така проблема стосується не тільки такого параметра як квантування позиції, це також спостерігається й на інших параметрах: квантування текстурних координат та квантування нормалей.

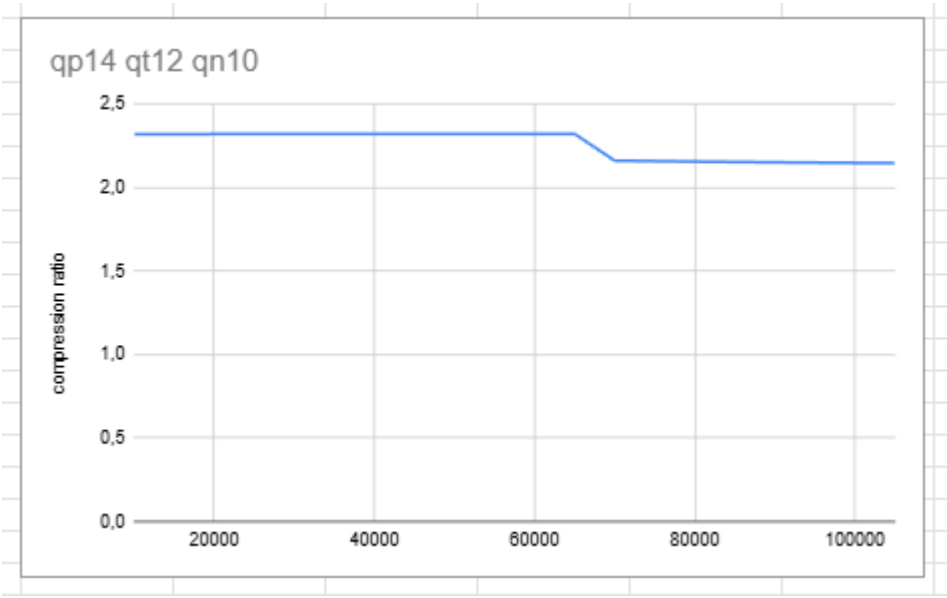



Рисунок 6 — Квантування позиції 14, текстур 12, нормалей 10 алгоритму edgebreaker

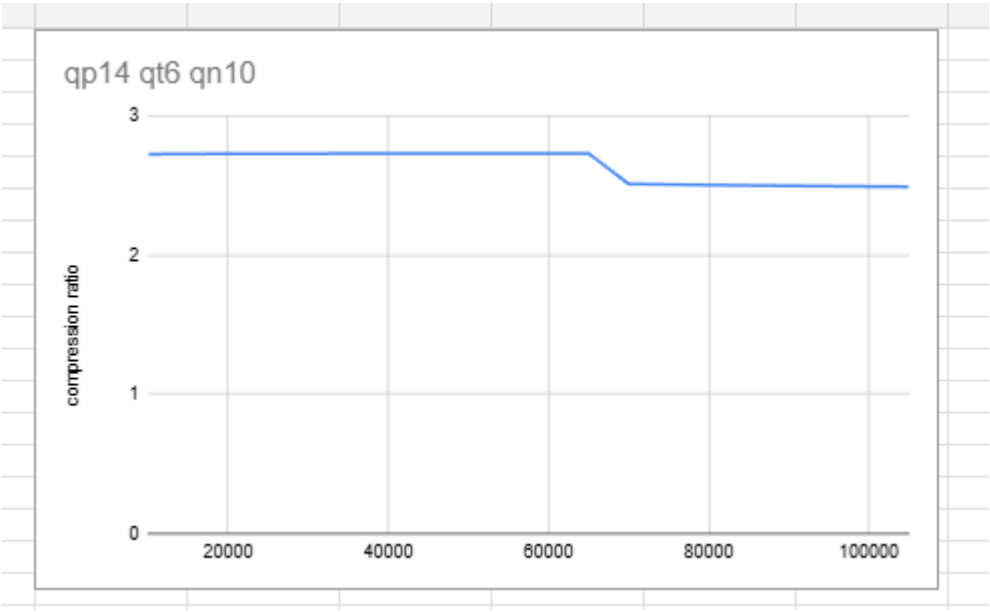



Рисунок 7 — Квантування позиції 14, текстур 6, нормалей 10 алгоритму edgebreaker

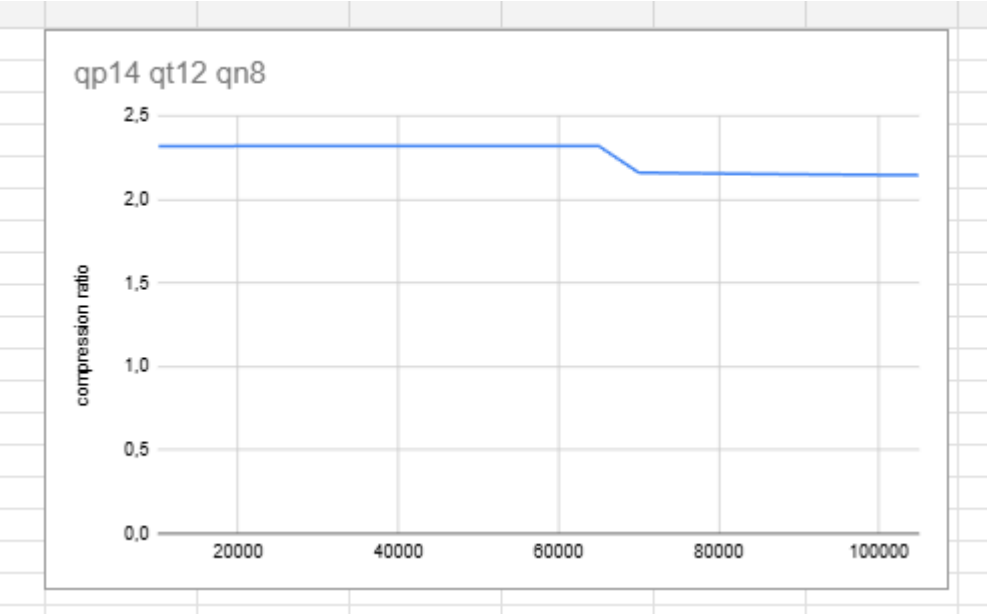
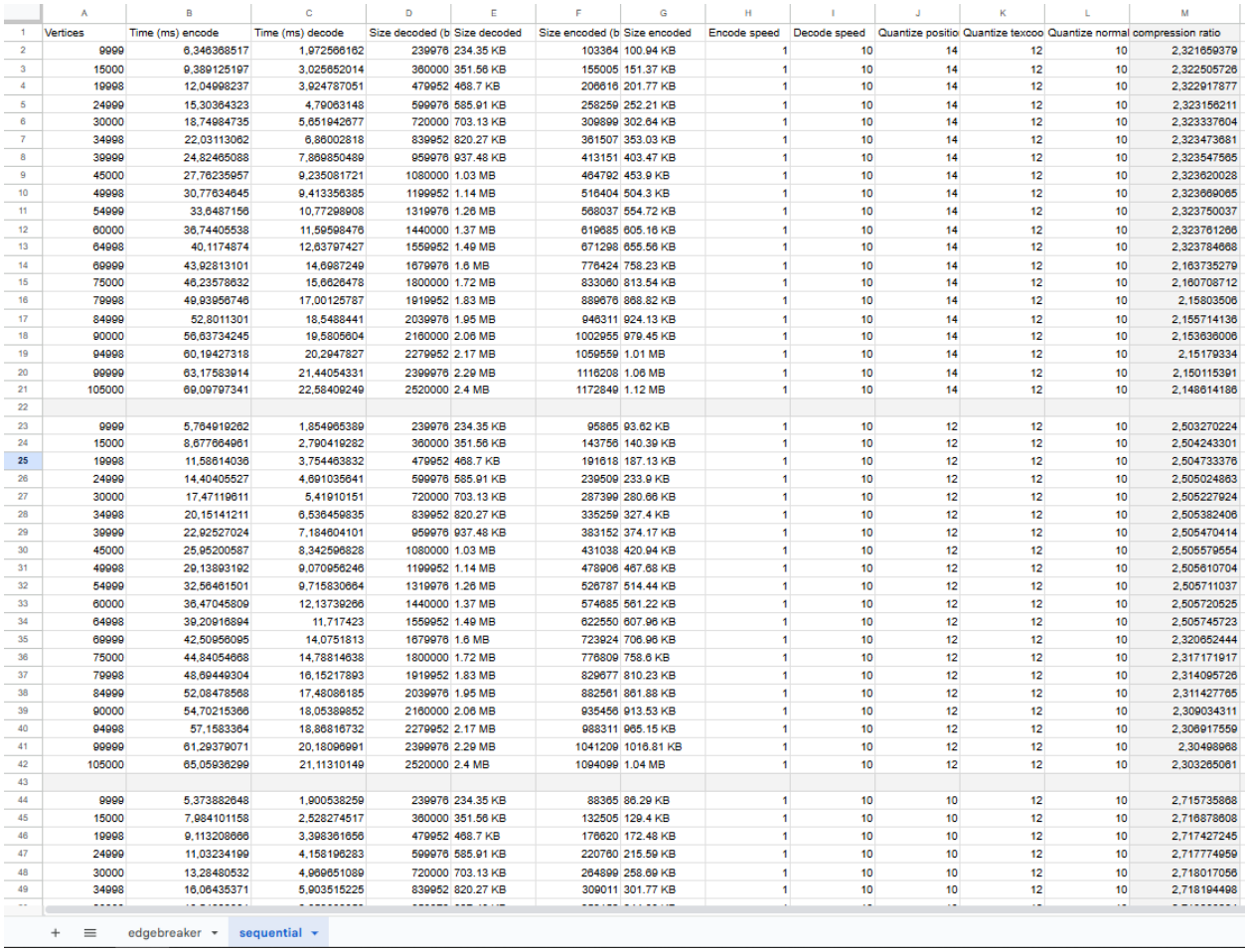



Рисунок 8 — Квантування позиції 14, текстур 12, нормалей 8 алгоритму edge-breaker

Графіки, побудовані на основі зібраних даних, демонструють нерівномірність змін коефіцієнтів стиснення що залежить від кількості вершин. Це не залежить від ніяких параметрів стиснення. Зниження ефективності після 65 000 вершин свідчить про необхідність обмеження розміру моделей для досягнення оптимальних результатів стиснення.

### 4.2.2 Аналіз ефективності стиснення Sequential



| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Vertices | Time (ms) encode | Time (ms) decode | Size decoded (b | Size decoded | Size encoded (b | Size encoded | Encode speed | Decode speed | Quantize positio | Quantize texcoo | Quantize normal | compression ratio |
| 2 | 9999 | 6,346368517 | 1,972566162 | 239976 | 234.35 KB | 103364 | 100.94 KB | 1 | 10 | 14 | 12 | 10 | 2,321659379 |
| 3 | 15000 | 9,389125197 | 3,025652014 | 360000 | 351.56 KB | 155005 | 151.37 KB | 1 | 10 | 14 | 12 | 10 | 2,322505726 |
| 4 | 19998 | 12,04998237 | 3,924787051 | 479952 | 468.7 KB | 206616 | 201.77 KB | 1 | 10 | 14 | 12 | 10 | 2,322917877 |
| 5 | 24999 | 15,30364323 | 4,79063148 | 599976 | 585.91 KB | 258259 | 252.21 KB | 1 | 10 | 14 | 12 | 10 | 2,323156211 |
| 6 | 30000 | 18,74984735 | 5,651942677 | 720000 | 703.13 KB | 309899 | 302.64 KB | 1 | 10 | 14 | 12 | 10 | 2,323337604 |
| 7 | 34998 | 22,03113062 | 6,86002818 | 839952 | 820.27 KB | 361507 | 353.03 KB | 1 | 10 | 14 | 12 | 10 | 2,323473681 |
| 8 | 39999 | 24,82465088 | 7,869850489 | 959976 | 937.48 KB | 413151 | 403.47 KB | 1 | 10 | 14 | 12 | 10 | 2,323547565 |
| 9 | 45000 | 27,76235957 | 9,235081721 | 1080000 | 1.03 MB | 464792 | 453.9 KB | 1 | 10 | 14 | 12 | 10 | 2,323620028 |
| 10 | 49998 | 30,77634645 | 9,413356385 | 1199952 | 1.14 MB | 516404 | 504.3 KB | 1 | 10 | 14 | 12 | 10 | 2,323669065 |
| 11 | 54999 | 33,6487156 | 10,77298908 | 1319976 | 1.26 MB | 568037 | 554.72 KB | 1 | 10 | 14 | 12 | 10 | 2,323750037 |
| 12 | 60000 | 36,74405538 | 11,59598476 | 1440000 | 1.37 MB | 619685 | 605.16 KB | 1 | 10 | 14 | 12 | 10 | 2,323761266 |
| 13 | 64998 | 40,1174874 | 12,63797427 | 1559952 | 1.49 MB | 671298 | 655.56 KB | 1 | 10 | 14 | 12 | 10 | 2,323784668 |
| 14 | 69999 | 43,92813101 | 14,6987249 | 1679976 | 1.6 MB | 776424 | 758.23 KB | 1 | 10 | 14 | 12 | 10 | 2,163735279 |
| 15 | 75000 | 46,23578632 | 15,6626478 | 1800000 | 1.72 MB | 833060 | 813.54 KB | 1 | 10 | 14 | 12 | 10 | 2,160708712 |
| 16 | 79998 | 49,93956746 | 17,00125787 | 1919952 | 1.83 MB | 889676 | 868.82 KB | 1 | 10 | 14 | 12 | 10 | 2,15803506 |
| 17 | 84999 | 52,8011301 | 18,5488441 | 2039976 | 1.95 MB | 946311 | 924.13 KB | 1 | 10 | 14 | 12 | 10 | 2,155714136 |
| 18 | 90000 | 56,63734245 | 19,5805604 | 2160000 | 2.06 MB | 1002955 | 979.45 KB | 1 | 10 | 14 | 12 | 10 | 2,153636006 |
| 19 | 94998 | 60,19427318 | 20,2947827 | 2279952 | 2.17 MB | 1059559 | 1.01 MB | 1 | 10 | 14 | 12 | 10 | 2,15179334 |
| 20 | 99999 | 63,17583914 | 21,44054331 | 2399976 | 2.29 MB | 1116208 | 1.06 MB | 1 | 10 | 14 | 12 | 10 | 2,150115391 |
| 21 | 105000 | 69,09797341 | 22,58409249 | 2520000 | 2.4 MB | 1172849 | 1.12 MB | 1 | 10 | 14 | 12 | 10 | 2,148614186 |
| 22 | | | | | | | | | | | | | |
| 23 | 9999 | 5,764919262 | 1,854965389 | 239976 | 234.35 KB | 95865 | 93.62 KB | 1 | 10 | 12 | 12 | 10 | 2,503270224 |
| 24 | 15000 | 8,677664961 | 2,790419282 | 360000 | 351.56 KB | 143756 | 140.39 KB | 1 | 10 | 12 | 12 | 10 | 2,504243301 |
| 25 | 19998 | 11,58614036 | 3,754463832 | 479952 | 468.7 KB | 191618 | 187.13 KB | 1 | 10 | 12 | 12 | 10 | 2,504733376 |
| 26 | 24999 | 14,40405527 | 4,691035641 | 599976 | 585.91 KB | 239509 | 233.9 KB | 1 | 10 | 12 | 12 | 10 | 2,505024863 |
| 27 | 30000 | 17,47119611 | 5,41910151 | 720000 | 703.13 KB | 287399 | 280.66 KB | 1 | 10 | 12 | 12 | 10 | 2,505227924 |
| 28 | 34998 | 20,15141211 | 6,536459835 | 839952 | 820.27 KB | 335259 | 327.4 KB | 1 | 10 | 12 | 12 | 10 | 2,505382406 |
| 29 | 39999 | 22,92527024 | 7,184604101 | 959976 | 937.48 KB | 383152 | 374.17 KB | 1 | 10 | 12 | 12 | 10 | 2,505470414 |
| 30 | 45000 | 25,95200587 | 8,342596828 | 1080000 | 1.03 MB | 431038 | 420.94 KB | 1 | 10 | 12 | 12 | 10 | 2,505579554 |
| 31 | 49998 | 29,13893192 | 9,070956246 | 1199952 | 1.14 MB | 478906 | 467.68 KB | 1 | 10 | 12 | 12 | 10 | 2,505610704 |
| 32 | 54999 | 32,56461501 | 9,715830664 | 1319976 | 1.26 MB | 526787 | 514.44 KB | 1 | 10 | 12 | 12 | 10 | 2,505711037 |
| 33 | 60000 | 36,47045809 | 12,13739266 | 1440000 | 1.37 MB | 574685 | 561.22 KB | 1 | 10 | 12 | 12 | 10 | 2,505720525 |
| 34 | 64998 | 39,20916894 | 11,717423 | 1559952 | 1.49 MB | 622550 | 607.96 KB | 1 | 10 | 12 | 12 | 10 | 2,505745723 |
| 35 | 69999 | 42,50956095 | 14,0751813 | 1679976 | 1.6 MB | 723924 | 706.96 KB | 1 | 10 | 12 | 12 | 10 | 2,320652444 |
| 36 | 75000 | 44,84054668 | 14,78814638 | 1800000 | 1.72 MB | 776809 | 758.6 KB | 1 | 10 | 12 | 12 | 10 | 2,317171917 |
| 37 | 79998 | 48,69449304 | 16,15217893 | 1919952 | 1.83 MB | 829677 | 810.23 KB | 1 | 10 | 12 | 12 | 10 | 2,314095726 |
| 38 | 84999 | 52,08478568 | 17,48086185 | 2039976 | 1.95 MB | 882561 | 861.88 KB | 1 | 10 | 12 | 12 | 10 | 2,311427765 |
| 39 | 90000 | 54,70215366 | 18,05389852 | 2160000 | 2.06 MB | 935456 | 913.53 KB | 1 | 10 | 12 | 12 | 10 | 2,309034311 |
| 40 | 94998 | 57,1583364 | 18,86816732 | 2279952 | 2.17 MB | 988311 | 965.15 KB | 1 | 10 | 12 | 12 | 10 | 2,306917559 |
| 41 | 99999 | 61,29379071 | 20,18096991 | 2399976 | 2.29 MB | 1041209 | 1016.81 KB | 1 | 10 | 12 | 12 | 10 | 2,30498968 |
| 42 | 105000 | 65,05936299 | 21,11310149 | 2520000 | 2.4 MB | 1094099 | 1.04 MB | 1 | 10 | 12 | 12 | 10 | 2,303265061 |
| 43 | | | | | | | | | | | | | |
| 44 | 9999 | 5,373882648 | 1,900538259 | 239976 | 234.35 KB | 88365 | 86.29 KB | 1 | 10 | 10 | 12 | 10 | 2,715735868 |
| 45 | 15000 | 7,984101158 | 2,528274517 | 360000 | 351.56 KB | 132505 | 129.4 KB | 1 | 10 | 10 | 12 | 10 | 2,716878608 |
| 46 | 19998 | 9,113208666 | 3,398361656 | 479952 | 468.7 KB | 176620 | 172.48 KB | 1 | 10 | 10 | 12 | 10 | 2,717427245 |
| 47 | 24999 | 11,03234199 | 4,158196283 | 599976 | 585.91 KB | 220760 | 215.59 KB | 1 | 10 | 10 | 12 | 10 | 2,717774959 |
| 48 | 30000 | 13,28480532 | 4,969651089 | 720000 | 703.13 KB | 264899 | 258.69 KB | 1 | 10 | 10 | 12 | 10 | 2,718017056 |
| 49 | 34998 | 16,06435371 | 5,903515225 | 839952 | 820.27 KB | 309011 | 301.77 KB | 1 | 10 | 10 | 12 | 10 | 2,718194498 |

edgebreaker | sequential

Рисунок 9 — Додано розрахування коефіцієнту стиснення у Sequential

Використовуючи цей алгоритм дані про ступінь стиснення також не дуже значно змінюються від кількості вершин, поки не дійдуть до відмітки в 65000.

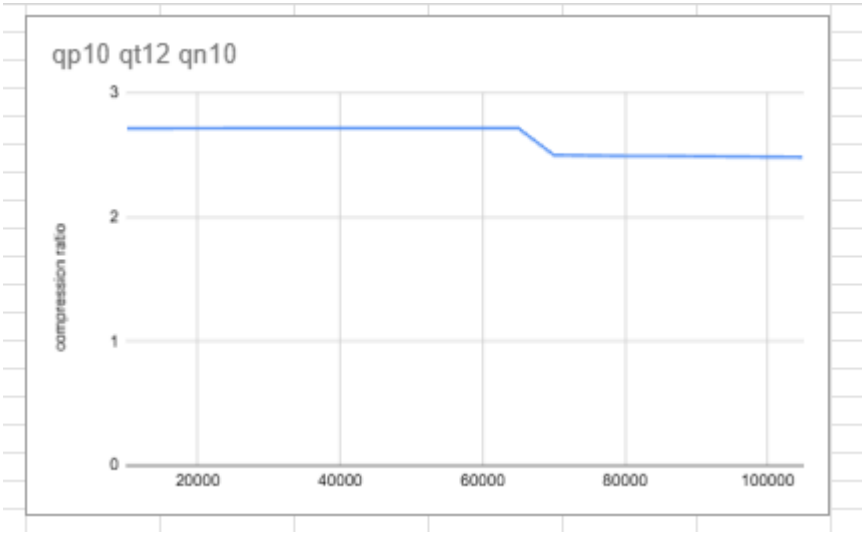



Рисунок 10 — Квантування позиції 10, текстур 12, нормалей 10 алгоритму sequential

Аналіз показав, що ефективність алгоритму Sequential значно змінюється залежно від кількості вершин у моделі. Незважаючи на загальну стабільність коефіцієнта стиснення, при збільшенні кількості вершин до 65 000 спостерігається поступове зростання цього показника, після чого ефективність різко падає. Така поведінка є характерною для моделей, де збільшення складності призводить до зниження продуктивності алгоритму. Графіки, побудовані на основі результатів експериментів, підтверджують наявність переламного моменту на 65 000 вершинах, після якого коефіцієнт стиснення починає знижуватися.

### 4.2.3 Порівняння алгоритмів в контексті ефективності стиснення

| vertices | average compression ratio difference edgebreaker and sequential |
|---|---|
| 9999 | -0,00000285184340 |
| 15000 | -0,00002316229044 |
| 19998 | -0,00000358062846 |
| 24999 | 0,0000258476442 |
| 30000 | 0,00001386821882 |
| 34998 | 0,00002423697608 |
| 39999 | 0,00003451109598 |
| 45000 | 0,000009296132035 |
| 49998 | 0,000009439320427 |
| 54999 | -0,00000177766938 |
| 60000 | 0,000002825248152 |
| 64998 | 0,00004217359583 |
| 69999 | 0,000007182905861 |
| 75000 | -0,00000575373948 |
| 79998 | -0,00000000281921 |
| 84999 | 0,00000729704663 |
| 90000 | 0,000006584133175 |
| 94998 | 0,000003325144584 |
| 99999 | 0,000001465332804 |
| 105000 | 0,00001553034694 |

Рисунок 11 — Різниця в середніх коефіцієнтах стиснення двох алгоритмів в залежності від кількості вершин

Порівняння алгоритмів edgebreaker та sequential демонструє, що обидва підходи мають схожий рівень стиснення за однакових параметрів. Проблема зниження ефективності після 65 000 вершин є спільною для обох алгоритмів. У середньому, різниця між алгоритмами не перевищує допустимої похибки, що дозволяє розглядати їх як еквівалентні в контексті ступеня стиснення.

### 4.2.4 Порівняння параметрів квантування в контексті ефективності стиснення

На основі таблиці можна зробити висновок про особистий вклад зміни конкретного параметра у коефіцієнт стиснення.

| vertices | qp -2 | qt -2 | qn -2 |
|---|---|---|---|
| 9999 | 0,1195037025 | 0,06790824639 | 0 |
| 15000 | 0,1197581367 | 0,06808875401 | 0 |
| 19998 | 0,1199053552 | 0,06820268268 | 0 |
| 24999 | 0,1199734191 | 0,06826999171 | 0 |
| 30000 | 0,1200375331 | 0,06830105099 | 0 |
| 34998 | 0,1200714222 | 0,06833713873 | 0 |
| 39999 | 0,120108691 | 0,06835248488 | 0 |
| 45000 | 0,1201333118 | 0,06837396885 | 0 |
| 49998 | 0,1201424145 | 0,0683812822 | 0 |
| 54999 | 0,1201508073 | 0,06839105305 | 0 |
| 60000 | 0,1201681098 | 0,06840717664 | 0 |
| 64998 | 0,1201849772 | 0,06841331224 | 0 |
| 69999 | 0,101318042 | 0,05860643203 | 0 |
| 75000 | 0,1009815245 | 0,05842981675 | 0 |
| 79998 | 0,1006856186 | 0,05827731649 | 0 |
| 84999 | 0,100433197 | 0,05814701458 | 0 |
| 90000 | 0,1002089071 | 0,05802448719 | 0 |
| 94998 | 0,1000057948 | 0,05792437485 | 0 |
| 99999 | 0,09982083397 | 0,05782805341 | 0 |
| 105000 | 0,09966293143 | 0,05774066772 | 0 |

Рисунок 12 — Середня зміна коефіцієнта стиснення з кроком квантування -2

Результати аналізу показали, що параметри квантування позицій найбільше впливають на коефіцієнт стиснення. Наприклад, зменшення квантування позицій із 14 до 10 для моделі з 15 000 вершинами збільшує коефіцієнт стиснення на 0.24. Утім, для моделей із більш ніж 65 000 вершинами цей ефект зникає, і загальна ефективність знижується. Аналогічна тенденція спостерігається для квантування текстурних координат, тоді як квантування нормалей має мінімальний вплив через обмежене використання нормалей у тестових моделях

Щодо параметра квантування текстурних координат, він також показує що йде збільшення ефективності стискання залежно від кількості вершин аж до показника в 65000 вершин, після чого з кожним додаванням додаткових вершин показник різко погіршується.

Квантування нормалей було додано для дослідження поведінки алгоритмів. В моделях які створюються в рамках цієї роботи нормалі не використовуються, тому й зміна квантування цього показника абсолютно ніяк не впливає на модель, що є правильною поведінкою з боку алгоритмів.

### 4.2.5 Оцінка часу кодування та декодування тривимірних моделей залежно від розміру моделі

Розмір моделі впливає на час обробки для компресії та декомпресії моделей. Ця залежність є доволі прогнозованою, в результатах немає значних просадок в часі, або навіть зміни часу у зворотну сторону.

| vertices | avg time encode | avg time decode | time difference encode | time difference decode |
|---|---|---|---|---|
| 9999 | 5,71651355 | 1,800214378 | 0 | 0 |
| 15000 | 8,398585521 | 2,589203819 | 2,682071971 | 0,7889894415 |
| 19998 | 10,92966972 | 3,362637568 | 2,531084201 | 0,7734337485 |
| 24999 | 13,47588603 | 4,214179269 | 2,546216303 | 0,851541701 |
| 30000 | 16,44139697 | 5,223979086 | 2,965510949 | 1,009799817 |
| 34998 | 18,95745674 | 5,906171453 | 2,516059768 | 0,6821923674 |
| 39999 | 21,51197371 | 6,73826581 | 2,554516971 | 0,8320943572 |
| 45000 | 24,06047492 | 7,526483568 | 2,548501209 | 0,7882177577 |
| 49998 | 26,74206144 | 8,305220331 | 2,681586517 | 0,7787367631 |
| 54999 | 30,37784877 | 9,153463091 | 3,635787328 | 0,8482427597 |
| 60000 | 33,05280088 | 10,08575388 | 2,674952112 | 0,9322907903 |
| 64998 | 35,69353575 | 10,88392753 | 2,640734871 | 0,7981736445 |
| 69999 | 39,51099741 | 13,18520225 | 3,817461657 | 2,301274723 |
| 75000 | 42,02518744 | 14,03763331 | 2,514190035 | 0,8524310573 |
| 79998 | 45,0786484 | 15,15928786 | 3,053460956 | 1,121654556 |
| 84999 | 46,76440224 | 15,72775081 | 1,685753843 | 0,5684629499 |
| 90000 | 49,5581873 | 16,83992777 | 2,793785061 | 1,112176957 |
| 94998 | 52,79848905 | 17,87626205 | 3,240301753 | 1,036334282 |
| 99999 | 55,64351753 | 18,79571003 | 2,845028476 | 0,9194479758 |
| 105000 | 59,89504003 | 19,54375294 | 4,2515225 | 0,7480429174 |

Рисунок 13 — Дослідження часу кодування та декодування в залежності від кількості вершин

У двох останніх стовпцях проходить порівняння зміни часу порівнюючи з попередньою 3D моделлю. Наприклад, від швидкості кодування або декодування моделі з 15000 вершинами віднімається швидкість за тим же показником

у попередньої моделі з 9999 вершинами. Це дозволяє наочно побачити що різниця в результатах майже непомітна та не відповідає ніякій закономірності. Слід тільки виділити аномально низьку зміну швидкості кодування на моделі з 84999 вершинами (зміна в 1,68 мілісекунди), та аномально високу зміну швидкості кодування на моделі з 105000 вершинами (4,25 мілісекунди). В декодуванні виділився лише показник на моделі з 69999 вершинами, зміна часу декодування якого стала найвищою серед всіх інших, зміна у 2.3 мілісекунди, хоча інші приблизно мають різницю в 0.8-1 мілісекунду.

Взагалі зміни досить рівномірні, ці результати підкреслюють важливість збалансування складності моделі з вимогами до продуктивності в практичних застосуваннях. Для випадків використання надзвичайно великих моделей для підтримки ефективності може знадобитися попередня обробка або паралельна обробка. І навпаки, для невеликих моделей Draco забезпечує майже миттєве кодування та декодування, що робить його практичним вибором для веб-застосунків і динамічних середовищ. Цей аналіз додатково підтверджує масштабованість Draco, оскільки вона ефективно керує як малими, так і великими наборами даних із передбачуваними характеристиками продуктивності.

### 4.2.6 Оцінка часу кодування та декодування 3D моделей залежно від параметрів квантування

Різні алгоритми стиснення мають на увазі й різні показники ефективності. Хоча ступінь стиснення залежно від використаного алгоритму майже не змінювалася, час роботи цих алгоритмів суттєво відрізняється:

| vertices | encode(edge.-seq.) | decode(edge.-seq.) | qp |
|---|---|---|---|
| 9999 | -0,2393402496 | 0,01653752377 | |
| 15000 | -0,4050640001 | 0,008159033388 | |
| 19998 | -0,6749162317 | -0,1737500898 | |
| 24999 | -0,9182639243 | -0,04377756946 | |
| 30000 | -1,009040559 | 0,0523874777 | |
| 34998 | -1,491698049 | -0,08374307528 | |
| 39999 | -1,494062695 | -0,2845606464 | |
| 45000 | -1,517414834 | -0,2254184594 | |
| 49998 | -1,849995603 | -0,1085268355 | |
| 54999 | -1,812564342 | -0,2012404778 | |
| 60000 | -2,096353687 | -0,3234396464 | |
| 64998 | -2,361965592 | 0,06104791637 | |
| 69999 | -2,185853277 | -0,1463618323 | |
| 75000 | -2,870523466 | -0,5459802407 | |
| 79998 | -3,682590669 | -0,2350784853 | |
| 84999 | -3,386869023 | -1,054983597 | |
| 90000 | -3,116591199 | -0,1511672648 | |
| 94998 | -3,125922527 | -0,5582467951 | |
| 99999 | -4,559392414 | -0,08145718953 | |
| 105000 | -4,116064067 | -0,4678178405 | |

Рисунок 14 — Різниця між алгоритмами в середньому часі кодування та декодування моделей під час зміни квантування позиції в мілісекундах

Спочатку може здатися, що різниця між цими двома алгоритмами близькі до похибки: edgebreaker на незначний показник під час кодування швидше, а sequential  на незначний показник при декодуванні. Але починаючи з моделі в 19998 вершин sequential  навіть в декодуванні становиться швидшим. Найбільша різниця настає на самих великих моделях: edgebreaker пережинає sequential в обох показниках, а у кодуванні приблизно на 4,5 мілісекунди, що вже є значною різницею. Але це стосується лише зміни квантування позиції, тому що при квантуванні текстурних координат ситуація наступна:

| vertices | encode(edge.-seq.) | decode(edge.-seq.) | qt |
|---|---|---|---|
| 9999 | 0,1382720481 | -0,09347863315 | |
| 15000 | 0,2299488524 | -0,1561634544 | |
| 19998 | 0,6437149716 | -0,1133522696 | |
| 24999 | 0,7164507801 | -0,09373335584 | |
| 30000 | 0,4967853418 | 0,2308672485 | |
| 34998 | 0,7372620524 | -0,2529195863 | |
| 39999 | 0,4029453109 | -0,2910924824 | |
| 45000 | 0,4282810265 | -0,3372346327 | |
| 49998 | 0,866334186 | -0,3607586262 | |
| 54999 | 1,069740066 | -0,3731113465 | |
| 60000 | 1,402258267 | -0,4707372639 | |
| 64998 | 1,515581294 | -0,4449813529 | |
| 69999 | 1,003735659 | -0,586377049 | |
| 75000 | 1,469729195 | -0,6232496629 | |
| 79998 | 2,25978709 | -0,1083216633 | |
| 84999 | 0,5684024585 | -0,8253399424 | |
| 90000 | 1,632841781 | -0,8584762096 | |
| 94998 | 2,379345307 | 0,0436641338 | |
| 99999 | 2,617122211 | -0,7299828136 | |
| 105000 | 2,159023509 | -0,7741026991 | |

Рисунок 15 — Різниця між алгоритмами в середньому часі кодування та декодування моделей під час зміни квантування текстурних координат в мілісекундах

Алгоритм sequential під час декодування став показувати ще гірші результати, але порівнюючи з edgebreaker значно став швидшим під час кодування.

Вплив параметрів квантування на час обробки моделей є значним. Алгоритм edgebreaker забезпечує швидше кодування та декодування при змінених параметрах квантування позиції, тоді як sequential перевершує його на етапі кодування при знижених параметрах квантування текстурних координат. Утім в більшості випадків обидва алгоритми мають схожу продуктивність, що вказує на їхню гнучкість у роботі з різними параметрами

### 4.3 Формування рекомендацій щодо використання технології Draco

Результати досліджень показали, що параметри квантування суттєво впливають на ефективність стиснення моделі, однак різні алгоритми продемонстрували приблизно однаковий ступінь компресії. Для моделей із кількістю вершин до 65 000 квантування позицій та текстур має значно більший вплив, ніж для моделей із більшим числом вершин. При цьому перехід є досить різким, що свідчить про доцільність обмеження розміру моделі в 65 000 вершин для досягнення найкращих результатів.

Щодо алгоритмів стиснення, дослідження виявили, що алгоритм sequential забезпечує більшу швидкість декодування порівняно з edgebreaker, тоді як останній демонструє перевагу під час процесу кодування. Це вимагає чіткого розуміння, на якому етапі (кодування чи декодування) важливіша швидкість обробки. Для сценаріїв, де необхідно швидко стиснути модель та передати її, доцільно використовувати edgebreaker. Натомість для забезпечення швидкого декодування на стороні користувача рекомендовано застосовувати sequential.

Також варто враховувати залежність від якості стиснення. Для моделей, що не потребують високої геометричної точності, наприклад, в ігрових застосунках, доцільно знижувати параметри квантування для досягнення більшого ступеня стиснення, тому що, як показало дослідження, зменшення квантуваня суттєво зменшує розмір моделі.

Таким чином, застосування технології Draco для стиснення 3D-моделей дозволяє суттєво покращити продуктивність веб-застосунків, зменшити обсяг переданих даних, а також оптимізувати процес завантаження й інтерактивної роботи із 3D-графікою.

### 4.4 Висновки з розділу 4

1. Заповнення таблиці купою результатів виконання програми дозволило проводити подальший аналіз використання технології Draco для оптимізації 3D-моделей.

2. Порівняння алгоритмів edgebreaker і sequential показало, що вони мають схожу продуктивність та схожі проблеми. При збільшенні кількості вершин понад 65 000 спостерігається значне збільшення часу обробки, що вказує на зниження продуктивності. Така залежність підкреслює необхідність обмеження кількості вершин для досягнення оптимальних результатів стиснення.

3. Виявлено, що параметри квантування, такі як квантування позицій і текстурних координат, значно впливають на ефективність стиснення. Менші значення квантування забезпечують краще стиснення для моделей з меншою кількістю вершин, однак ефективність знижується для великих моделей. Рекомендується використовувати оптимальні параметри квантування для досягнення кращого компресії без суттєвих втрат в якості моделі.

4. Виявлено що sequential порівнюючи з edgebreaker став значно швидшим під час кодування, але повільнішим при декодуванні під час роботи з квантуванням текстурних координат, а за іншими параметрами перевага за edgebreaker.

# ВИСНОВКИ

1. Досліджена проблема стиснення 3D моделей в контексті веб-розробки: необхідність заощаджувати трафік та пропускну здатність мережі. Також було визначено обмеження та проблеми, пов’язані з обробкою в реальному часі та сумісністю між платформами.

2. Проаналізовано різні технології стиснення 3D-моделей, з особливим наголосом на Draco та його конкурентах. Це порівняльне дослідження продемонструвало переваги стиснення на основі геометрії та обговорило передові методи, такі як квантування та оптимізація топології.

3. Було детально описано дизайн і впровадження програми для аналізу технологій стиснення. Завдяки збору та порівнянню даних про ефективність у структурованих форматах дослідження запропонувало практичне розуміння компромісів між ефективністю стиснення, часом обробки та сумісністю.

4. Алгоритми edgebreaker і sequential продемонстрували різну ефективність залежно від етапу обробки. edgebreaker виявився ефективнішим для декодування, тоді як sequential забезпечує швидше кодування. Зменшення точності квантування дозволяє досягти високого ступеня стиснення, але може негативно вплинути на якість моделі та час обробки. При цьому, для моделей із кількістю вершин понад 65 000 ефективність стиснення помітно знижується, що вказує на необхідність обмеження розміру моделей для забезпечення стабільної продуктивності.

5. Загалом дослідження прийшло до висновку, що оптимізація 3D-моделей має вирішальне значення не лише для ефективності роботи програми, але й для забезпечення залучення користувачів у середовищах з обмеженими ресурсами.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Boutsi A.-M., Ioannidis C., Verykokou S. Multi-Resolution 3D Rendering for High-Performance Web AR. *Sensors*. 2023. Т. 23, № 15. С. 6885. URL : https://doi.org/10.3390/s23156885 (дата звернення: 05.11.2024).

2. Delivering and using 3D models on the web: are we ready? / R. Scopigno та ін. *Virtual Archaeology Review*. 2017. Т. 8, № 17. С. 1. URL : https://doi.org/10.4995/var.2017.6405 (дата звернення: 05.11.2024).

3. Riwinoto R., Putra C. L. Optimization of VR Application Mahakarya Vokasi Land Using Level of Detail (LOD) Technique. *Journal of applied multimedia and networking*. 2023. Т. 7, № 1. С. 90—95. URL : https://doi.org/10.30871/jamn.v7i1.5681 (дата звернення: 05.11.2024).

4. Optimizing Art and Technology in 3D Character Creation: Strategies for High-Fidelity Modeling and Resource Efficiency in Real-Time Environments: RSUCON, 26 серпня 2024р.

5. A Model Simplification Algorithm for 3D Reconstruction / Z. Liu та ін. *Remote Sensing*. 2022. Т. 14, № 17. С. 4216. URL : https://doi.org/10.3390/rs14174216 (дата звернення: 05.11.2024).

6. 3D graphics on the web: A survey / A. Evans та ін. *Computers & Graphics*. 2014. Т. 41. С. 43—61. URL: https://doi.org/10.1016/j.cag.2014.02.002 (дата звернення: 13.11.2024).

7. Koskela T., Vatjus-Anttila J. Optimization Techniques for 3D Graphics Deployment on Mobile Devices. *3D Research*. 2015. Т. 6, № 1. URL : https://doi.org/10.1007/s13319-015-0040-0 (дата звернення: 13.11.2024).

8. Грицай Р. Методи стиснення інформації. *Природничі та гуманітарні науки. Актуальні питання*: збірник матеріалів VII Всеукраїнської студентської науково-технічної конференції (м. Тернопіль, 24-25 квітня 2014 р.). Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2014. С. 55

9. Garland M., Heckbert P. S. Surface Simplification Using Quadric Error Metrics. *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. New York, NY, USA, 2023. С. 131—138. URL : https://doi.org/10.1145/3596711.3596727 (дата звернення: 14.11.2024).

10. Progressive Compression of Triangle Meshes / V. Vidal та ін. *Image Processing On Line*. 2023. Т. 13. С. 1—21. URL : https://doi.org/10.5201/ipol.2023.418 (дата звернення: 16.11.2024).

11. A Psychophysical Evaluation of Texture Compression Masking Effects / G. Lavoue та ін. *IEEE Transactions on Visualization and Computer Graphics*. 2019. Т. 25, № 2. С. 1336—1346. URL : https://doi.org/10.1109/tvcg.2018.2805355 (дата звернення: 07.12.2024).

12. VEND: Vertex Encoding for Edge Nonexistence Determination / Y. Li та ін. *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, м. Anaheim, CA, USA, 3—7 квіт. 2023 р. 2023. URL : https://doi.org/10.1109/icde55515.2023.00032 (дата звернення: 16.11.2024).

13. Tang Z., Ye H., Zhang H. Multi-Scale Point Octree Encoding Network for Point Cloud Based Place Recognition. *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, м. Detroit, MI, USA, 1—5 жовт. 2023 р. 2023. URL : https://doi.org/10.1109/iros55552.2023.10341943 (дата звернення: 16.11.2024).

14. Liu C. C. Adaptive Contrast Enhancement of Optical Imagery Based on Level of Detail (LOD). *Remote Sensing*. 2020. Т. 12, № 10. С. 1555. URL : https://doi.org/10.3390/rs12101555 (дата звернення: 18.11.2024).

15. Performance Comparison of Meshlet Generation Strategies: Journal of Computer Graphics Techniques, 12 серпня 2023р. С. 1-27

16. J. Arvo, A. Euranto, L. Järvenpää, T. Lehtonen, T. 3D Mesh simplification: a survey of algorithms and CAD model simplification tests. *Knuutila University of Turku Technical Reports*, 2015, No.3. С. 1—57. URL :

https://www.academia.edu/84002211/3D_Mesh_Simplification_A_survey_of_algorithms_and_CAD_model_simplification_tests (дата звернення: 18.11.2024)

17. Research on Lightweighting and Rendering Optimization Techniques for Building Information Models / W. Tong та ін. *Journal of Networking and Network Applications*. 2023. Т. 3, № 2. С. 89—98. URL : https://doi.org/10.33969/j-nana.2023.030205 (дата звернення: 21.11.2024).

18. Optimization of 3D Immunofluorescence Analysis and Visualization Using IMARIS and MeshLab / Z. H. Abu Bakar та ін. *Cells*. 2023. Т. 12, № 2. С. 218. URL : https://doi.org/10.3390/cells12020218 (дата звернення: 21.11.2024).

19. Pacheco-Gutierrez S., Caliskanelli I., Skilton R. Point Cloud Compression and Transmission for Remote Handling Applications. *Journal of Software*. 2021. С. 14—23. URL : https://doi.org/10.17706/jsw.16.1.14-23 (дата звернення : 21.11.2024).

20. Draco: Efficient Resource Management for Resource-Constrained Control Tasks / P. Marti та ін. *IEEE Transactions on Computers*. 2009. Т. 58, № 1. С. 90—105. URL : https://doi.org/10.1109/tc.2008.136 (дата звернення : 23.11.2024).

21. Finley M. G., Bell T. Depth range reduction for 3D range geometry compression. *Optics and Lasers in Engineering*. 2021. Т. 138. С. 106457. URL : https://doi.org/10.1016/j.optlaseng.2020.106457 (дата звернення : 23.11.2024).

22. Deshko I., Tsvetkov V. Journal of Physics: conference Series, Boston 2022, 28 May — 2 June. Boston, 2022. С. 1—5. URL : http://dx.doi.org/10.1088/1742-6596/2388/1/012024 (дата звернення : 30.11.2024).

23. J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes, IEEE Transactions on Visualization and Computer Graphics. 1999. Vol. 5, No. 1 С. 1—15. URL : https://faculty.cc.gatech.edu/~jarek/papers/EdgeBreaker.pdf (дата звернення : 30.11.2024).

24. Position-based Scaled Gradient for Model Quantization and Pruning / Jangho Kim та ін. 2020. URL: https://doi.org/10.48550/arXiv.2005.11035 (дата звернення : 02.12.2024).

25. Michael Deering. Geometry compression. SIGGRAPH 95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. 1995, С. 13—20. URL : https://dl.acm.org/doi/10.1145/218380.218391 (дата звернення : 02. 12.2024)

26. Pérez-Delgado M.-L., Celebi M. E. A comparative study of color quantization methods using various image quality assessment indices. *Multimedia Systems*. 2024. Т. 30, № 1. URL : https://doi.org/10.1007/s00530-023-01206-7 (дата звернення : 05.12.2024).

27. Пальчевський Б. О., Валецький, Б. П., Вараніцький Т. Л. Системи 3D моделювання : навчальний посібник. Луцьк, 2016. 176с.

28. Lemoine W., Wijnants M. Progressive Network Streaming of Textured Meshes in the Binary glTF 2.0 Format. *Web3D '23: The 28th International ACM Conference on 3D Web Technology*, м. San Sebastian Spain. New York, NY, USA, 2023. URL : https://doi.org/10.1145/3611314.3615907 (дата звернення : 05.12.2024).

29. Johnstone J. A Category-Major Interoperable Data Format for Optic Morphometry. *CAD'24*. 2024. URL : https://doi.org/10.14733/cadconfp.2024.121-125 (дата звернення : 05.12.2024).

30. Bayraktar E., Wang Y., DelBue A. Fast re-OBJ: real-time object re-identification in rigid scenes. *Machine Vision and Applications*. 2022. Т. 33, № 6. URL : https://doi.org/10.1007/s00138-022-01349-z (дата звернення: 05.12.2024).

31. Береговой К.В, Полякова Н.П доцент, канд. техн. наук, науковий керівник. Особливості використання технології Draco для оптимізації графічних даних в контексті Web-розробки. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «Молода наука-2024» / Запорізький національний університет. – Запоріжжя : ЗНУ, 2024. – Т.5. – С. 166-167.

32. Береговой К.В, Полякова Н.П доцент, канд. техн. наук,науковий керівник. Аспекти застосування технології Draco для оптимізації графічної інформації в контексті розробки Web-застосунків. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених І Міжнародна науково-практична конференція «Інженерні інновації та розбудова національної економіки» / Запорізький національний університет. – Запоріжжя : ЗНУ, 2024. – С. 602-604.

**Декларація**

**академічної доброчесності**

**здобувача ступеня вищої освіти ЗНУ**

Я, Береговой Кіріл Вікторович , студент 2 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz19bd-02@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Особливості використання технології Draco для оптимізації графічних даних в контексті Web-розробки**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 12.12.2024 Підпис _____ Береговой Кіріл Вікторович
(студент)

Дата 12. 12. 2024 Підпис _____ Полякова Наталія Петрівна
(науковий керівник)