МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,
д-р техн. наук, проф.

_____ І. М. Журавська

« __ » _____ 202__ р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

СИСТЕМА СТИСНЕННЯ ЗОБРАЖЕНЬ НА БАЗІ АЛГОРИТМУ QOI

Спеціальність «Комп'ютерна інженерія»
123 – КМР.1 – 605.21710510

Студент _____Є. В. Дзяман
підпис

«__» _____ 202__ р.

Керівник канд. техн. наук, доцент _____Я. М. Крайник
підпис

«__» _____ 202__ р.

Миколаїв – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CR	–	Compression Rate
FPGA	–	Field-Programmable Gate Array
JPEG	–	Joint Photographic Experts Group
PNG	–	Portable Network Graphics
QOI	–	Quite Ok Image
RLE	–	Run-Length encoding
GIF	–	Graphics Interchange Format

ВСТУП

Стиснення зображення – це процес, застосований до графічного файлу для мінімізації його розміру в байтах без погіршення якості зображення нижче прийнятного порогу. Зменшивши розмір файлу, можна зберігати більше зображень на заданому об’ємі диска чи пам’яті. Стиснене зображення також вимагає меншої пропускної здатності під час передачі через Інтернет або завантаження з веб-сторінки, що зменшує навантаження мережі та пришвидшує доставку вмісту.

Визначення найкращого формату зображення без втрат для конкретного застосування – це процес, який передбачає вивчення багатьох змінних для прийняття обґрунтованого рішення. Від ступеня стиснення, якого кодек здатний досягти для певного типу зображення, до алгоритмічної складності, швидкості чи вимог до пам’яті для програмного застосунку, до вимог до розміру та потужності для апаратної реалізації – усе це є вирішальними факторами, які формуватимуть форму збереження, кінцевий продукт.

Мета: підвищення адаптивності мультимедійних систем відносно вимог щодо складності та ступеню стиснення зображення за рахунок використання алгоритму стиснення QOI та його модифікацій.

Об’єкт: системи та засоби стиснення мультимедійних даних у комп'ютерних системах.

Предмет: система стиснення зображень на базі алгоритму QOI та його модифікацій.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

– виконати огляд предметної області та визначити можливості для застосування алгоритму QOI у мультимедійних системах, зокрема, стиснення зображень;

– дослідити характеристики алгоритму стиснення QOI та визначити сильні та слабкі сторони алгоритму та вказати можливі варіанти модифікацій;

– виконати модифікації алгоритму та дослідити отримані характеристики з точки зору обчислювальної складності та показників стиснення у порівнянні з початковим варіантом та з іншими форматами;

– розробити програмне забезпечення для виконання перевірки та розглянути апаратні платформи, де може використовуватись досліджуваний алгоритм;

– представити отримані результати у вигляді таблиць та графіків порівняння та надати рекомендації з приводу застосування модифікацій у конкретних ситуаціях.

Практичне значення отриманих результатів: розроблений програмний модуль може бути рекомендовано до застосування у вбудованих системах із обмеженою кількістю оперативної пам’яті а також для передачі мультимедійних стиснених даних за допомогою мережевих інтерфейсів.

Апробація результатів магістерської роботи відбулася в рамках Всеукраїнської науково-практичної конференції молодих вчених, аспірантів і студентів «Інформаційні технології та інженерія» (Миколаїв, ЧНУ ім. Петра Могили, 07-10 лютого 2023 р.).

Публікації. За результатами кваліфікаційної роботи створено публікацію у збірнику матеріалів Всеукраїнської конференції [1].

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРИ ТА ПАТЕНТНОЇ ІНФОРМАЦІЇ

1.1 Стиснення зображень

Постійна тенденція збільшення роздільної здатності зображення в більшості комерційних і професійних застосунків вимагає постійного розвитку набору рішень, які полегшують ефективне кодування зображень, зменшуючи їх розмір без шкоди для якості. Від найсучаснішого відео 4K і 8K до програм із наднизьким енергоспоживанням, співвідношення між розміром необроблених даних зображення та розміром закодованого зображення вважається одним із основних показників для визначення ефективності кожного формату, оскільки це вирішальний фактор для вимог програми до зберігання та пропускної здатності у випадку обробки в реальному часі.

Стиснення зображень – це метод, який використовується для зменшення розміру зображень, що може покращити швидкість завантаження та загальну продуктивність веб-сайту.

Методи, які використовуються для стиснення файлів зображень, зазвичай, належать до однієї з двох категорій: із втратами та без втрат. Стиснення із втратами зменшує розмір файлу зображення, остаточно видаляючи менш важливу інформацію, зокрема зайві дані. Стиснення з втратами може значно зменшити розмір файлу, але також може знизити якість зображення до рівня спотворення, особливо якщо зображення надто стиснене. Однак якість можна зберегти, якщо стиснення застосовано ретельно.

Ступінь стиснення може відрізнятися залежно від формату зображення, а також тип даних зображення. Вибір відповідного формату для цільового типу даних може стати вирішальним для продуктивності всієї програми. З появою нових форматів зображень, таких як QOI, формат Quite

OK Image [6], разом із безперервним удосконаленням існуючих форматів і реалізацій, вимірювання ефективності кожного формату для наборів даних, що відображають значну дисперсію типів зображень, є корисний посібник для визначення правильного кодека для сполучення з кожною програмою.

Однією з проблем стиснення з втратами є його незворотність. Після того, як його було застосовано до зображення, це зображення ніколи не можна відновити до початкового стану. Якщо стиснення з втратами багаторазово застосовується до того самого зображення, воно дедалі більше спотворюється. Тим не менш, стиснення з втратами даних виявилося цінною стратегією для Інтернету, де часто допускається помірне погіршення якості зображення.

Найпоширенішим прикладом стиснення з втратами є JPEG, формат стиснення зображень, який широко використовується в Інтернеті та цифровій фотографії. Одна з великих переваг JPEG полягає в тому, що він дозволяє дизайнеру точно налаштувати ступінь стиснення. Це призводить до кращої якості зображення при правильному використанні, а також призводить до найменшого прийнятного розміру файлу. Оскільки JPEG використовує стиснення з втратами, зображення, збережені в цьому форматі, схильні до «артефактів», коли можна побачити пікселізацію та дивні ореоли навколо певних частин зображення. Найчастіше вони трапляються в областях зображення, де є різкий контраст між кольорами. Як правило, що більший контраст має зображення, то вища якість зображення має бути збережена, щоб отримати гідне кінцеве зображення [13].

Цей загальновизнаний формат підтримується численними інструментами та програмами. Крім того, стиснення можна застосовувати в градусах, що дає змогу використовувати стиснення JPEG, яке найкраще забезпечує баланс між розміром файлу та якістю. На рис. 1.1 показані ступені стиснення зображення з допомогою JPEG.

Рисунок 1.1 – Стиснення JPEG

Інший підхід до стиснення зображення називається стиснення без втрат. Цей метод застосовує стиснення без видалення критичних даних або зниження якості зображення, що призводить до стисненого зображення, яке можна відновити до початкового стану без погіршення чи спотворення. Однак стиснення без втрат не зменшує розмір файлу майже так само, як стиснення з втратами, пропонуючи невелику перевагу з точки зору місця для зберігання, пропускної здатності мережі або швидкості завантаження. Стиснення без втрат, зазвичай, використовується в ситуаціях, коли якість зображення важливіша за простір на диску чи продуктивність мережі, наприклад для зображень продуктів або для демонстрації творів мистецтва.

Одним із найпоширеніших форматів без втрат є PNG, широко використовуваний формат, який зменшує розмір файлу шляхом визначення шаблонів і стиснення цих шаблонів разом.

PNG (Portable Network Graphics) — ще один формат растрового зображення, який використовує стиснення даних без втрат і був створений для заміни формату зображення GIF. Формат PNG тривалий час практично не підтримувався Internet Explorer, через що він використовувався рідше, ніж формати GIF і JPEG, хоча зараз він належним чином підтримується всіма основними браузерами. Файли PNG підтримують кольори на основі палітри (24-розрядний RGB або 32-розрядний RGBA), градації сірого, колірні простори RGBA та RGB. Однією з найбільших переваг PNG є те, що він

підтримує низку параметрів прозорості, включаючи прозорість альфа-каналу (рис. 1.2).

Рисунок 1.2 – Стиснення PNG

Хоча файли PNG, як правило, більші за файли JPEG, веб-сайти широко використовують їх, коли потрібно більше деталей зображення, наприклад для логотипів, значків, скріншотів або зображень із текстом. Іншим знайомим форматом без втрат є BMP, власний підхід до стиснення зображень, запроваджений Microsoft і використовується переважно для продуктів Microsoft, зокрема для комп’ютерів Windows.

GIF — це формат стиснення, який відноситься до категорії без втрат, хоча існує певна плутанина щодо того, чи це формат стиснення, чи без втрат. GIF розшифровується як Graphics Interchange Format і є форматом растрових зображень, представленим CompuServe у 1987 році. Він підтримує до 8 біт на піксель, тобто зображення може мати до 256 різних кольорів RGB. Однією з найбільших переваг формату GIF є те, що він дозволяє анімовані зображення.

Зображення GIF обмежені 256 кольорами, тому перетворення зображення з більшою кількістю кольорів у GIF призводить до втрати якості, що іноді пояснюється стисненням із втратами. Але алгоритми стиснення, які використовує GIF, є без втрат. Якщо якість втрачається, це пов’язано з

проблемами, пов’язаними з перетворенням файлу. В даний час формат GIF використовується переважно для простих відео та анімацій.

1.1.1 Стиснення flate/deflate

Алгоритм стиснення deflate без втрат базується на двох інших алгоритмах стиснення: кодуванні Хаффмана та стисненні LZ77. Потрібно розуміти як працюють ці два алгоритми, щоб зрозуміти стиснення deflate [10].

Deflate — це розумний алгоритм, який адаптує спосіб стиснення даних до самих фактичних даних. Компресор має три режими стиснення:

– Зовсім не стиснутий. Це розумний вибір для даних, які вже стиснено. Дані, що зберігаються в цьому режимі, трохи розширяться, але не так сильно, як якщо б вони вже були стиснуті та використали один із інших методів стиснення.

– Стиснення, спочатку з LZ77, а потім з трохи модифікованою версією кодування Хаффмана. Дерева, які використовуються для стиснення в цьому режимі, визначаються самою специфікацією deflate, тому для зберігання цих дерев не потрібно займати додатковий простір.

– Стиснення, спочатку за допомогою LZ77, а потім за допомогою дещо модифікованої версії кодування Хаффмана з деревами, які компресор створює та зберігає разом із даними.

Дані розбиваються на «блоки», і кожен блок використовує один режим стиснення. Якщо компресор бажає перейти від нестиснутого сховища до стиснення за допомогою дерев, визначених специфікацією, або до стиснення за допомогою вказаних дерев Хаффмана, або до стиснення за допомогою іншої пари дерев Хаффмана, поточний блок має бути завершено та розпочато новий.

1.1.2 Стиснення JPEG

JPEG означає Joint Photographic Experts Group, яка є комітетом стандартизації. Це також розшифровується як алгоритм стиснення, винайдений цим комітетом. Алгоритм JPEG був розроблений для максимального зменшення розміру файлу природних фотографічних зображень у справжніх кольорах, не впливаючи на якість зображення, яке відчуває сенсорна система людини.

Стиснення JPEG відбувається з втратами: коли розпаковується стиснене зображення JPEG і порівнюється результат із вихідним зображенням, яке було стиснено, у розпакованому зображенні будуть відмінності та втрата деталей і різкості. Інші алгоритми стиснення, такі як LZW, працюють без втрат: розпаковане зображення ідентичне вихідному зображенню.

Людина легше сприймає невеликі зміни яскравості, ніж невеликі зміни кольору. Людська зорова система також не дуже добре може впоратись з високочастотними даними – якщо деталі зображення змінюються дуже швидко в частині зображення, мозок не може розрізняти чітко всі ці зміни. Ці два аспекти сприйняття використовуються в стисненні JPEG для зменшення розміру файлу зображення.

Стиснення JPEG не використовує єдиний фіксований спосіб стиснення даних. Алгоритм може сильно стискати дані зображення зі значною втратою деталей або обмежити втрату деталей шляхом зниження коефіцієнта стиснення. У таких програмах, як Adobe Photoshop, повзунок дозволяє вибрати ступінь стиснення.

Поєднуючи кілька алгоритмів стиснення, JPEG досягає чудових коефіцієнтів стиснення. Можна легко стиснути файл до однієї п’ятої його початкового розміру. Для веб-публікацій або обміну електронною поштою можна досягти ще кращих співвідношень до 20 до 1. Декомпресія JPEG підтримується в протоколах RIP PostScript рівня 2 і 3. Це означає, що файли

меншого розміру можна надсилати через мережу до RIP, що швидше звільняє станцію надсилання, мінімізує витрати на сервер та прискорює RIP.

1.1.3 Алгоритм стиснення LZ77

LZ77 був оголошений у 1977 році та названий основою багатьох інших алгоритмів стиснення без втрат. Зазвичай він використовує метод «ковзного вікна». У цій методології він піклується про словник, який використовує трійки для представлення:

– Зсув — це можна назвати фактичним початком фрази та початком файлу.

– Довжина — визначається як кількість символів, які допомагають у створенні фрази.

– Відхиляючі символи — це маркетологи, які вказують нову фразу.

Він містить вказівку на те, що використана фраза повністю відповідає вихідній фразі, а також визначає, чи є інший символ. Під час аналізу файлу словник динамічно оновлюється для відображення вмісту та розміру стиснутих даних.

1.1.4 Алгоритм стиснення LZW

Стиснення LZW замінює рядки символів окремими кодами. Він не аналізує вхідний текст. Замість цього він просто додає кожен новий рядок символів, який бачить, до таблиці рядків. Стиснення відбувається, коли замість рядка символів виводиться один код. Код, який виводить алгоритм LZW, може мати будь-яку довільну довжину, але він повинен містити більше бітів, ніж один символ. Перші 256 кодів (при використанні восьмибітних символів) за замовчуванням призначаються стандартному набору символів. Решта кодів призначаються рядкам під час виконання алгоритму. Зразок програми працює, як показано, з дванадцятибітовими кодами. Це означає, що коди 0-255 стосуються окремих байтів, тоді як коди 256-4095 стосуються підрядків.

Переваги і недоліки:

– Стиснення LZW найкраще працює для файлів, які містять багато даних, що повторюються. Це часто трапляється з текстом і монохромними зображеннями. Файли, які стиснуті, але взагалі не містять жодної повторюваної інформації, можуть навіть збільшуватися.

– Стиснення LZW відбувається швидко.

– LZW є досить старою технікою стиснення. Усі останні комп’ютерні системи мають потужність, щоб використовувати більш ефективні алгоритми.

Деякі версії стиснення LZW захищені авторським правом. Їх власник Unisys вимагає роялті від будь-якої компанії, що використовує їхній алгоритм. Це серйозно вплинуло на популярність стиснення LZW, і в довгостроковій перспективі, ймовірно його замінять менш дорогі або безкоштовні алгоритми. Кінцевий користувач не повинен хвилюватися, оскільки лише виробники програмного забезпечення мають платити ліцензійні збори.

1.1.5 Алгоритм стиснення RLE

RLE означає Run Length Encoding. Це алгоритм без втрат, який пропонує пристойні коефіцієнти стиснення лише для певних типів даних.

RLE є, мабуть, найпростішим алгоритмом стиснення. Він замінює послідовності однакових значень даних у файлі числом підрахунку та одним значенням. Припустимо, слід стиснути наступний рядок даних (17 байт): ABBBBBBBBBCDEEEEF Використовуючи стиснення RLE, стиснутий файл займає 10 байт і може виглядати так: A *8B CD *4E F.

Як бачимо, рядки даних, що повторюються, замінюються контрольним символом (*), за яким іде кількість повторюваних символів і сам символ, що повторюється. Контрольний характер не є фіксованим, він може відрізнятися від реалізації до реалізації. Якщо сам контрольний символ з’являється у файлі, кодується один додатковий символ. Кодування RLE є ефективним,

лише якщо є послідовності з 4 або більше символів, що повторюються, оскільки три символи використовуються для проведення RLE, тому кодування двох символів, що повторюються, навіть призведе до збільшення розміру файлу. Важливо знати, що існує багато різних схем кодування довжини серії. Наведений вище приклад використовувався для демонстрації основного принципу кодування RLE. Іноді реалізацію RLE адаптують до типу даних, які стискаються.

Цей алгоритм дуже простий у реалізації та не вимагає великої потужності процесора. Стиснення RLE ефективне лише для файлів, які містять багато даних, що повторюються. Це можуть бути текстові файли, якщо вони містять багато проміжків для відступів, але штрихові зображення, які містять великі білі або чорні області, є набагато придатнішими. Кольорові зображення, створені комп’ютером (наприклад, архітектурні креслення), також можуть забезпечувати справедливі коефіцієнти стиснення.

1.2 Засоби стиснення зображень

Стиснення зображень без втрати якості є одним із найважливіших факторів, які слід враховувати під час пошуку інструменту стиснення, і Wondershare UniConverter не лише виконує цю вимогу, але й багато інших. Це програмне забезпечення для Windows і Mac легко та швидко інсталювати, після чого воно готове допомогти вам виконувати численні завдання, включаючи стиснення файлів JPG. Пакетна обробка, стиснення якості без втрат, редагування та швидка обробка є одними з інших додаткових переваг використання програмного забезпечення.

Для роботи потрібно запустити у своїй системі програмне забезпечення Wondershare і в розділі Toolbox вибрати Image Converte Натиснiути піктограму «Додати зображення» у верхньому лівому куті, щоб переглянути та імпортувати зображення, які потрібно стиснути. Крім того, можна натиснути на знак + у центрі, щоб додати зображення. Додані файли з’являться в інтерфейсі, і за замовчуванням усі файли будуть вибрані. У

верхньому правому куті натиснути на значок шестірні, і відкриється вікно налаштування якості. Зняти прапорець Зберігати оригінальний розмір. Щоб стиснути зображення, можна вибрати нестандартний розмір або навіть перемістити повзунок, щоб вибрати бажану якість для стиснення.

Caesium Image Compressor. Це безкоштовне програмне забезпечення для Windows, яке допомагає стискати зображення у форматах JPG, PNG, JPEG і BMP. За допомогою програми можна вибрати необхідний рівень стиснення. Інструмент також дає вам можливість перевірити, скільки місця ви зберегли за допомогою стиснення.

– Для стиснення JPEG без втрат можна вибрати різні рівні стиснення;

– Можна перевірити попередній перегляд оригінального та стисненого зображення;

– Збережений простір для зберігання можна перевірити;

– Доступно лише для системи Windows;

– Необхідно завантажити та встановити програмне забезпечення.

Mass Image Compressor. Для оптимізації зображень, проблем із зберіганням та інших цілей це гідний інструмент стиснення, доступний для завантаження в Windows. Якість і розмір зображень можна змінити більш ніж на 90%, щоб стиснути або зменшити розмір. Процес зберігає метадані, а також підтримується пакетна обробка.

– Підтримується пакетна обробка;

– Метадані зображення зберігаються (теги EXIF);

– Підтримка форматів JPEG, PNG і RAW;

– Підтримка тільки для системи Windows;

– Інтерфейс не дуже вражає.

TinyPNG. Це безкоштовний онлайн-інструмент, який працює у вашому браузері та дозволяє стискати зображення у форматах JPEG, PNG і WebP. Інструмент дозволяє обробляти до 20 зображень одночасно з максимальним

розміром 5 МБ кожне. Оптимальна стратегія та кодування використовуються для автоматичного аналізу та стиснення зображень. Інтерфейс інструменту також показує, скільки місця зберігається завдяки стисненню.

– Безкоштовний онлайн-інструмент без необхідності завантажувати програмне забезпечення;

– Стислі зображення можна завантажити локально або зберегти в Dropbox;

– Можливість додавати 20 зображень за раз;

– Не можна вибрати параметри якості або розміру для стиснення;

– Максимальний розмір зображення обмежено 5 Мб.

JPEG Optimizer. Застосунок дозволяє завантажувати та стискати фотографії в Інтернеті (рис. 1.3). Цей простий інструмент працює, як випливає з його назви, лише з файлами JPEG.

Рисунок 1.3 – JPEG Optimizer

Що чудово в JPEG Optimizer, так це те, що він дозволяє змінювати розміри зображень перед оптимізацією. Зміна розміру зображень заощадить час завантаження, а з цією онлайн-платформою не доведеться розділяти робочий процес на два етапи.

JPEG Optimizer також дозволяє вибрати власний рівень оптимізації, що дозволяє контролювати якість оптимізованого зображення. Ця функція особливо важлива для фотографів, оскільки можна знайти оптимальне співвідношення між збереженням якості та економією місця.

1.3 Техніка стиснень зображень

У сфері обробки зображень стиснення зображень є важливим кроком перед початком обробки більших зображень або відео. Стиснення зображень виконується кодером і виводить стиснуту форму зображення. У процесах стиснення математичні перетворення відіграють життєво важливу роль. Блок-схему процесу стиснення зображення зображено на рис. 1.4.

Рисунок 1.4 – Блок-схему процесу стиснення зображення

Загальне представлення зображення в комп’ютері схоже на вектор пікселів. Кожен піксель представлений фіксованою кількістю бітів. Ці біти визначають інтенсивність кольору (у градаціях сірого, якщо чорно-біле зображення, і має три канали RGB, якщо зображення кольорові).

Розглянемо чорно-біле зображення з роздільною здатністю 1000*1000, і кожен піксель використовує 8 біт для представлення інтенсивності. Отже, загальна кількість необхідних бітів = 1000*1000*8 = 80 00 000 біт на зображення. Тоді загальна кількість бітів для відео тривалістю 3 секунди дорівнює: 3*(30*(8 000 000))=720 000 000 біти.

Щоб лише зберегти 3-секундне відео, потрібно дуже багато бітів. Отже, потрібен спосіб мати належне представлення, а також зберігати

інформацію про зображення в мінімальній кількості бітів без втрати характеру зображення. Таким чином, стиснення зображення відіграє важливу роль. Основні етапи стиснення зображення:

– Застосування трансформації зображення;
– Квантування рівнів;
– Кодування послідовностей.

1.1.6 Трансформація зображень

Перетворення – це функція, яка відображає один домен (векторний простір) в інший домен (інший векторний простір). Припустимо, T є перетворенням, f(t):X->X' є функцією, тоді T(f(t)) називається перетворенням функції. У простому розумінні можна сказати, що T змінює форму (подання) функції, оскільки це відображення з одного векторного простору в інший (без зміни основної функції f(t), тобто зв’язку між доменом і співобластю). Зазвичай виконується перетворення функції з одного векторного простору в інший, оскільки коли це в новому спроектованому векторному просторі, отримуємо більше інформації про функцію. Реальний приклад трансформації зображено на рис. 1.5.

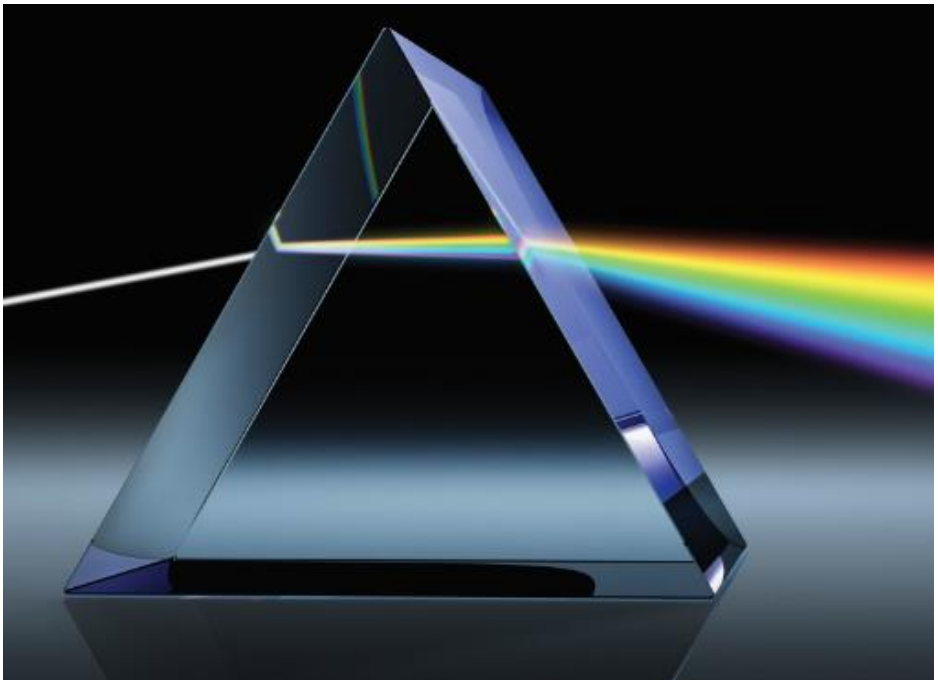

Рисунок 1.5 – Реальний приклад трансформації

Тут можна сказати, що призма є функцією перетворення, в якій вона розділяє біле світло (f(t)) на його компоненти, тобто представлення білого світла. І ми спостерігаємо, що ми можемо вивести більше інформації про світло в його компонентному представленні, ніж біле світло. Ось як перетворення допомагають ефективно розуміти функції.

1.1.7 Перетворення в обробці зображень

Зображення також є функцією розташування пікселів. тобто I(x, y), де (x, y) – координати пікселя на зображенні. Тому зазвичай перетворюється зображення з просторової області в частотну. Чому важлива трансформація зображення:

– Стає легко дізнатися, які основні компоненти складають зображення та допомагають у стиснутому представленні;

– Це полегшує обчислення.

Приклад пошуку згортки в часовій області до перетворення:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Знаходження згортки в частотній області після перетворення:

$$(f * g)(t) = F(s)G(s)$$

Можна побачити, що вартість обчислень зменшилася, оскільки перейшли до частотної області. Також бачимо, що в часовій області згортка була еквівалентною оператору інтеграції, але в частотній області вона дорівнює простому добутку доданків. Таким чином, вартість обчислень зменшується. Таким чином, коли перетворюється зображення з домену в інший, виконання операцій просторової фільтрації стає легшим.

Процес квантування є життєво важливим кроком, під час якого різні рівні інтенсивності групуються в певний рівень на основі математичної функції, визначеної для пікселів. Як правило, новіший рівень визначається фіксованим розміром фільтра «m», діленням кожного з членів «m» фільтра та

округленням його найближчого цілого числа та повторним множенням на «m» [11]. Основна функція квантування:

$$[pixelvalue/m] * m$$

Отже, найближче зі значень пікселів наближається до одного рівня, отже, оскільки кількість різних рівнів, задіяних у зображенні, стає меншою. Таким чином, зменшується надмірність у рівні інтенсивності. Це значить, що квантування допомагає зменшити різні рівні.

Приклад для m=9 наведено на рис. 1.6.

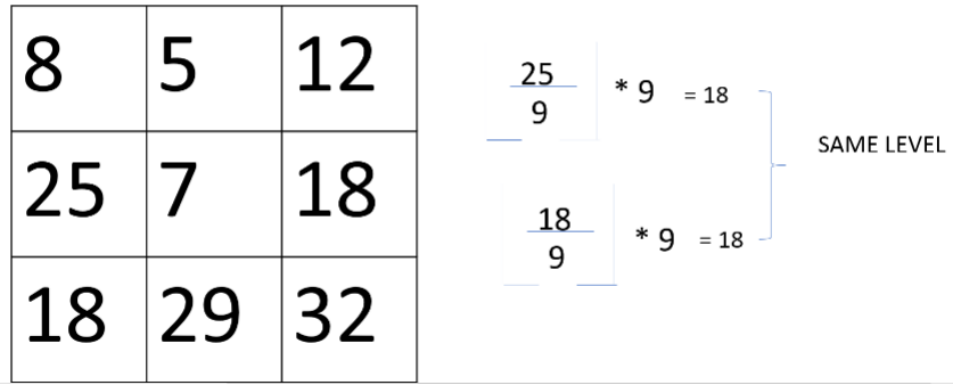

Рисунок 1.6 – Приклад квантування

Таким чином, у наведеному вище прикладі обидва значення інтенсивності округлюються до 18, зменшується кількість окремих рівнів (залучених символів) у специфікації зображення.

1.1.8 Кодування символів

Етап символу передбачає кодування окремих символів зображення таким чином, що ні кількість бітів, необхідних для представлення символу, є оптимальною залежно від частоти появи символу. Простіше кажучи, на цьому етапі кодові слова генеруються для різних присутніх символів. Таким чином прагнеться зменшити число бітів, необхідних для представлення рівнів інтенсивності, і представлення їх в оптимальній кількості бітів.

Існує багато алгоритмів кодування. Деякі з популярних:

– Кодування Хаффмана зі змінною довжиною;
– Кодування довжини серії.

У схемі кодування Хаффмана намагаємося знайти коди таким чином, щоб жоден із кодів не був префіксом іншого. І на основі ймовірності появи символу визначається довжина коду. Щоб отримати оптимальне рішення, найвірогідніший символ має код найменшої довжини (рис. 1.7).

SYMBOLS (with instensity value gray scale)	Probability (arranged in decreasing order)	Binary Code	Huffman code	Length of Huffman code
a1 - 18	0.6	00010010	0	1
a2 - 25	0.3	00011001	10	2
a3 - 255	0.06	11111111	110	3
A4 - 128	0.02	10000000	1110	4
a5 - 200	0.01	11001000	11110	5
a6 - 140	0.01	10001100	11111	5

Рисунок 1.7 – Приклад кодування

Бачимо фактичне 8-бітне представлення, а також нові коди меншої довжини. Механізм генерації кодів зображено на рис. 1.8.

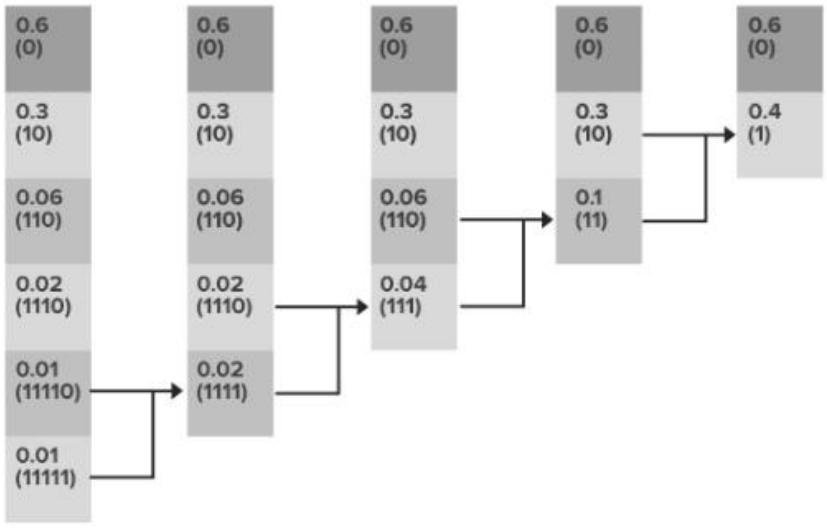

Рисунок 1.8 – Механізм генерації кодів

Таким чином, механізм квантування сприяє стисненню. Після стиснення зображень їх легко зберігати на пристрої або передавати. Виходячи з типу використовуваних перетворень, типу квантування та схеми кодування, декодери розроблені на основі зворотної логіки стиснення, щоб

вихідне зображення можна було перебудувати на основі даних, отриманих зі стиснених зображень.

РОЗДІЛ 2 МАТЕМАТИЧНІ МЕТОДИ СИСТЕМИ

2.1 Методологія

Методологія, яка використовується для оцінки CR кожного формату, спрямована на встановлення точки відліку, спільної для всіх кодеків, незалежно від початкового кодування вибраного набору зображень. Таким чином, усі зображення були закодовані у необроблений формат RGB, який згодом використовувався як точка відліку для всіх вимірювань стиснення. Оскільки JPEG-LS і JPEG2000 не підтримують альфа-канал, під час процесу кодування RGB було застосовано фільтр, який відображає альфа-канал на білий фон.

Основний набір даних, який використовувався для вимірювань, був зібраний і наданий через тест QOI. Набір даних був обраний завдяки наявності широкого спектру зображень, що охоплюють різні випадки використання, від природних зображень і фотографій до штучно створених зображень, скріншотів, значків, текстур і різних об’єктів. Кілька зображень із цього набору даних було видалено через їхні малі розміри, що спричиняло проблеми з налаштуванням кодека JPEG2000. Крім того, одне зображення, яке після видалення альфа-версії було закодовано лише на білому полотні, також було вилучено з тестів. Крім того, результати також повідомляються для зображень, класифікованих у такі ширші групи:

– Природні зображення включають фотографії світу природи;

– Штучні зображення включають комп'ютерну графіку, будь то зображення або різні скріншоти різного ступеня складності;

– Піктограми та об’єкти містять різноманітні піктограми інтерфейсу користувача та зображення виділених об’єктів. Самі об’єкти можуть бути штучними або природними, відображеними на білому фоні;

– Текстури включають різні зображення текстур, які використовуються в обробці тривимірної графіки. Текстури включають як

фотографії поверхонь, рослини, так і синтетичні зображення, тоді як деякі розділені на різні компоненти, такі як дифузія або карти висоти.

Стиснення — це процес кодування інформації з використанням меншої кількості бітів, ніж у вихідному представленні. Величина, яка використовується для вираження ефективності стиснення, називається ступенем стиснення , і це просто частка між нестисненою інформацією та стисненою. Існує два види стиснення:

– Стиснення з втратами: ці процеси кодують інформацію таким чином, що видаляють її частину. Під час декодування інформації «A» після застосування стиснення з втратами ви отримаєте інший результат «B». Загалом інформація, яка втрачається, є менш важливою. JPEG, популярний формат зображення, має втрати, але він видаляє лише інформацію, до якої наші очі менш чутливі, він досягає рівня стиснення порядку 10. JPEG використовує особливості сприйняття зображень, щоб видалити інформацію, яку ми не в змозі обробити, тому інколи буває важко відрізнити оригінальне зображення від втраченого.

– Стиснення без втрат — це тип стиснення, який не видаляє жодної інформації, процес кодування та подальшого декодування будь-якої інформації повертає вихідний результат. Прикладом стиснення зображень без втрат є PNG. Це справді складний формат, у якому досягається ступінь стиснення 5–10 без втрати інформації, і його буде використано як альтернативу порівняння з форматом QOI.

Щоб зрозуміти будь-яку техніку стиснення зображення, потрібно зрозуміти, як комп’ютер представляє зображення. Зображення - це лише матриця пікселів, кожен піксель зберігає інформацію про свій колір. Інформація про колір може бути представлена 3 або 4 числами (так звані канали)(рис. 2.1).

Рисунок 2.1 – Приклад декомпозиції зображення по трьох його каналах

У випадку 3 каналів вони представляють відповідно інтенсивність кольорів червоного, зеленого та синього (RGB); якщо є четвертий канал (альфа), він буде використовуватися для представлення прозорості кожного пікселя.

2.2 Технічні деталі

Змішуючи червоний, зелений і синій, можна отримати будь-який інший колір, тому будь-який піксель зображення можна розкласти на три значення інтенсивності. Кожен канал зазвичай використовує 1 байт інформації, тому для зображення 4K RGB (3840 x 2160 пікселів) знадобиться загалом 24 883 200 байт, що дорівнює 24 Мб. Таким чином, маючи лише 40 хороших зображень, знадобиться 1 Гб пам’яті.

QOI — це формат стиснення без втрат, він має напрочуд невелику складність: O(n) і досягає рівня стиснення, подібного до PNG [8]. Кодер і декодер працюють, лише один раз перебираючи всі пікселі зображення, рядок за рядком. Формат має 14-байтовий заголовок, за яким слідує будь-яка кількість «фрагментів» даних і 8-байтовий кінцевий маркер. Кожен фрагмент може займати 1, 2 або 4 байти. Існує шість типів послідовностей, кожна з яких позначена заголовком [7]:

– QOI_OP_RGB (рис. 2.2): ця частина є просто значенням RGB пікселя, вона не забезпечує стиснення.

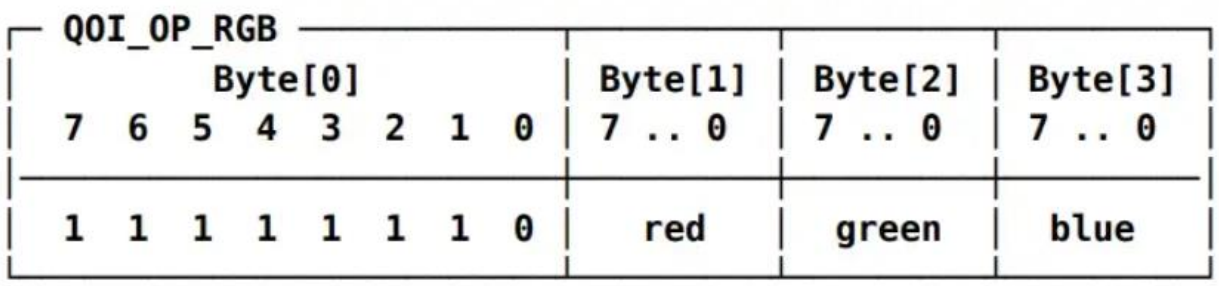

Рисунок 2.2 – Приклад QOI_OP_RGB

– QOI_OP_RGBA (рис. 2.3): це значення RGBA (червоний, зелений, синій і альфа-канал) пікселів: якщо ми починаємо із зображення з інформацією про прозорість, ми використовуватимемо цей тип блоку замість QUI_OP_RGB. Зауважимо, що також у цьому типі фрагментів стиснення не досягається.

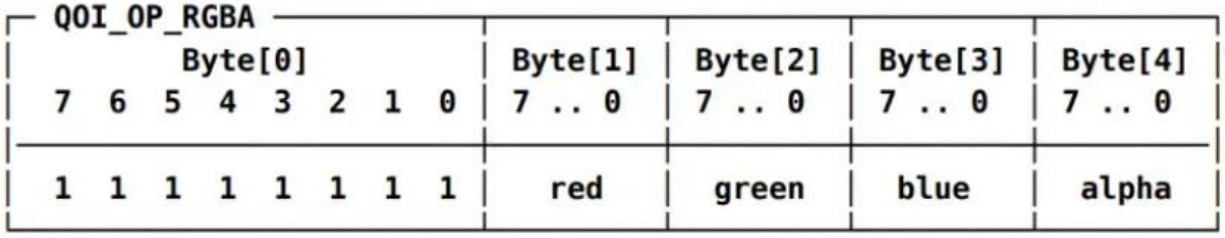

Рисунок 2.3 – Приклад QOI_OP_RGBA

– QOI_OP_INDEX (рис. 2.4): ця частина представляє індекс у масиві раніше побачених значень, це посилання на раніше побачене значення пікселя. Кодер і декодер використовують поточний масив із 64 пікселів, які раніше бачили. Кожен піксель, який зустрічають алгоритми, копіюється в масиві в індексі, обчисленому за допомогою хеш-функції (на основі чотирьох каналів r, g, b, a) $index\_position = (r * 3 + g * 5 + b * 7 + a * 11) \% 64$ Зауважимо, що використання хеш-функції створює зіставлення кольору пікселя з певним індексом у масиві, це дозволяє миттєво отримати попередній піксель, якщо він присутній, і дозволяє уникнути лінійного пошуку в масиві. Це економить багато обчислень для кодера. Підсумовуючи, цей фрагмент є посиланням на попереднє значення, і йому потрібен лише 1 байт пам’яті, що забезпечує деяке стиснення.

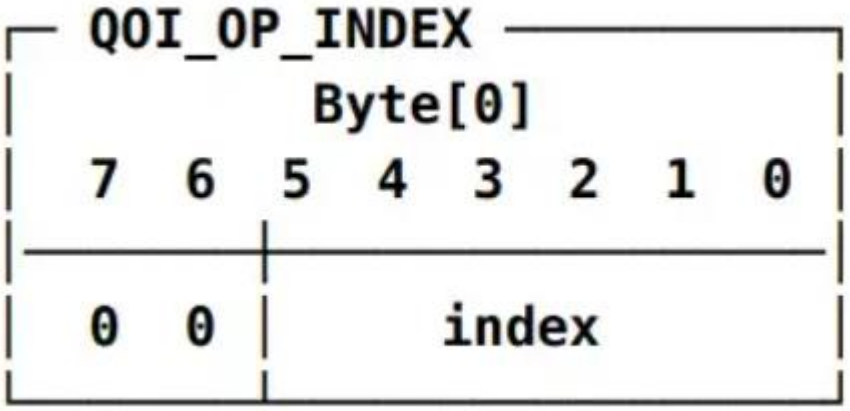

Рисунок 2.4 – Приклад QOI_OP_INDEX

– QOI_OP_DIFF (рис. 2.5): цей фрагмент міститиме модуль різниці 255 (максимальне значення, яке можна представити в 1-байтовому каналі) між поточним пікселем і попереднім. Це корисно, оскільки природні зображення часто складаються із зон подібних кольорів, і часто різниця між двома пікселями є постійною, як у градієнтних зображеннях. Зберігання цієї різниці вимагає лише однієї третини місця, необхідного оригінальному пікселю.

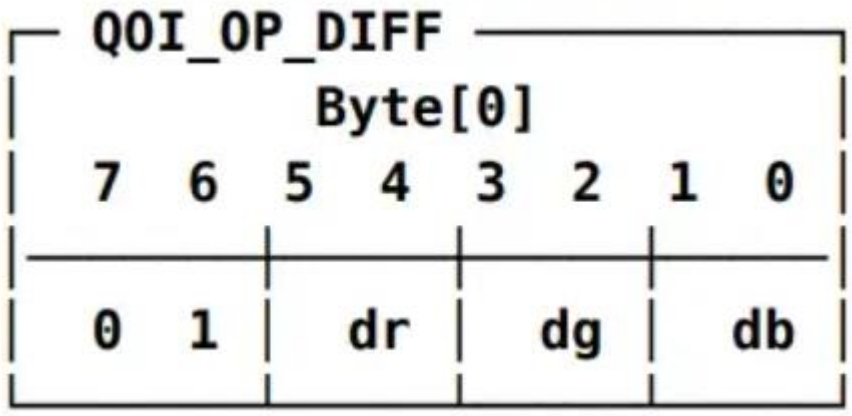

Рисунок 2.5 – Приклад QOI_OP_DIFF

– QOI_OP_LUMA (рис. 2.6): це ще один фрагмент, який представляє різницю між поточним пікселем і попереднім, цього разу зелена різниця використовується як базова лінія для різниці в інших кольорах. Цей метод, як було доведено, збільшує ступінь стиснення в тестах порівняння.

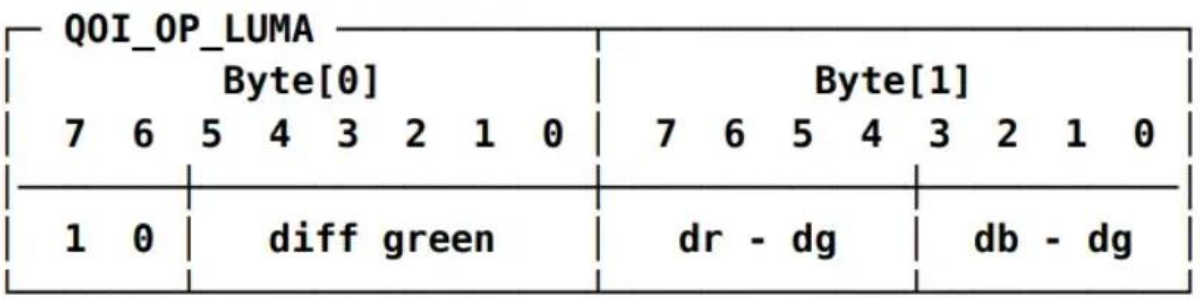

Рисунок 2.6 – Приклад QOI_OP_LUMA

– QOI_OP_RUN (рис. 2.7): цей фрагмент представляє кількість послідовних пікселів, які повторюють те саме значення, що й останній, тут зберігається багато місця, представляючи більше пікселів лише в одному фрагменті. Враховуючи його розмір лише 6 біт, він може представляти лише числа до 62 (63 і 64 зарезервовані для інших заголовків), тому для представлення будь-якої більшої послідовності знадобиться ще один фрагмент.

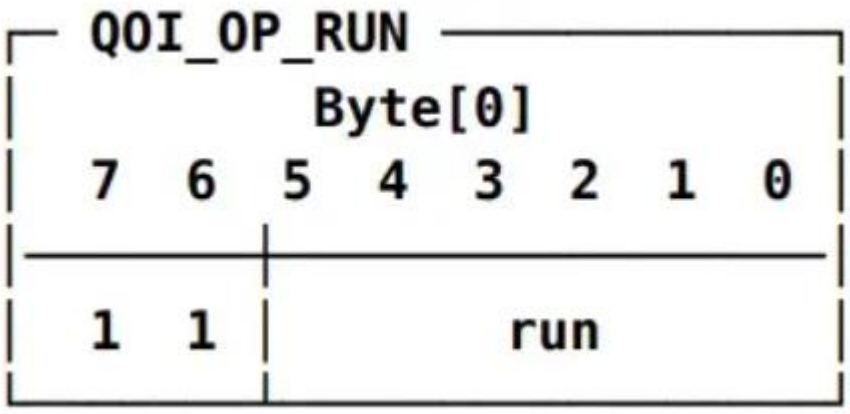

Рисунок 2.7 – Приклад QOI_OP_RUN

Особливість Quite OK Image в тому, що кодування та декодування відбувається за один прохід за зображенням – алгоритм стосується кожного пікселя лише один раз і при цьому кожен піксель кодується одним із чотирьох різних способів.

Результуючі значення записуються в чанки, починаючи з 2..4 tag-біта, який вказує на спосіб кодування, а потім слідує кількість даних у бітах. Усі чанки побайтово вирівняні.

Як було зазначено вище, алгоритм включає у собі 4 різних типи кодування пікселів:

– прогін попереднього пікселя: якщо поточний піксель точно збігається з попереднім, довжина серії збільшується на одиницю. У разі виявлення пікселя, що відрізняється від попереднього, довжина серії зберігається в закодованих даних і поточний піксель упаковується одним із трьох інших методів. Фрагменти довжини прогону бувають двох видів і залежить від кількості необхідних бітів.

```
QOI_RUN_8 {
    u8 tag  :  3;   // b010
    u8 run  :  5;   // 5-бітовий інтервал повторення попереднього пікселя: 1..32
}

QOI_RUN_16 {
    u8 tag  :  3;   // b011
    u16 run : 13;   // 13-бітовий інтервал повторення попереднього пікселя: 33..8224
}
```

– індексування раніше переглянутого пікселя: кодувальник зберігає масив з 64 раніше переглянутих пікселів і якщо виявляє піксель, що досі зберігається в масиві, зберігає його індекс в потік. Для збереження складності O(n) у масиві використовується лише один вид пошуку по хеш значення rgba (r^g^b^a). Лінійний пошук або інші алгоритми ускладнили б кодувальник і сповільнили його.

```
QOI_INDEX {
    u8 tag  :  2;   // b00
    u8 idx  :  6;   // 6-бітний індекс в масиві кольорів: 0..63
}
```

– на відміну від попереднього пікселя: якщо колір поточного пікселя не критично відрізняється від попереднього, різниця між цими пікселями записується в потік. Є три види записів, які вибираються в залежності від

того, наскільки велика різниця. Цікаво, що обробка альфа-каналу займає більше часу, ніж RGB.

```
QOI_DIFF_8 {
    u8 tag  :  2;   // b10
    u8 dr   :  2;   // 2-бітна різниця червоного каналу: -1..2
    u8 dg   :  2;   // 2-бітна різниця зеленого каналу: -1..2
    u8 db   :  2;   // 2-бітна різниця синього каналу: -1..2
}

QOI_DIFF_16 {
    u8 tag  :  3;   // b110
    u8 dr   :  5;   // 5-бітна різниця червоного каналу: -15..16
    u8 dg   :  4;   // 4-бітна різниця зеленого каналу:  -7.. 8
    u8 db   :  4;   // 4-бітна різниця синього каналу:  -7.. 8
}

QOI_DIFF_24 {
    u8 tag  :  4;   // b1110
    u8 dr   :  5;   // 5-бітна різниця червоного каналу: -15..16
    u8 dg   :  5;   // 5-бітна різниця зеленого каналу: -15..16
    u8 db   :  5;   // 5-бітна різниця синього каналу: -15..16
    u8 da   :  5;   // 5-бітна різниця альфа-каналу: -15..16
}
```

– цілі значення RGBA: якщо три попередні способи не вдається застосувати, то в потік зберігаються цілі значення пікселя.

```
QOI_COLOR {
    u8 tag  :  4;   // b1111
    u8 has_r:  1;   // червоний байт
    u8 has_g:  1;   // зелений байт
    u8 has_b:  1;   // синій байт
    u8 has_a:  1;   // альфа-байт
    u8 r;           // значення червоного байту, якщо has_r == 1: 0..255
    u8 g;           // значення зеленого байту, якщо has_g == 1: 0..255
    u8 b;           // значення синього байту, якщо has_b == 1: 0..255
    u8 a;           // значення альфа-байту, якщо has_a == 1: 0..255
}
```

Є багато параметрів, за якими QOI можна покращити, але більшість випадків передбачає компроміси, наприклад, більша складність задля кращого стиснення. Сфери, де QOI міг би бути суворо кращим, розміри, за якими він не знаходиться на межі Парето, є більш значущою критикою — втрачені можливості. З основних недоліків можна перерахувати:

– Поля big-endian байтів є дивним вибором для формату файлу QOI. Little-endian домінує в галузі, і це призвело б до дещо меншого сліду декодера на типових машинах сьогодні, якби QOI використовував little-endian;

– Заголовок має два прапорці і витрачає цілий байт на кожен. Натомість він мав мати байт прапора з двома бітами, призначеними цим прапорам. Один прапорець вказує, чи альфа-канал є важливим, а інший вибирає між двома просторами кольорів (sRGB, лінійний);

– Формат 4-канального кодування пікселів — це ABGR (або RGBA), де альфа-канал розміщується поруч із синім каналом. Це дещо нетрадиційно. Декодер, швидше за все, використовуватиме одне завантаження в 32-бітове ціле число, і в ідеалі воно вже має потрібний формат або близький до нього;

– Хеш-функція QOI працює на окремих каналах з індивідуальним переповненням, що робить її повільнішою та більшою, ніж необхідно.

Зміна параметру довжини послідовності, яку можна максимально зберігати, може впливати на якість стиснення та швидкість обробки даних.

Збільшення довжини послідовності може призвести до більш ефективного стиснення даних, оскільки довші послідовності даних можуть бути замінені одним токеном, що зменшує кількість даних, що потрібно зберігати. Однак це може збільшити час обробки даних, оскільки більші послідовності даних потребують більшої обчислювальної потужності та пам'яті для їх обробки.

Зменшення довжини послідовності може зменшити час обробки даних, оскільки менші послідовності даних потребують меншої обчислювальної

потужності та пам'яті для їх обробки. Однак це може погіршити якість стиснення даних, оскільки менші послідовності даних не можуть бути замінені на один токен, що зменшує ефективність стиснення.

Стиснення зображень для вбудованих систем з малою кількістю оперативної пам'яті є важливою задачею, оскільки обмежені ресурси систем можуть ускладнити зберігання та обробку зображень.

2.3 QOI кодер / декодер

QOIE Core — це кодер, який реалізує високоефективний, малопотужний механізм стиснення зображень без втрат (рис. 2.8), сумісний із специфікацією формату зображення Quite OK (QOI) версії 1.0. Алгоритм QOI стискає зображення RGB або RGBA з 8 бітами на колір без будь-яких втрат. Він має ефективність стиснення, близьку до стиснення PNG, але з часткою обчислювальної складності.

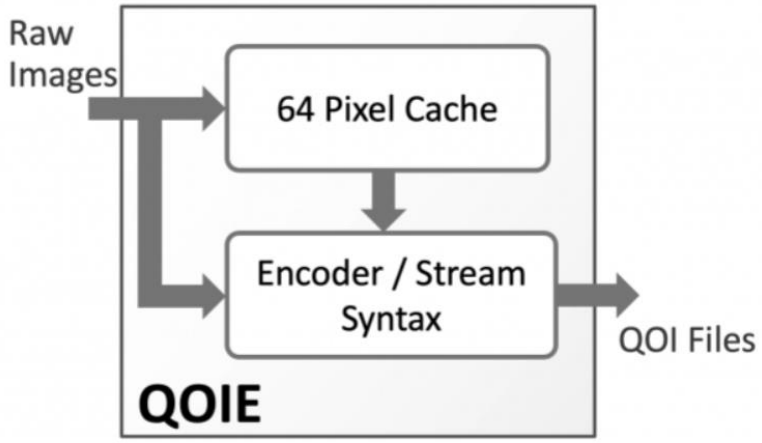

Рисунок 2.8 – Приклад QOI кодер

Використовуючи простоту алгоритму QOI, ядро кодера QOIE може стискати зображення з дуже високою швидкістю та з мінімальними ресурсами обчислювального ядра. Ядро займає приблизно 15 000 еквівалентних вентилів NAND2 і може обробляти один піксель за такт. Одноядерний екземпляр може стискати зображення зі швидкістю, достатньою для відео UHD 4k30 навіть у низькокласних FPGA, 4k60 у FPGA середнього класу та 8k30 або 60 у сучасних технологіях ASIC.

Кодер QOI базується на таких характеристиках:

– Один 24-бітний RGB-піксель, закодований, гарантований кожен такт;

– Виведення стиснених даних через 32-розрядний інтерфейс;

– Повністю синхронна конструкція на основі висхідного фронту годинника.

Оскільки кожен піксель може бути закодований до 4 байтів, 32-розрядний вихідний інтерфейс необхідний для забезпечення кодування пікселя кожного такту в гіршому випадку. Підтримка пікселів RGBA (альфа) може потребувати до 5 байт для кодування пікселя. Для цього знадобився б значно більший вихідний інтерфейс (64-бітний), що, на мою думку, було невиправданим на цьому етапі. Крім того, QOI не намагається стиснути піксель, коли змінюється альфа: він просто виводить 5 байтів (піксель RGBA + код).

Кодер QOI дуже простий в експлуатації. Після активації, після затримки в 62 цикли, повний 24-бітний піксель RGB приймається кожного такту. Стиснення можна призупинити в будь-який час шляхом зниження сигналу синхронного дозволу.

Ця затримка потрібна для очищення кешу пікселів, як того вимагає алгоритм QOI. У FPGA піксельний кеш реалізований як розподілена оперативна пам’ять, і останню неможливо очистити за один такт. У ASIC, якщо піксельний кеш реалізований як тригери, це, безумовно, можливість. Однак, якщо говорити реально, потрібно враховувати той факт, що 62 цикли на початку є незначною затримкою порівняно з мільйонами пікселів у зображенні. У будь-якому випадку, це також можна вирішити в FPGA.

Конструкція декодера QOI доповнює кодер (рис. 2.9). Коли декодер активований, стислі дані запитуються з 32-бітного інтерфейсу. Після кількох тактових циклів затримки 24-бітові пікселі декодуються та виводяться, по одному на кожен такт.

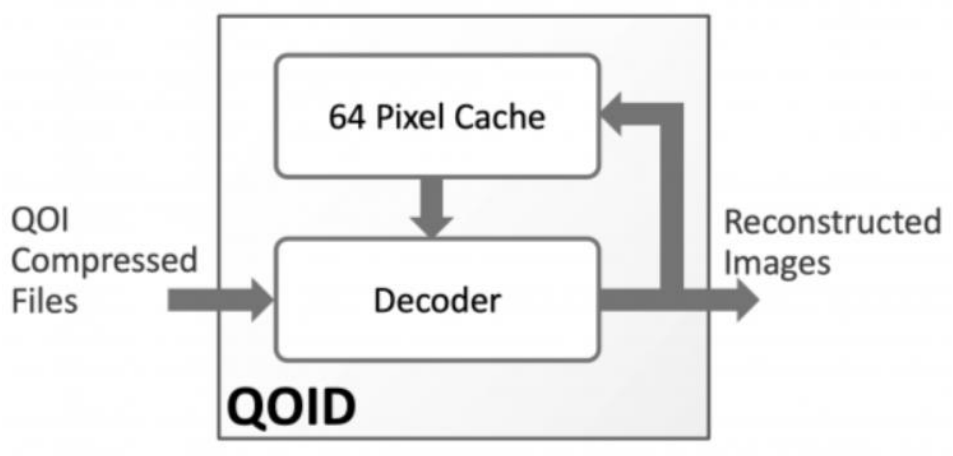

Рисунок 2.9 – Приклад QOI декодер

Декодування можна призупинити в будь-який час шляхом зниження сигналу синхронного дозволу. Як і кодер, декодер не обробляє заголовок QOI, хоча його можна легко додати.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ

3.1 Visual Studio Code

Visual Studio Code (VS Code) — потужний і універсальний редактор коду, розроблений Microsoft. Це безкоштовне середовище розробки з відкритим вихідним кодом, яке підходить для широкого діапазону мов програмування та платформ, що робить його популярним вибором для багатьох розробників. Це чудовий інструмент для будь-якого програміста, який хоче створити високопродуктивну, багатофункціональну програму. Інтерфейс застосунку наведено на рис. 3.1.

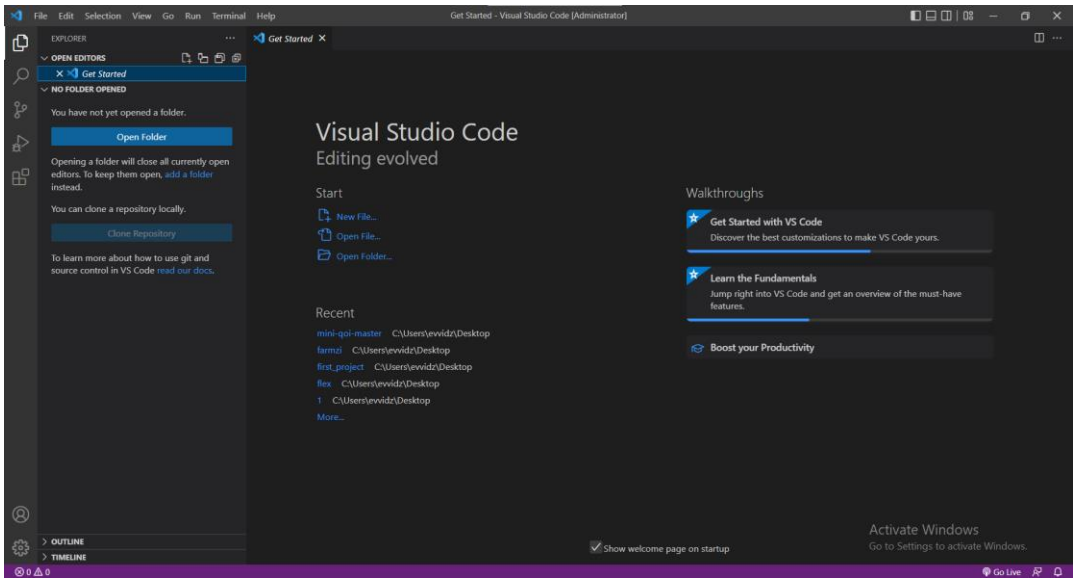

Рисунок 3.1 – Інтерфейс VS Code

Microsoft Visual Studio — чудовий інструмент для розробки комп’ютерних програм, веб-сайтів, мобільних додатків і веб-сервісів. VS Code — це абсолютно інший продукт порівняно з Visual Studio. Це кросплатформна програма – працює на Windows, Linux і Mac, тоді як Visual Studio працює лише на операційній системі Windows і Mac. VS Code можна використовувати для написання, редагування та налагодження коду одночасно. Як результат, він поставляється з вбудованими розширеннями для таких мов, як Python, C#, PHP, Java та багатьох інших. Для розробників із 90% або більше досвідом VS Code є кращим вибором, ніж Visual Studio.

Багато його функцій не мають великої цінності для деяких розробників, але вони віддають перевагу Visual Studio над іншими альтернативами. Завдяки своїй легкості, преміум-класу, простоті, найкращий редактор коду на ринку.

VS Code — чудовий інструмент для розробників усіх рівнів. Його можуть використовувати початківці, які тільки починають вивчати програмування, оскільки він надає простий у використанні текстовий редактор та інструменти для налагодження. Досвідчені розробники також можуть використовувати VS Code для своїх проектів, оскільки він пропонує розширені функції, такі як підсвічування синтаксису, IntelliSense та можливості налагодження. Завдяки широкому спектру функцій і гнучкості VS Code є чудовим вибором для будь-якого типу проекту розробки.

VS Code має кілька дуже унікальних функцій:

– Підтримка кількох мов програмування: підтримує кілька мов програмування. Тому раніше програмістам потрібен був Web-Support: інший редактор для різних мов, але він має вбудовану багатомовну підтримку. Це також означає, що він легко виявляє будь-яку помилку чи посилання на іншу мову;

– Intelli-Sense: він може виявити, якщо якийсь фрагмент коду залишився незавершеним. Крім того, загальний синтаксис змінних і оголошення змінних створюються автоматично. Приклад: якщо в програмі використовується певна змінна, яку користувач забув оголосити, intelli-sense оголосить її для користувача (рис.3.2). VS Code IntelliSense доступний для JavaScript, TypeScript, JSON, HTML, CSS, Less і Sass. Також можна додати розширення IntelliSense для інших мов, які не підтримуються за замовчуванням;

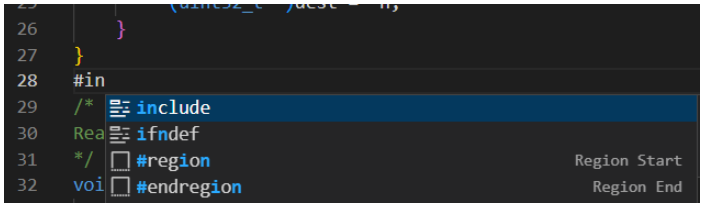

Рисунок 3.2 – Використання Intelli-Sense

– Підтримка між платформами: традиційно редактори підтримували системи Windows, Linux або Mac . Але VS Code є кросплатформенним. Тому він може працювати на всіх трьох платформах. Крім того, код працює на всіх трьох платформах; інакше код програмного забезпечення з відкритим кодом і програмного забезпечення відрізнялися;

– Розширення та підтримка: зазвичай підтримує всі мови програмування, але якщо користувач/програміст хоче використовувати мову програмування, яка не підтримується, він може завантажити розширення та використовувати його (рис. 3.3). З точки зору продуктивності розширення не сповільнює роботу редактора, оскільки воно виглядає як інший процес;

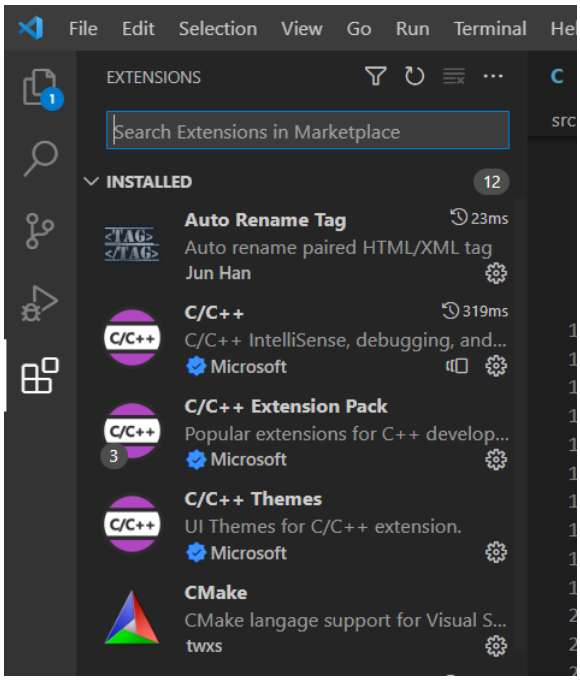

Рисунок 3.3 – Панель розширень

– Репозиторій: оскільки попит на код постійно зростає, безпечне та своєчасне зберігання є однаково важливим. Він підключений до Git або може бути підключений до будь-якого іншого сховища для отримання або збереження екземплярів. Ліворуч на бічній панелі можна знайти піктограму Git (рис. 3.4), за допомогою якої можна ініціалізувати Git, а також виконати кілька команд Git, таких як commit, pull, push, rebase, public та переглядати зміни у файлі. VS Code працює з будь-яким локальним або віддаленим

репозиторієм Git і пропонує візуальний символ для вирішення конфліктів перед фіксацією коду;

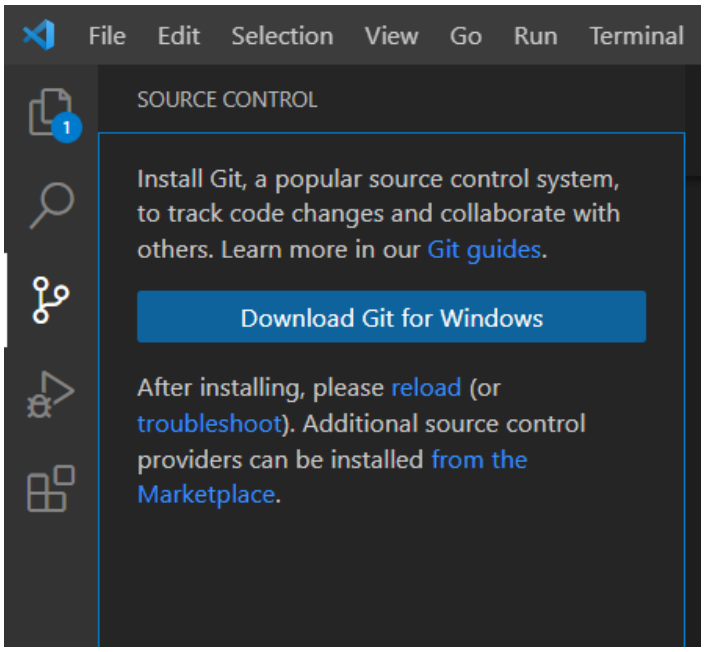

Рисунок 3.4 – Панель Git

– Веб-підтримка: поставляється з вбудованою підтримкою веб-додатків. Таким чином, веб-програми можна створювати та підтримувати у VSC;

– Ієрархічна структура: файли коду розташовані у файлах і папках. Необхідні файли коду також містять деякі файли, які можуть знадобитися для інших складних проектів;

– Удосконалення коду: деякі фрагменти коду можна декларувати дещо інакше, що може допомогти користувачеві в коді. Ця функція пропонує користувачеві, якщо необхідно, змінити його на запропонований варіант;

– Підтримка терміналу: у багатьох випадках користувачеві потрібно починати з кореня каталогу, щоб розпочати певну дію, вбудований термінал або консоль надає підтримку користувачам, щоб не перемикатися між двома екранами для того самого (рис. 3.5);

Рисунок 3.5 – Термінал VS Code

– Кілька проектів: кілька проектів, що містять кілька файлів/папок, можна відкривати одночасно (рис. 3.6). Ці проекти/папки можуть бути або не пов’язані один з одним;

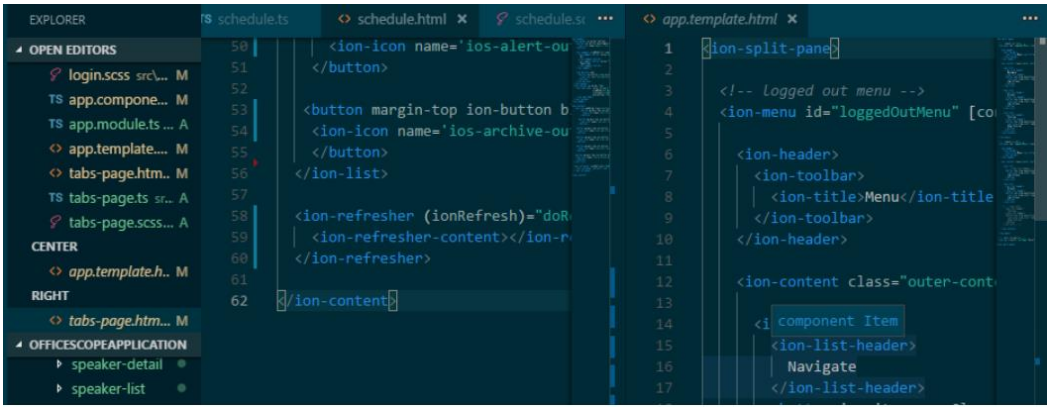

Рисунок 3.6 – Відкриті файли

– Підтримка Git: ресурси можна отримати з Git Hub Repo онлайн і навпаки; також можна заощадити. Вилучення ресурсів також означає клонування коду, доступного в Інтернеті. Пізніше цей код можна змінити та зберегти;

– Коментування: загальна функція, але деякі мови її не підтримують. Коментування коду допомагає користувачеві згадати або відстежити відповідно до потрібної йому послідовності.

Завдяки прогресу технологій день у день VS Code відіграватиме ключову роль у розробці програмного забезпечення. Завдяки функціям, що постійно розвиваються, і новим налаштуванням, які незабаром будуть додані, і які дозволять користувачам працювати з ним звідки завгодно.

3.2 Програмна частина

Mini QOI — це потоковий декодер QOI , розроблений для вбудованої системи з дуже малим обсягом оперативної пам'яті, який має ряд переваг:

– Він обходить обмеження в 400 МП еталонного декодера;

– Не вимагає динамічного розподілу пам'яті;

– Для декодування зображення потрібно лише близько 300 байт оперативної пам'яті;

– Він підтримує декодування вхідного потоку QOI байт за байтом.

Бібліотека Mini QOI для Arduino є інструментом, що дозволяє використовувати формат стисненого зображення QOI на пристроях з обмеженою кількістю оперативної пам'яті, таких як Arduino.

Основна функціональність бібліотеки включає:

– Завантаження та розпакування стисненого зображення QOI: бібліотека містить методи для завантаження стисненого зображення з пам'яті мікроконтролера та його розпакування в оригінальний формат зображення;

– Збереження зображення в форматі QOI: бібліотека також містить методи для стиснення зображення в формат QOI та збереження його в пам'яті мікроконтролера;

– Розмір стисненого зображення: бібліотека надає інформацію про розмір стисненого зображення, що дозволяє оцінити ефективність стиснення;

– Режим економії пам'яті: бібліотека містить параметри, що дозволяють вибирати рівень економії пам'яті, зберігаючи менше даних під час стиснення зображення, але збільшуючи час розпакування;

– Різноманітні опції: бібліотека містить різноманітні опції, такі як зміна розміру зображення або зміна якості стиснення, що дозволяють оптимізувати процес стиснення під конкретні потреби.

Бібліотека Mini QOI є легкою у використанні та має низькі вимоги до пам'яті, що робить її ідеальним вибором для проектів, що вимагають ефективного стиснення зображень на пристроях з обмеженими ресурсами.

На вході функції encode приймається масив data - це вхідні дані, які потрібно стиснути. Функція виконує кілька дій, включаючи обчислення середнього значення пікселів і їх розкиду, після чого виконується кодування даних з використанням алгоритму QOI. Стиснені дані повертаються з функції.

Функція decode здійснює розпакування стиснених даних, переданих як вхідний параметр data, і повертає масив розпакованих даних.

Програма включає в себе допоміжні функції, які використовуються для реалізації основних функцій.

Одна з ключових особливостей цієї програми - це використання статичних таблиць для зберігання даних, які використовуються для кодування і декодування даних. Завдяки цьому програма займає менше пам'яті та працює швидше.

Функція qoi_compress() приймає на вхід масив даних (у вигляді масиву байтів), розмір масиву, вказівник на буфер для зберігання стисненого вихідного потоку та його розмір. Функція обробляє вхідні дані та записує стиснений вихід до буферу.

Функція qoi_decompress() виконує розпакування стисненого потоку та повертає масив байтів розміром, що дорівнює розміру розпакованого потоку.

Алгоритм стиснення QOI заснований на стисненні повторюваних підрядків байтів. Основним інструментом для цього є хеш-таблиці, які зберігають попередні повторювані підрядки та їх індекси у вихідному потоці.

Для забезпечення ефективного використання пам'яті, таблиця хешів обмежена максимальним розміром та містить лише останні 2048 знайдених повторень, алгоритм використовує вказівники на попередні знайдені повтори замість зберігання всього повторюваного підрядку. Крім того, алгоритм використовує просту евристику, щоб визначити, коли використовувати повторення, а коли просто додати новий байт до вихідного потоку.

У цілому, код реалізує ефективний і оптимізований алгоритм стиснення зображень для мікроконтролерів з обмеженою кількістю оперативної пам'яті.

Наступна частина коду читає байт з об’єкта дескриптора зображення mQOI. Повертає NULL, якщо нічого не залишилося для читання.

```
uint8_t * mqoi_desc_pop(mqoi_desc_t * desc) {
    if (desc->head >= sizeof(mqoi_desc_t) - 1) return NULL;
    return (uint8_t *)(desc + (++desc->head));
}
```

Перевіряє, чи дійсний дескриптор зображення mQOI, і зчитує його ширину та висоту в w і h. Якщо він повертає ненульовий код, він недійсний.

```
uint8_t mqoi_desc_verify(mqoi_desc_t * desc, uint32_t * w, uint32_t * h) {
    if (desc->magic[0] != 'q' || desc->magic[1] != 'o'
        || desc->magic[2] != 'i' || desc->magic[3] != 'f') {
        return MQOI_DESC_INVALID_MAGIC;
    }
    mqoi_u32_read(desc->width, w);
    mqoi_u32_read(desc->height, h);

    if (desc->channels != MQOI_CHANNELS_RGB && desc->channels != MQOI_CHANNELS_RGBA)
{
        return MQOI_DESC_INVALID_CHANNELS;
    }
    if (desc->colorspace != MQOI_COLORSPACE_SRGB && desc->colorspace !=
MQOI_COLORSPACE_LINEAR) {
        return MQOI_DESC_INVALID_COLORSPACE;
    }
    return MQOI_DESC_OK;
}
```

Повертає true, коли об’єкт дескриптора зображення mQOI повністю заповнений.

```
MQOI_INLINE bool mqoi_desc_done(const mqoi_desc_t * desc) {
    return desc->head >= sizeof(mqoi_desc_t) - 1;
}
```

Ініціалізує об’єкт декодера mQOI. Якщо задано кількість пікселів у зображенні (через n_pix), функція mqoi_dec_done працюватиме.

```
void mqoi_dec_init(mqoi_dec_t * dec, uint32_t n_pix) {
    memset(dec, 0, sizeof(mqoi_dec_t));
    dec->prev_px.a = 0xff;
    dec->pix_left = n_pix;
}
```

Надсилає байт до декодера mQOI. Не потрібно викликати це більше ніж один раз, якщо всі пікселі не витягнуто за допомогою mqoi_dec_pop. Також, не потрібно змішувати цю функцію з викликами mqoi_dec_take.

```
void mqoi_dec_push(mqoi_dec_t * dec, uint8_t byte) {
    dec->curr_chunk.value[dec->curr_chunk_head++] = byte;

    if (dec->curr_chunk_size == 0 && dec->curr_chunk_head == 1) { // if we have the
head of a new chunk, get its size

        switch (dec->curr_chunk.head) { // test 8-bit tags
            case MQOI_OP8_RUN_RGBA: dec->curr_chunk_size = 5; break;
            case MQOI_OP8_RUN_RGB: dec->curr_chunk_size = 4; break;
            default: { // test 2-bit tags
                // chunk size is 1 unless it's an OP_LUMA chunk
                dec->curr_chunk_size = 1 + ((dec->curr_chunk.head & MQOI_MASK_OP_2B)
== MQOI_OP2_LUMA);
                break;
            }
        }
    }
}
```

Автоматично читає до 5 байтів, щоб наступний виклик mqoi_dec_pop не повертав NULL.

Повертає кількість байтів, які були "взяті" з масиву байтів. Також, не потрібно змішувати цю функцію з викликами mqoi_dec_push.

```
uint8_t mqoi_dec_take(mqoi_dec_t * dec, const uint8_t * bytes) {

    dec->curr_chunk.value[dec->curr_chunk_head++] = *(bytes++);
    switch (dec->curr_chunk.head) { // test 8-bit tags
        case MQOI_OP8_RUN_RGBA: dec->curr_chunk_size = 5; break;
        case MQOI_OP8_RUN_RGB: dec->curr_chunk_size = 4; break;
        default: { // test 2-bit tags
```

```
            // chunk size is 1 unless it's an OP_LUMA chunk
            dec->curr_chunk_size = 1 + ((dec->curr_chunk.head & MQOI_MASK_OP_2B) ==
MQOI_OP2_LUMA);
            break;
        }
    }
    while (dec->curr_chunk_head < dec->curr_chunk_size) {
        dec->curr_chunk.value[dec->curr_chunk_head++] = *(bytes++);
    }
    return dec->curr_chunk_head;
}
```

Витягує піксель із декодера mQOI.

Повертає NULL, якщо потрібно більше даних, інакше повертає адресу наступного пікселя.

```
mqoi_rgba_t * mqoi_dec_pop(mqoi_dec_t * dec)
```

Якщо ми в кінці поточного фрагмента, то виконується наступний фрагмент:

```
if (dec->curr_chunk_size && dec->curr_chunk_head >= dec->curr_chunk_size) {
        mqoi_rgba_t px = { .a = dec->prev_px.a };
        mqoi_rgba_t * px_ptr = NULL;
        bool chunk_done = true;
```

Якщо немає необхідності маскувати біти, оскільки верхні біти дорівнюють нулю, вивести кожен канал і обчислити різницю.

```
switch (dec->curr_chunk.head & MQOI_MASK_OP_2B) {
                    case MQOI_OP2_INDEX:
                        px_ptr = &dec->hashtable[dec->curr_chunk.head]; /
                        break;
                    case MQOI_OP2_DIFF:
                    px.b = dec->prev_px.b - 2 + (dec->curr_chunk.head & 0b11);
                        dec->curr_chunk.head >>= 2;
                        px.g = dec->prev_px.g - 2 + (dec->curr_chunk.head & 0b11);
                        dec->curr_chunk.head >>= 2;
                        px.r = dec->prev_px.r - 2 + (dec->curr_chunk.head & 0b11);
                        break;
                    case MQOI_OP2_LUMA: {
                      int8_t dg = (dec->curr_chunk.head & MQOI_MASK_OP_LUMA_DG) - 32;
```

```
                    px.g = dec->prev_px.g + dg;
                px.b = dec->prev_px.b + (dec->curr_chunk.drdb & 0b1111) - 8 + dg;
                    dec->curr_chunk.drdb >>= 4;
                px.r = dec->prev_px.r + (dec->curr_chunk.drdb & 0b1111) - 8 + dg;
                    break;
                }
```

Повертає true, якщо декодер випустив усі пікселі, необхідні для завершення декодованого зображення.

Потрібно звернути увагу, що ця функція працюватиме, лише якщо n_pix було задано під час ініціалізації декодера.

```
MQOI_INLINE bool mqoi_dec_done(const mqoi_dec_t * dec) {
    return dec->pix_left == 0;
```

Записує беззнакове 32-розрядне ціле число від n до dest.

```
void mqoi_u32_write(const uint32_t * n, uint8_t * dest) {
    if (MQOI_ISHOSTLE) {
        *(uint32_t *)dest = MQOI_BSWAP32(*n); // swap bytes if little endian
    } else {
        *(uint32_t *)dest = *n;
    }
}
```

Зчитує 32-розрядне ціле беззнакове число з порядком старшого порядку з src у n.

```
void mqoi_u32_read(const uint8_t * src, uint32_t * n) {
    if (MQOI_ISHOSTLE) {
        *n = MQOI_BSWAP32(*(uint32_t *)src); // swap bytes if little endian
    } else {
        *n = *(uint32_t *)src;
```

РОЗДІЛ 4 ЕКСПЕРЕМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

4.1 Ефективність стиснення

Основний набір даних, який використовувався для вимірювань, був зібраний і наданий через тест QOI [4]. Набір даних був обраний завдяки наявності широкого спектру зображень, що охоплюють різні випадки використання, від природних зображень і фотографій до штучно створених зображень, скріншотів, значків, текстур і різних об’єктів. Кілька зображень із цього набору даних було видалено через їхні малі розміри, що спричиняло проблеми з налаштуванням кодека JPEG2000. Крім того, одне зображення, яке після видалення альфа-версії було закодовано лише на білому полотні, також було вилучено з тестів. Набір даних було доповнено включенням 8-бітних варіантів RGB високоякісних нових тестових зображень.

Зображення та результати, що повідомляються, організовані таким же чином, як і у звіті про рівень стиснення проекту QOI [4]. Правила іменування груп відповідають структурі каталогів бібліотеки зображень, а саме icon_64, icon_512, photo_kodak, photo_tecnick, photo_wikipedia, pngimg, screenshot_game, screenshot_web, textures_photo, textures_pk, textures_pk01, textures_pk02 і textures_plants. Зображення з набору даних New Test Images [5] називаються rgb8bit. Крім того, результати також повідомляються для зображень, класифікованих у такі ширші групи:

– Природні зображення включають фотографії світу природи;

– Штучні зображення включають комп'ютерну графіку, будь то зображення або різні скріншоти різного ступеня складності;

– Піктограми та об’єкти містять різноманітні піктограми інтерфейсу користувача та зображення виділених об’єктів. Самі об’єкти можуть бути штучними або природними, відображеними на білому фоні;

– Текстури включають різні зображення текстур, які використовуються в обробці тривимірної графіки. Текстури включають як

фотографії поверхонь, рослини, так і синтетичні зображення, тоді як деякі розділені на різні компоненти, такі як дифузія або карти висоти.

Результати тестування ступеня стиснення впорядковано спочатку за категоріями. Для кожної категорії та набору подано короткий опис. Результат найвищого коефіцієнта стиснення виключається з гістограми для зручності читання, оскільки в багатьох випадках він може бути на порядки більшим, ніж середнє та сукупне значення.

Подібно до діаграм розсіювання, деякі екстремальні значення іноді можуть бути опущені на діаграмі для ясності, тоді як друга діаграма, зосереджена на меншій підгрупі зображень, може бути присутньою для подальшого підвищення розбірливості.

Природні зображення. Категорія природних зображень складається з 186 зображень, взятих із наборів зображень photo_kodak, photo_tecnick, photo_wikipedia та rgb8bit. Підсумок цієї категорії наведено в таблиці 4.1, та на рисунку 4.2, на рисунку 4.1 показано коефіцієнт стиснення зображення на формат усієї категорії.

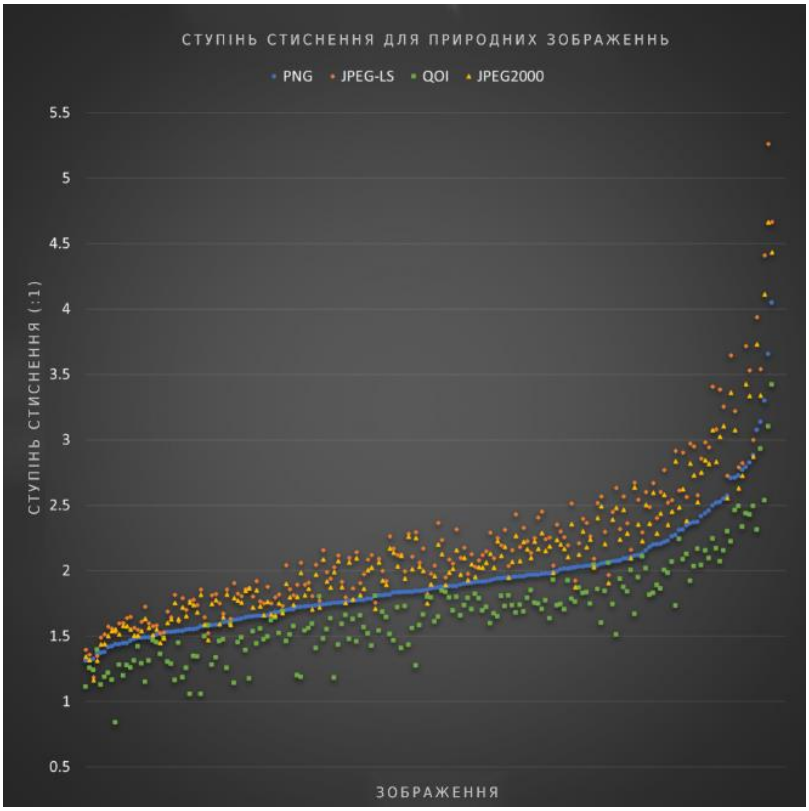

Рисунок 4.1 – Ступінь стиснення для природних зображень

У цій категорії представлені фотографії світу природи. JPEG-LS і JPEG2000 досягають найвищого CR на більшості зразків. PNG не надається переваги в цій конкретній категорії, тоді як QOI трохи відстає, але все ще є життєздатним варіантом для стиснення.

Таблиця 4.1 – Підсумок коефіцієнта стиснення для природних зображень

Формат	**PNG**	**JPEG-LS**	**QOI**	**JPEG2000**
Найкращий (x:1)	4,046	5,262	3,421	4,662
Найгірший (x:1)	1,312	1,184	0,839	1,163
Середнє (x:1)	1,922	2,194	1,686	2,099
Сукупність (x:1)	1,855	2,136	1,502	2,061

Між середнім, сукупним, найкращим і найгіршим випадками для кожного формату спостерігається невелика дисперсія, тоді як існує послідовне впорядкування форматів між середнім і сукупним значеннями.

Рисунок 4.2 – Підсумок коефіцієнта стиснення для природних зображень

Штучні зображення. Категорія штучних зображень складається з 613 зображень, взятих із наборів зображень rgb8bit, screenshot_game та screenshot_web. Підсумок цієї категорії наведено в таблиці 4.2, а також на

рисунку 4.4, на рисунку 4.3 показано коефіцієнт стиснення зображення на формат усієї категорії.

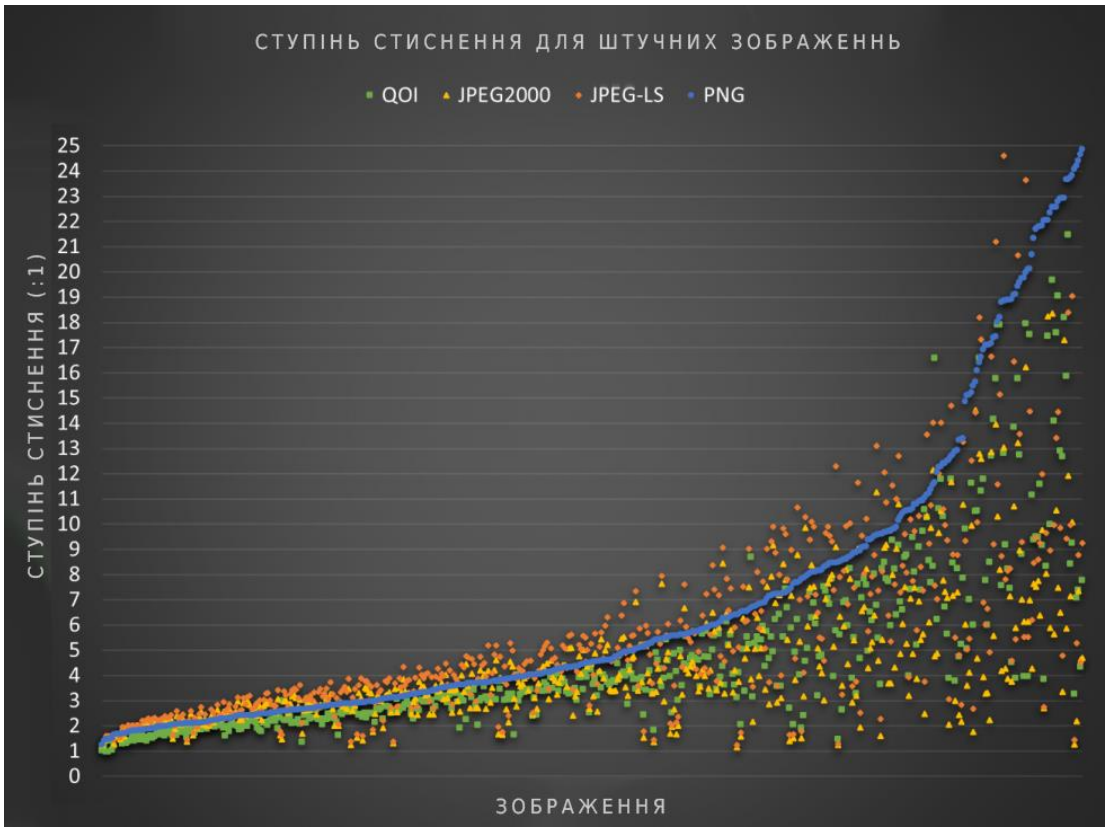

Рисунок 4.3 – Ступінь стиснення для штучних зображень

Ця категорія складається зі створених комп’ютером зображень і скріншотів. У середньому PNG перевершує інші формати, а продуктивність JPEG-LS і QOI є порівнянними. Зображення, які можна ідентифікувати як більш стислі, здається, надають перевагу PNG і QOI, тоді як JPEG-LS і JPEG2000 потенційно можуть бути на одному рівні з двома іншими форматами для зображень цієї категорії, які важче стискати.

Таблиця 4.2 – Підсумок коефіцієнта стиснення для штучних зображень

Формат	**PNG**	**JPEG-LS**	**QOI**	**JPEG2000**
Найкращий (x:1)	493,061	417,311	152,801	187,735
Найгірший (x:1)	1,235	1,237	0,973	1,164
Середнє (x:1)	26,976	12,794	12,686	6,517
Сукупність (x:1)	4,448	4,476	3,542	3,753

Спостерігається більша різниця між мінімальним і максимальним CR, при цьому найкращий результат на два порядки перевищує найгірший для кожного формату. Різниця більш помітна у верхніх 25% результатів CR, при цьому PNG має значно кращі результати для цих конкретних зображень. Це також є причиною того, що середній коефіцієнт стиснення кодування PNG у два або більше разів перевищує середнє значення кожного іншого формату.

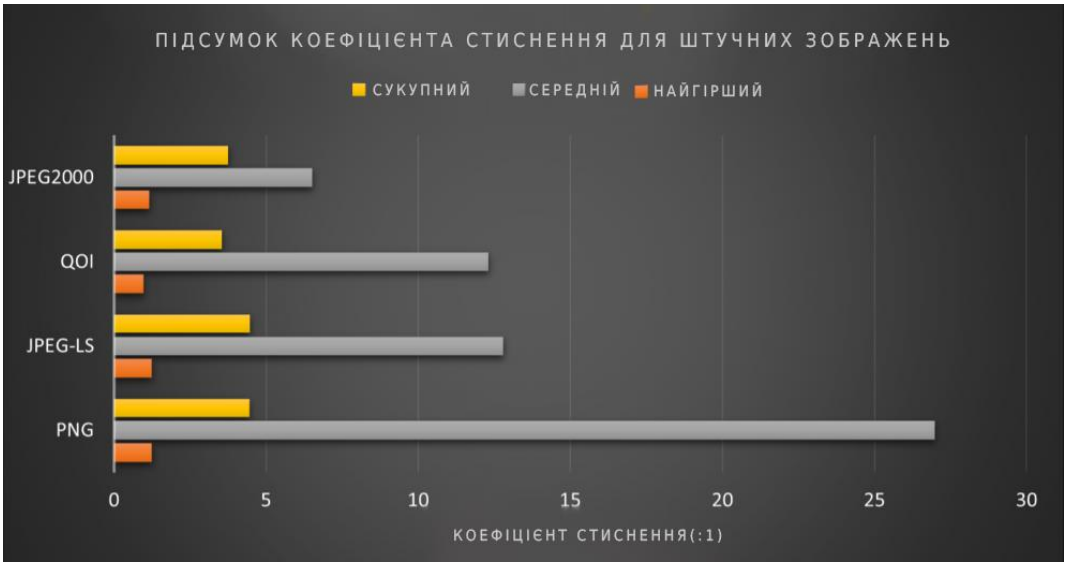

Рисунок 4.4 – Підсумок коефіцієнта стиснення для штучних зображень

Об’єкти та значки. Категорія об’єктів і значків складається з 612 зображень, взятих із наборів зображень icons_64, icons_512 і pngimg. Підсумок цієї категорії представлено в таблиці 4.3, а також на рисунку 4.6, на рисунку 4.5 показано коефіцієнт стиснення зображення на формат усієї категорії.

Ця категорія включає як природні, так і штучні зображення об’єктів, наприклад обрізані елементи різної складності та елементи інтерфейсу користувача.

Результати для цієї категорії можна порівняти з результатами штучних зображень, хоча й з менш вираженими відмінностями між форматами.

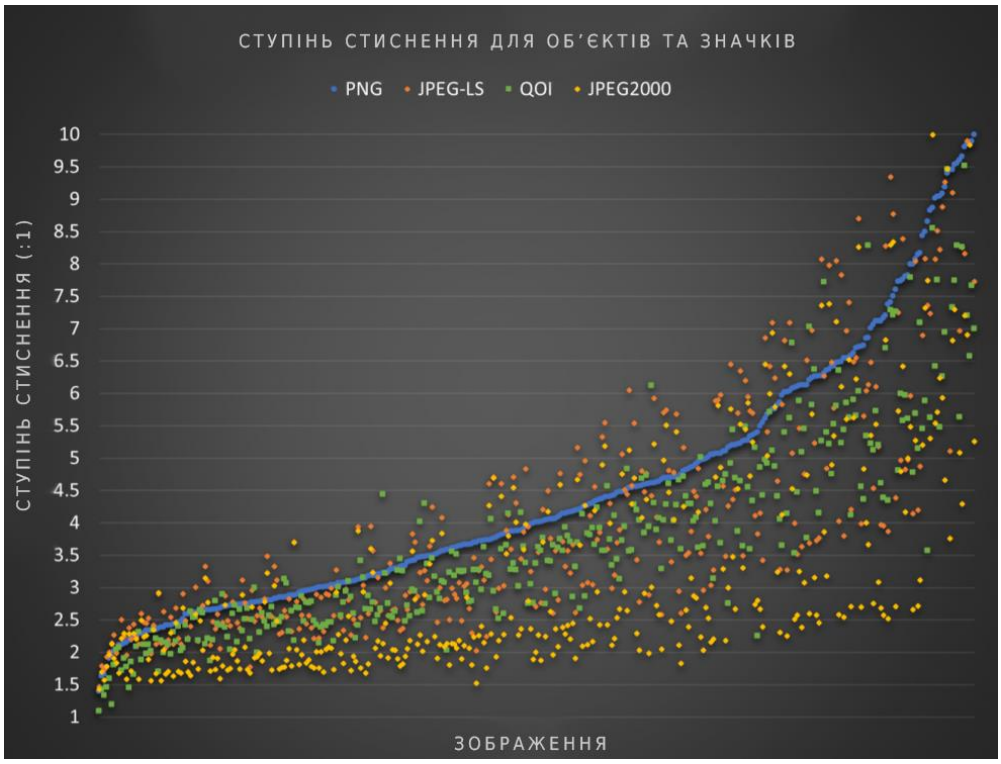

Рисунок 4.5 – Ступінь стиснення для об’єктів та значків

Включення деяких природних об’єктів, здається, надає перевагу JPEG-LS і JPEG2000 у цьому випадку, навіть якщо фон зображення в основному однорідний. JPEG2000 перевершує QOI з цієї причини, але PNG усе ще забезпечує найкраще CR у середньому. Сукупні результати надають перевагу JPEG-LS у цьому випадку, а продуктивність JPEG2000 і PNG порівнянна.

Рисунок 4.6 – Підсумок коефіцієнта стиснення для об’єктів та значків

Це, знову ж таки, свідчить про те, що PNG досягає кращого CR на зображеннях, які сильно стискаються, а також на менших зображеннях, таких як прості піктограми, які складають значну частину цього набору даних. JPEG-LS, з іншого боку, краще працює на більших і складніших зображеннях, таких як деякі природні об’єкти в цій категорії, що призводить до вищого CR при обчисленні байт за байтом у всій категорії.

Таблиця 4.3 – Підсумок коефіцієнта стиснення для об’єктів і значків

Формат	**PNG**	**JPEG-LS**	**QOI**	**JPEG2000**
Найкращий (x:1)	620,556	424,088	160,618	674,315
Найгірший (x:1)	1,397	1,470	1,090	1,426
Середнє (x:1)	14,837	10,284	9,478	7,777
Сукупність (x:1)	5,855	6,136	4,502	5,061

Текстури. Категорія текстур складається з 1385 зображень, взятих із наборів зображень textures_photo, textures_pk, textures_pk01, textures_pk02 і textures_plants. Підсумок цієї категорії наведено в таблиці 4.4, а також на рисунку 4.7, на рисунку 4.8 показано коефіцієнт стиснення зображення на формат усієї категорії.

Рисунок 4.7 – Підсумок коефіцієнта стиснення для текстур

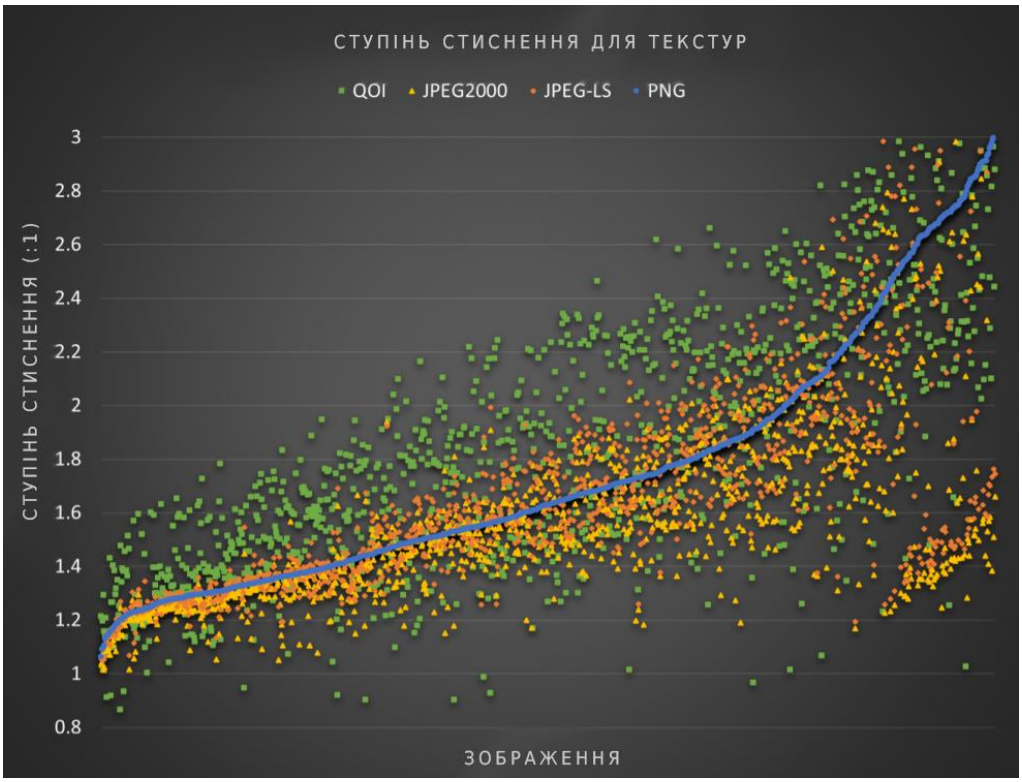

Рисунок 4.8 – Ступінь стиснення для текстур

Ця категорія складається з природних і штучних зображень, які використовуються як текстури для створених комп’ютером об’єктів, наприклад 3D-моделей. PNG має невелику перевагу над JPEG-LS, навіть якщо різні набори текстур демонструють різноманітність природних і синтетичних зображень.

Вивчення окремих наборів зображень, має дати більш чітку картину в цьому питанні. QOI добре працює в менш стиснутих зображеннях у цій категорії, хоча останні два формати перевершують його, оскільки досліджуються менш складні зображення.

Відстань між гіршим і найкращим випадками знову помітно велика для всіх форматів. Однак приблизно 80% зразків зображень призвели до CR близько або менше середнього значення.

Таблиця 4.4 – Підсумок коефіцієнта стиснення для текстур

Формат	**PNG**	**JPEG-LS**	**QOI**	**JPEG2000**
Найкращий (x:1)	214,170	270,066	126,355	128,881
Найгірший (x:1)	1,063	1,051	0,085	1,015
Середнє (x:1)	3,149	3,001	2,857	2,432
Сукупність (x:1)	2,444	2,322	2,193	2,164

Вибір відповідного формату зображення на основі типу даних програми може стати ключовим моментом у її остаточній реалізації. У той час як коефіцієнт стиснення для зображень із меншою ступенем стиснення не настільки чутливий до різних форматів, що використовуються, зображення, які віддають перевагу стисненню та, отже, отримають більші переваги від процесу, як правило, віддають перевагу певному кодеку залежно від типу зображення.

Для штучних зображень і зображень з меншою складністю та однорідніші областями загалом формат PNG залишається кращим форматом. Природні зображення, з іншого боку, де складність підвищена через часту зміну кольорів і відтінків кольорів, JPEG-LS є кращим рішенням, ніж решта форматів, з JPEG2000.

QOI не зовсім відповідає продуктивності решти перевірених форматів. Однак, враховуючи простоту його специфікації та здатність створювати легкі програмні додатки та апаратні реалізації з мінімальною площею, одержуючи при цьому порівняльні результати, існують програми, які, безсумнівно, виграють від цієї комбінації характеристик.

Одним із важливих зауважень щодо QOI є те, що існує можливість розширення зображення після кодування. Це можна спостерігати, дивлячись на найгірший випадок у представлених зведених таблицях вище. Спільне між цими випадками призводить до спостереження, що QOI більш чутливий до зображень, що містять випадковий шум або шумоподібні шаблони.

ВИСНОВКИ

У представленій кваліфікаційній роботі магістра було розглянуто принципи та засоби стиснення зображень, було досліджено алгоритм стиснення QOI та розроблено програмну частину для стиснення зображень. Результатами дипломної роботи є виконання поставлених задач а саме:

- виконано огляд предметної області та визначено можливості для застосування алгоритму QOI у стисненні зображень;
- досліджено алгоритм стиснення QOI та визначено сильні та слабкі сторони;
- досліджено отримані характеристики з точки зору обчислювальної складності та показників стиснення у порівнянні з іншими форматами;
- розробити програмне забезпечення для виконання перевірки та розглянути апаратні платформи, де може використовуватись досліджуваний алгоритм;
- представлено результати у вигляді таблиць та графіків порівняння та надано рекомендації з приводу застосування модифікацій у конкретних ситуаціях;
- розроблено питання з охорони праці та безпеки життєдіяльності.

QOI не стискає зображення так само добре, як оптимізований кодувальник PNG, і це нормально. Уже є формати зображень, які все одно стискають PNG. Перевага QOI полягає в його простоті.

Алгоритм має потужність індивідуального налаштування, таким чином, його можна застосувати для різних видів зображень з різними характеристиками. Крім того, QOI легко інтегрується з будь-якими проектами завдяки доступності джерела та простоті використання.

Однак, як і в будь-якого алгоритму стиснення, є деяка пожертва в області швидкодії. Зазвичай, чим більше ступінь стиснення, тим більш довгий час стиснення. Також слід пам'ятати, що чим більше розмір зображення, тим більше ресурсів потрібно для стиснення зображення.

В процесі виконання також було виконано частину з охорони праці. Були проаналізовані умови в приміщенні підприємства в якому проводились дослідження. Були надані рекомендації які б сприяли підвищенню продуктивності праці та якості продукції. Також було досліджено дії персоналу та роботодавця у надзвичайних ситуаціях, а саме під час повітряної тривоги.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Є. В. Дзяман, Я. М. Крайник, «Застосування алгоритму QOI у стисненні зображень,» Інформаційні технології та інженерія : тези доп. Всеукр. наук.-практ. конф. Миколаїв, 07–10 лют. 2023 р. Миколаїв : Чорном. нац. ун-т ім. Петра Могили, 2023. С. 70–71.

2. Y. Krainyk, “Combined Run-Length and Huffman Encoding for Image Compression,” ResearchSquare, 2022, doi: https://doi.org/10.21203/rs.3.rs-1982410/v1 (Дата звернення: 25.12.2022).

3. Díaz, Real-‘Time Hyperspectral Image Compression Onto Embedded GPUs’, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, № 12, с. 2792

4. Santosh Kumar B. P,Venkata Ramanaiah K., ‘An Efficient Hybrid Optimization Algorithm for Image Compression”, Multimedia Research, Vol.2,No.4, pp.1-11,2019.

5. K. S. Krasnikova, A. Kopaniev v., M. P. Vasylenko. SMART LEAKAGE MONITORING SYSTEM WITH BLYNK IoT INTEGRATION USING ARDUINO. Електроніка та системи управління. 2019. Вип. 1, № 59.

6. «QOI — The Quite OK Image Format» URL : https://qoiformat.org/ (Дата звернення: 25.12.2022).

7. «QOI-Specification» URL: https://qoiformat.org/qoi-specification.pdf (Дата звернення: 25.12.2022).

8. «Lossless Image Compression in O(n) Time» URL: https://phoboslab.org/log/2021/11/qoi-fast-lossless-image-compression (Дата звернення: 07.01.2023).

9. «The QOI File Format Specification» URL: https://phoboslab.org/log/2021/12/qoi-specification (Дата звернення: 07.01.2023).

10. «Compression algorithms» URL: https://www.prepressure.com/library/compression-algorithm. (Дата звернення: 07.01.2023).

11. «What is Image Compression?» URL: https://www.geeksforgeeks.org/what-is-image-compression/. (Дата звернення: 07.01.2023).

12. «phoboslab/qoi» URL: https://github.com/phoboslab/qoi. (Дата звернення: 15.01.2023).

13. «QOI vs PNG-spec - compare differences and reviews» URL: https://www.libhunt.com/compare-qoi-vs-PNG-spec. (Дата звернення: 15.01.2023).

14. «JPEG vs PNG vs GIF—which image format to use and when?» URL: https://imagekit.io/blog/jpeg-vs-png-vs-gif-which-image-format-use/. (Дата звернення: 15.01.2023).

15. «Huffman Coding» URL: https://www.programiz.com/dsa/huffman-coding/. (Дата звернення: 15.01.2023).

16. «Introduction to Data Compression» URL: https://www.mbit.edu.in/wp-content/uploads/2020/05/data_compression.pdf/. (Дата звернення: 27.01.2023).

17. «Visual Studio Code - Code Editing» URL: https://code.visualstudio.com/. (Дата звернення: 27.01.2023).

18. «Getting started with Visual Studio Code» URL: https://code.visualstudio.com/docs/introvideos/basics/. (Дата звернення: 04.02.2023).

19. «The QOI File Format Specification» URL: https://news.ycombinator.com/item?id=29625084. (Дата звернення: 04.02.2023).

20. «What is image compression and how does it work?» URL: https://www.techtarget.com/whatis/definition/image-compression/. (Дата звернення: 04.02.2023).

ДОДАТОК А

Лістинг коду:

```
void mqoi_u32_write(const uint32_t *n, uint8_t *dest)
{
    if (MQOI_ISHOSTLE)
    {
        *(uint32_t *)dest = MQOI_BSWAP32(*n);
    }
    else
    {
        *(uint32_t *)dest = *n;
    }
}
void mqoi_u32_read(const uint8_t *src, uint32_t *n)
{
    if (MQOI_ISHOSTLE)
    {
        *n = MQOI_BSWAP32(*(uint32_t *)src);
    }
    else
    {
        *n = *(uint32_t *)src;
    }
}
uint8_t mqoi_desc_verify(mqoi_desc_t *desc, uint32_t *w, uint32_t *h)
{
    if (desc->magic[0] != 'q' || desc->magic[1] != 'o' || desc->magic[2] !=
'i' || desc->magic[3] != 'f')
    {
        return MQOI_DESC_INVALID_MAGIC;
    }

    mqoi_u32_read(desc->width, w);
    mqoi_u32_read(desc->height, h);

    if   (desc->channels   !=   MQOI_CHANNELS_RGB   &&   desc->channels   !=
MQOI_CHANNELS_RGBA)
    {
        return MQOI_DESC_INVALID_CHANNELS;
    }

    if  (desc->colorspace  !=  MQOI_COLORSPACE_SRGB  &&  desc->colorspace  !=
MQOI_COLORSPACE_LINEAR)
    {
        return MQOI_DESC_INVALID_COLORSPACE;
    }

    return MQOI_DESC_OK;
}
void mqoi_dec_init(mqoi_dec_t *dec, uint32_t n_pix)
{
    memset(dec, 0, sizeof(mqoi_dec_t));

    dec->prev_px.a = 0xff;
    dec->pix_left = n_pix;
}
void mqoi_dec_push(mqoi_dec_t *dec, uint8_t byte)
```

```
      {
          dec->curr_chunk.value[dec->curr_chunk_head++] = byte;

          if (dec->curr_chunk_size == 0 && dec->curr_chunk_head == 1)
          {

              switch (dec->curr_chunk.head)
              {
              case MQOI_OP8_RUN_RGBA:
                  dec->curr_chunk_size = 5;
                  break;
              case MQOI_OP8_RUN_RGB:
                  dec->curr_chunk_size = 4;
                  break;
              default:
              {
                  dec->curr_chunk_size    =    1    +    ((dec->curr_chunk.head    &
MQOI_MASK_OP_2B) == MQOI_OP2_LUMA);
                  break;
              }
              }
          }
      }
      uint8_t mqoi_dec_take(mqoi_dec_t *dec, const uint8_t *bytes)
      {

          dec->curr_chunk.value[dec->curr_chunk_head++] = *(bytes++);

          switch (dec->curr_chunk.head)
          {
          case MQOI_OP8_RUN_RGBA:
              dec->curr_chunk_size = 5;
              break;
          case MQOI_OP8_RUN_RGB:
              dec->curr_chunk_size = 4;
              break;
          default:
          {

              dec->curr_chunk_size  =  1  +  ((dec->curr_chunk.head  &  MQOI_MASK_OP_2B)
== MQOI_OP2_LUMA);
              break;
          }
          }

          while (dec->curr_chunk_head < dec->curr_chunk_size)
          {
              dec->curr_chunk.value[dec->curr_chunk_head++] = *(bytes++);
          }

          return dec->curr_chunk_head;
      }
      MQOI_INLINE bool mqoi_dec_done(const mqoi_dec_t *dec)
      {
          return dec->pix_left == 0;
      }
```