НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і

спеціалізованих комп'ютерних систем

«На правах рукопису»
УДК 004.627

До захисту допущено:

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Системне програмування та спеціалізовані комп'ютерні системи»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Модифікація алгоритму стиснення QOI»

Виконав:
студент ІІ курсу, групи КВ-22 мп
Голуб Володимир Володимирович _____

Науковий керівник:
Доцент, кандидат технічних наук,
Клятченко Ярослав Михайлович _____

Рецензент:
Доцент кафедри ПЗКС,
Кандидат технічних наук
Заболотня Тетяна Миколаївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»»

Освітньо-професійна програма «**Системне програмування та спеціалізовані комп'ютерні системи»**

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Голуба Володимира Володимировича

1. Тема дисертації «Модифікація алгоритму стиснення QOI», науковий керівник дисертації Клятченко Ярослав Михайлович, кандидат технічних наук , доцент затверджені наказом по університету від «9» листопада 2023 р. №5217-с

2. Термін подання студентом дисертації 10 січня 2024

3. Об'єкт дослідження

- алгоритм стиснення даних QOI

4. Вихідні дані

- удосконалений алгоритм стиснення без втрат QOI

5. Перелік завдань, які потрібно розробити

- реалізувати алгоритм стиснення QOI

- створити модифікації алгоритму QOI

- провести тестування на однакових наборах даних

- провести порівняння всіх варіантів алгоритму.

- порівняти якість стиснення зображень у форматі qoi при стисканні в zip-архів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу - презентація

7. Орієнтовний перелік публікацій

- Прикладна математика та комп'ютинг. «XVI науково-практична конференція магістрантів та аспірантів ПМК-2023 факультету прикладної математики присвячена 125-річчю КПІ ім. Ігоря Сікорського».

- Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології №2 (10) (2023).

8. Дата видачі завдання 1 листопада 2022р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою МД	10.11.2022	
2.	Розроблення та узгодження технічного завдання	15.11.2022	
3.	Аналіз існуючих рішень	14.12.2022	
4.	Підготовка матеріалів першого розділу МД	18.05.2023	
5.	Аналіз алгоритму стиснення та його реалізація	16.09.2023	
6.	Підготовка матеріалів другого розділу МД	14.10.2023	
7.	Аналіз можливості модифікації та реалізації модифікованих алгоритмів	19.11.2023	
8.	Тестування та аналіз результатів	10.12.2023	
9.	Оформлення документації дипломної МД	16.12.2023	
10.	Проходження попереднього захисту	20.12.2023	

Студент Володимир ГОЛУБ

Науковий керівник Ярослав КЛЯТЧЕНКО

РЕФЕРАТ

Актуальність теми. З кожним роком обсяг даних, які генеруються та зберігаються, тільки зростає. Це стосується всіх типів даних тексту, зображень, відео, аудіо, сенсорних даних, та інших форм інформації. З розвитком технологій зображення та відео вищої роздільної здатності стають все популярнішими. Але ці файли є громіздкими та великими. Модифікація алгоритмів стиснення допомагає зменшити їх розмір, зберігаючи при цьому прийнятну якість мультимедійного контенту. Збільшення розміру даних призводить до більшого споживання мережевих ресурсів та пам'яті, що може стати проблемою для користувачів та організацій. Надлишковість даних може бути зайвим фактором в зберіганні та передачі інформації. Ефективні алгоритми стиснення даних можуть значно зменшити її, після чого для збереження на пристроях або при передачі по мережі знизиться витрати на пам'ять і пропускну спроможність мережі. Якщо врахувати споживання пам'яті та енергії обчислювальних пристроїв, то ефективні алгоритми стиснення можуть покращити продуктивність та зберігання даних на різних пристроях, включаючи мобільні телефони, сервери тощо. Отже, модифікація алгоритмів стиснення даних залишається актуальною та важливою темою в сучасному світі, де зростає обсяг та різноманітність інформації, а також де ефективне управління цією інформацією стає ключовим завданням для багатьох сфер життя та бізнесу.

Об’єктом дослідження є алгоритм стиснення зображень без втрат QOI.

Предметом дослідження є розробка вдосконаленого алгоритму стиснення зображень без втрат QOI.

Мета роботи полягає у підвищення ефективності стиснення алгоритму QOI. Аналіз алгоритму на можливість модифікації для збільшення його ефективності. Проведення тестів удосконаленого алгоритму та оцінка ефективності модифікацій.

Наукова новизна цієї роботи полягає в наступному:

1. Створення нового покращеного модифікованого алгоритму.
2. Аналітична оцінка ефективності модифікацій в порівнянні із початковим алгоритмів.

Практична цінність цієї роботи, це створені модифіковані алгоритми. Модифікації, що покращують ефективність роботи початкового алгоритму.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XVI науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2023 (Київ, 28-23 листопада 2023 р.).Також, в науковій статті, на сторінках журналу «Вісник Національного технічного університету "ХПІ". Серія : Системний аналіз управління та інформаційні технології»

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їхнє впровадження.

У першому розділі розглянуто існуючі алгоритми стиснення даних. Ознайомились із властивостями які має зображення.

У другому розділі. Детально розглянули опис алгоритму стиснення QOI. Ознайомились теоретично із архітектурою апаратних пристроїв та теоретично розглянули реалізацію кодера та декодер для алгоритму QOI. Розглянули способи модифікації різних алгоритмів. Розглянули дві модифікації алгоритму та визначили їх особливості.

У третьому розділі . Розглянули програмне середовище для розробки та вибір мови програмування. Детально розглянули реалізацію кожного із варіантів модифікації та основного алгоритму. Порівняли зміни в коді алгоритму із початковим. Описали теоретично вплив модифікації на результат.

У четвертому розділі Описали набори даних для проведення дослідження. Виконали експериментальне дослідження на різних наборах зображень. Експериментально підтвердити покращення алгоритмів та їх доцільність. Додатково провели дослідження на якість стиснення зображень у форматі qoi у zip-архівах.

У висновках представлені результати проведеної роботи.

Робота представлена на 90 аркушах, містить посилання на список використаних літературних джерел.

Ключові слова: стиснення даних, QOI, коефіцієнт стиснення, Python, PyCharm, PNG.

ABSTRACT

Actuality of theme. Every year, the volume of data that is generated and stored only grows. This applies to all types of text data, images, video, audio, sensory data, and other forms of information. As technology advances, higher resolution images and videos are becoming increasingly popular. But these files are bulky and large. Modification of compression algorithms helps to reduce their size, while maintaining an acceptable quality of multimedia content. Increasing data size leads to higher consumption of network resources and memory, which can be a problem for users and organizations. Data redundancy can be an unnecessary factor in information storage and transmission. Effective data compression algorithms can significantly reduce it, after which memory and network bandwidth costs will be reduced for storage on devices or for transmission over the network. Considering the memory and power consumption of computing devices, efficient compression algorithms can improve the performance and storage of data on various devices, including mobile phones, servers, and more. Therefore, the modification of data compression algorithms remains a relevant and important topic in today's world, where the volume and variety of information is growing, and where effective management of this information is becoming a key task for many areas of life and business.

The object of research is an image compression algorithm without loss of QOI.

The subject of the research is the development of an improved image compression algorithm without loss of QOI

The goal of the work is to improve the compression efficiency of the QOI algorithm. Analysis of the algorithm for the possibility of modification to increase its effectiveness. Conducting tests of the improved algorithm and evaluating the effectiveness of modifications.

Scientific novelty:

1. Creation of a new modified algorithm and its application in practice.

2. Analytical assessment of the effectiveness of modifications in comparison with the original algorithms.

Practical value of the results obtained in the work is that modified algorithms were created. Modifications that improve the efficiency of the initial algorithm. Testing and analysis was carried out to determine the effectiveness of modifications on certain types of images.

Approbation of work. The main provisions and results of the work were presented and discussed at the XVI Scientific Conference of Master's and Postgraduate Students "Applied Mathematics and Computing" PMK-2023 (Kyiv, November 28-23, 2023), as well as in a scientific article, on the pages of the journal, Bulletin KhPI National Technical University. Series: System analysis of management and information technologies.

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The introduction provides a general description of the work, assesses the current state of the problem, substantiates the relevance of the research direction, formulates the purpose and tasks of the research, shows the scientific novelty of the obtained results and the practical value of the work, provides information on the approbation of the results and their implementation.

In the first section, existing data compression algorithms are considered. We got acquainted with the properties of the image.

In the second chapter. The description of the QOI compression algorithm was considered in detail. We got acquainted theoretically with the architecture of hardware devices and theoretically considered the implementation of the encoder and decoder for the QOI algorithm. We considered ways to modify various algorithms. We considered two modifications of the algorithm and determined their features

In the third chapter. We considered the software development environment and the choice of programming language. The implementation of each of the modification options and the main algorithm was considered in detail. We compared the changes in the algorithm code with the initial one. The impact of the modification on the result was described theoretically.

In the fourth chapter, the datasets for the research were described. An experimental study was performed on different sets of images. Experimentally confirm the improvement of algorithms and their feasibility. In addition, research was conducted on the quality of image compression in qoi format in zip archives.

The results of the work are presented in the conclusions.

The work is presented on 90 sheets, contains links to the list of used literary sources

Keywords: data compression, QOI, compression ratio, Python, PyCharm, PNG.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

QOI (Quite Ok Image) – Скорочена назва алгоритму;

PNG–Portable Network Graphics – формат стиснення зображеня;

RLE –Run-Length encoding – спосіб стиснення зображення;

CR – Compression Rate – Рівень стиснення зображення.

ВСТУП

Щодня у світі збільшуються кількість даних, які люди зберігають, обробляють та використовують у житті. У наш час не існує і дня, щоб ми не зробили фотографію чи знімок екрану, під час навчання, листуванням із другом чи на роботі при обговоренні робочих питань. Вимоги до зображення зростають з кожним днем, звичайно розмір зображення має бути щонайменшим, але водночас якість на вищому рівні. Щоб максимально можливо забезпечити дані вимоги створюються різноманітні алгоритми стиснення зображення, тексту, аудіо чи відео даних.

Стиснення зображення – це процес, під час якого файл із графікою за допомогою одного чи більше алгоритмів зменшують його розмір, зберігаючи при цьому прийнятну якісь зображення. Чим менший розмір зображення тим вища швидкість передачі по мережі, більше вільного місця на пристрої, більше даних буде збережено.

Визначити який алгоритм є найкращим серед всіх є надскладною задачею, тому що кожен алгоритм має свої особливості та недоліки на різних видах зображення. Алгоритм повинен пройти складних процес, який включає в себе різнобічні дослідження на різних графічних файлах. Фактори, що повинен враховувати алгоритм це на сам перед коефіцієнт стиснення, час стиснення, об’єм використання оперативної пам’яті пристрою. Ці всі змінні формують результат, який і покаже чи буде алгоритм популярний чи залишиться невідомим.

Завданням дипломного проекту є підвищення ефективності алгоритму стиснення зображень QOI. Вдосконалення на основі аналізу роботи алгоритму, перевірити якість стиснення зображень у форматі qoi в zip-архівах.

АНАЛІЗ ІСНУЮЧИХ АЛГОРИМТІВ СТИСНЕННЯ

1.1 Змістовний опис та аналіз предметної області

Стиснення даних – це процедура перекодування даних, яка проводиться із метою зменшення їхнього обсягу, розміру, об’єму [1].

Тип даних – це характеристика, яка призначається об’єкту(наприклад, змінна, функція, поле запису, константа, масив тощо) явно чи неявно. Тип даних визначає набір допустимих значень, їх зберігання, розмір виділеної пам’яті та набір операцій, які можна виконувати над даними [2].

Текстові дані – це тип даних, який складається із послідовності символів, що представляють слова, символи чи числа. Текстові дані зберігаються у певному форматі та відповідно формату кодуються. Кожен формат має свою відповідну таблицю кодування кожного символу, цифри. Числові значення кодуються відповідно до їх представлення у 16-річній системі числення чи іншому форматі, наприклад у двійковому форматі. Символи кодуються теж числами, але вже відповідно до таблиці, яка у кожного формату має свої відмінності..

Фото дані – це тип даних, який складається із набору пікселів. Пікселі мають в собі інформацію колір, та насиченість, що представляються у числові значення.

Відео дані – це тип даних, який складається із набору послідовності зображень, що відображаються із певною частотою.

Всім вище зазначені типам даних притаманна природна надлишковість (рисунок 1.1), в особливому порядку це фото та відео дані. Відеодані мають значно більшу надлишковість, ніж графічні дані, і, в свою чергу, графічні дані є більш надлишковими, ніж текстові дані.

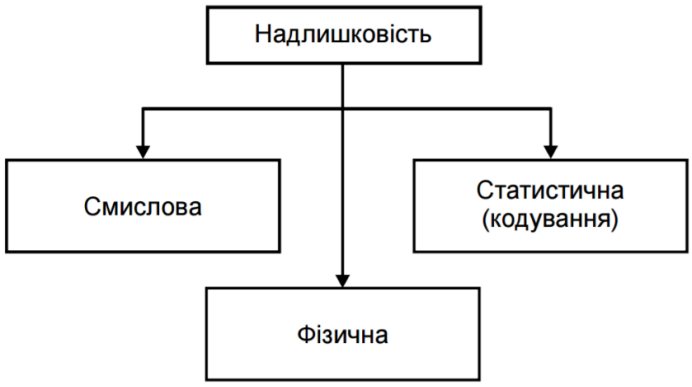


Рисунок 1.1 – Види надлишковості

Надлишковість даних – надлишковість даних, це присутність повторюваних даних у фрагменті. Для кодування можемо визначити наступний термін для поняття надлишковість даних. Надлишковість даних – це відношення між довжиною коду та кількістю інформації, яку він несе. Чим буде більше надлишковість тим більше коду потрібно буде для передачі чи зберігання інформації. Надлишковість даних впливає на вибір системи кодування.

Системи кодування можуть бути різними, включаючи звичайні мови спілкування, які служать системами кодування для виразу понять та ідей. Наприклад, якщо розглядати кодування текстових даних українською мовою порівняно з кодуванням аналогічних даних англійською мовою, можна визначити, що перша буде більше надлишковою в порівнянні з другим.

Це свідчить про те, що для подібних обсягів інформації українська мова, через свою будову та особливості, використовує більше ресурсів для кодування та збереження інформації порівняно з англійською мовою. Такий вплив природи даних та системи кодування на надлишковість даних важливий для розуміння та оптимізації процесів збереження та передачі інформації в різних мовних середовищах.

Надлишковість даних в людському середовищі є звичним явищем. Надлишковість даних збільшує якість інформації для розуміння людини, однак в межах передачі, зберіганні інформації, надлишковість є основною негативною характеристикою. Комп’ютери потребують більше пам’яті для зберігання

надлишкової інформації, це впливає на вартість комп’ютерів. Якщо передавати інформацію по мережі, великий розмір використано більше даних, більше часу, та ще і кожен байт переданої інформації має свою власну ціну при транспортуванні.

Значну роль ця проблема відіграє якщо йдеться про обробку великих об’ємів інформації, тоді як обсяг об’єму носіїв даних обмежений. У зв’язку з чим, гостро постає проблема позбавлення надлишкової інформації. Вирішення задачі по стисненню даних лягає в першу чергу на плечі комп’ютерних інженерів, інженерів корпорацій та фірм, що займаються виготовленням носіїв, обробки та зберігання даних. Із сучасним розвитком технології, та загальному розвитку людства, та маючи при собі тільки комп’ютер, логічне мислення, чи практичний досвід, розробниками алгоритмів, які позбавляють надлишкової інформації, або краще сказати стискають дані може стати і є кожен бажаючий.

Також процес стиснення даних ще називають – архівування, такий термін доволі часто використовують до вже оброблених файлів. Результатом такого стиснення називають – архів, а програму що виконує стиснення – архіватор.

Стиснення даних залежить від того де розміщуються дані, та які наступні дії будуть виконувати над даним. Отже розрізнити можна за такими критеріями:

1. ***Стиснення файлів***: З метою зменшити розмір файлу, щоб в подальшому передати їх по мережі чи інших типах, каналах зв’язку. Або для зберіганні на носії з обмеженим об’ємом пам’яті.

2. ***Стиснення (архівування) папок***: з метою зменшення обсягу папок для довготривалого зберігання. Наприклад, резервне копіювання даних;

3. ***Стиснення (ущільнення) дисків***: з метою підвищення ефективності використання дискового простору шляхом стиснення даних при записі їх на носії інформації (як правило, засобами операційної системи).

1.2 Стиснення зображення

1.2.1 Комп’ютерна графіка

Комп’ютерна графіка – це зображення, що створюються, зберігаються, редагуються і відображаються за допомогою комп’ютерної графіки [3].

Графічна інформація – графічні зображення, що представлені цифровими кодами й зберігаються у файлах певного формату.

Комп’ютерна графіка ділиться на три види (рисунок 1.2).

Рисунок 1.2 – Види комп’ютерної графіки

Растрова графіка – зображення що відтворюються за допомогою пікселів [3]. Такі зображення отримуються за допомогою фотоапаратів, сканерів, графічних редакторів.

Піксель – мінімальний елемент растрового зображення, колір і яскравість якого можна незалежного від інших. Чим більша кількість наявна у зображенні, ти це зображення є якіснішим [3].

Векторна графіка – це зображення, які використовують елементарні геометричні об’єкти у своєму представленні. До елементарних графічних об’єктів відносять такі прості фігури, як точки, відрізки, прямі, дуги, кола і еліпси, квадрати і прямокутники, овали і кола і, як загальний випадок, криві деякого порядку. Також використовується більш складні фігур, ламані лінії, криволінійні відрізки і т. д.

Векторна графіка при зберіганні займає менше місця ніж растрова, по тій простій причині, що в основі векторної лежить математична модель зображення,

тобто математичні формули. Особливістю векторного зображення, є те що розмір зображення не залежить від розміру файлу. Як що ми будемо зберігати прямокутники різних розмірів і кольору, розмір кожного файлу буде однаковим.

Фрактальна графіка – це технологія створення зображень на основі фракталів.

Фрактал – це об’єкт, окремі елементи якого успадковують якості батьківських структур. Слово «фрактал» походить від латинського слова «fractus» і в перекладі означає складатися із фрагментів. Особливість фракталів– це самоподібність. Об’єкт називаються самоподібним, коли збільшені частини об’єкту схожі на сам об’єкт і один до одного [3]. Сам об’єкт описується математичними рівняннями. У природі фрактальним об’єктом є сніжинка.

1.2.2 Колірна модель

Для відображення кольорів у зображеннях використовуються різні колірні палітри.

Колірна модель – абстрактна модель опису кольорів, у вигляді, кортежів(наборів) чисел, зазвичай трьох або чотирьох значень, званих колірними компонентами або колірними координатами [4].

Різновиди колірної моделі:

- RGB (рисунок 1.3),
- CMYK (рисунок 1.4),
- HSB (рисунок 1.5).

Колірна модель RGB – це така модель в якій кожен колір представлений, як поєднання червоного, зеленого і синього. Три головні кольори утворюють ще три проміжні відтінки. Значення кожного головного кольору варіюється від 0 до 255. При максимальних значеннях всіх кольорів, буде відображено білий колір [5].

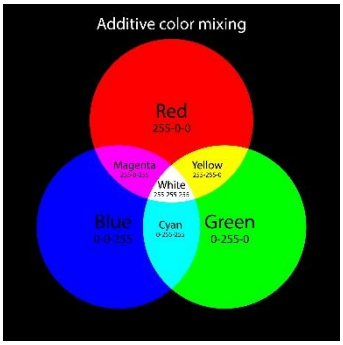


Рисунок 1.3 – Колірна модель RGB

Колірна модель CMYK – колірна модель, що враховує кількість відбитого світла, а не поглиненого. Основні кольори ціан, маджента, жовтий, та чорний [5].

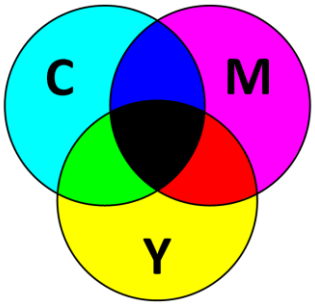


Рисунок 1.4 – Колірна модель CMYK

Колірна модель HSB – колірна модель, яка є аналогом RGB, тільки відмінність полягає в тому, що колір визначається як співвідношення тону(Hue), насиченістю(Saturation) і яскравістю(Brightness). Колір задає тон. Насиченість додає до нього білу фарбу, а яскравість – чорну [5].

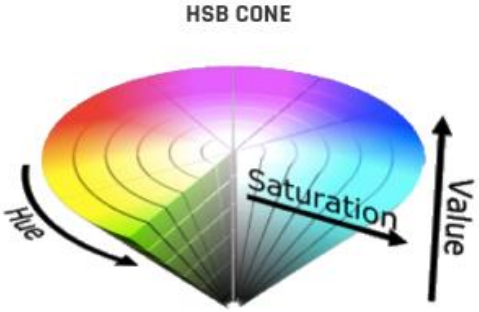


Рисунок 1.5 – Колірна модель HSB

1.2.3 Стиснення зображення

Постійне покращення якості зображення, збільшення роздільної здатності, програмного забезпечення яке обробляє графічні зображення, вимагають сталого постійного розвитку, у напрямку пошуку ефективного методу кодування зображень. Головна мета яких є найменший розмір зображення не втрачаючи оригінальну якість зображення. Співвідношення розміру вхідного зображення оригінального зображення до розміру вже закодованого зображення є одним із ключових основних факторів для визначення ефективності того чи іншого формату зображення. Даний показник грає вирішальну роль для застосування у професійних програмах обробки графіки, передачі зображення по мережі, завантаження та відображення на будь якому сайті, мобільного додатку тощо. Формат зображення пливає і на швидкість завантаження веб-сторінки.

Існують дві категорії стиснення зображення:

- Стиснення без втрат,
- Стиснення із втратами.

Стиснення без втрат – це стиснення що гарантує зменшення розміру зображення зберігаючи при цьому якісь вхідного (оригінального) зображення.

Стиснення зображення – це стиснення що зменшує розмір зображення нехтуючи даними деяких пікселів та кольорів, через що розмір вихідного зображення значно менший ніж розмір вхідного, але і якість теж різна.

Часто використовуючи стиснення із втратами, особливо при стисненнях у декілька разів чи десятки разів зображення спотворюються, інколи так, що вже не можна розрізнити що на ньому початкове зображення. Використовуючи стиснення із втратами із розумом якість зберігається, втрати не є помітними та критичними для кінцевого застосування.

Ступінь стиснення залежить від формату стиснення зображення і відрізняється один від одного. Який саме алгоритм обрати для використання може зіграти вирішальну роль в кінцевому використані вихідного файлу та вплинути на продуктивність програмного продукту або веб-ресурсу.

Розглянемо проблеми стиснення із втратами. Основною їх проблемою є безповоротність. Маючи вже стиснуте зображення відтворити початкове зображення неможливо. Кожне повторне використання стиснення із втратами над одним і тим же зображенням, спричиняє спотворення початкової інформації або навіть і її втрату. Незважаючи на це такі алгоритми набули ефективного застосування в мережі Інтернет де втрата незначної кількості даних не є критичною, а інколи і потрібною для швидкої передачі даних та ефективного відображення сторінки.

Гарним представником стисненням із втратами є JPEG (рисунок 1.6). Формат застосовується для зберігання цифрових фотографій, відображення картинок в інтернеті та відображення напівтонові ілюстрацій та графічної інформації зі плавним переходом тонів. Формат підтримується практично усіма графічними редакторами інструментами та програмами. Формат має три рівні стиснення зображення: колірна субдискретизація, дискретне косинусне перетворення і квантування та довжина циклу, дельта і кодування Гоффмана[6].

Рисунок 1.6 – Зображення формату JPEG

Інший спосіб зменшення зображення це стиснення без втрат. Такий метод стискає зображення без втрат якості та без втрат даних, завдяки чому стиснуте зображення без проблем можна відновити до початкового, оригінального зображення. Якщо порівнювати розмір зображення після стиснення даних із

втратами та без втрат, то розмір першого буде меншим ніж другого. Стиснення без втрат використовується в тих випадках де якість зображення є важливішою за його розмір, час передачі по мережі.

Серед алгоритмів стиснення без втрат найпоширеніший є PNG. В його основі лежить алгоритм стиснення Deflate [3]. Він зменшує розмір файлу пошуком шаблонів і стиснення цих шаблонів разом. Формат був винайдений для заміни вже існуючого формату GIF. Через свою не популярність у перші роки існування та рідкісне використання формат не підтримувався браузерам, та з часом став одним із масових алгоритмів які використовує користувач. PNG формат підтримує кольори що базуються на основі палітри (32-розрядний RGBA або 24-розрядний RGB), колірні палітри та простори RGBA та RGB. PNG також має перевагу в тому, що підтримує параметри прозорості, в тому числі альфа-каналу (рисунок 1.7).

Рисунок 1.7 – PNG стиснення

Хоча зазвичай файли у форматі PNG є більшими за файли у форматі JPEG, вони широко використовуються на веб-сайтах у випадках, коли потрібна більша деталізація зображень, це можуть бути логотипи, значки, скріншоти або зображення з текстом.

1.3 Методи стиснення зображення

Розглянемо загальний процес стиснення зображення. В комп'ютері зображення це вектор пікселів, пікселі у свою чергу це набір бітів. Над цими бітами виконуються певні математичні дії та перетворення, завдяки яким і відбувається процес стиснення даних. Пристрій, програма, функція яка виконує стиснення даних називають кодер. Декодер – виконує протилежну функцію кодеру.

У процесі стиснення над зображеннями можуть виконати наступні методи:

- квантування зображення ,
- кодування послідовностей,
- інші методи.

1.3.1 Квантування зображення

Квантування зображення — це обмеження або поділ кількості потенційних значень кожного пікселя зображення. Це досягається шляхом заміни кожного пікселя на значення, яке ближче до нього відповідає з обмеженого набору [4]. Такий спосіб зменшує розмір зображення, зменшуючи роздільну здатність або колірну модель. Квантування зображень є важливим етапом в алгоритмах стиснення зображень, таких як JPEG.

У процесі квантування різні рівні інтенсивності групуються в один рівень на основі математичної функції, визначеної на пікселях. Для отримання нового рівня зазвичай беруть фіксований розмір фільтра «m», ділять кожен значення фільтра на «m», округлюють до найближчого цілого числа та знову множать на «m» [7]. Функція квантування має наступний вигляд:

$$[pixelvalue/m] * m \quad (1)$$

Нехай у нас m=9, тоді квантування відбуватиметься так як зображено на рисунку 1.9.

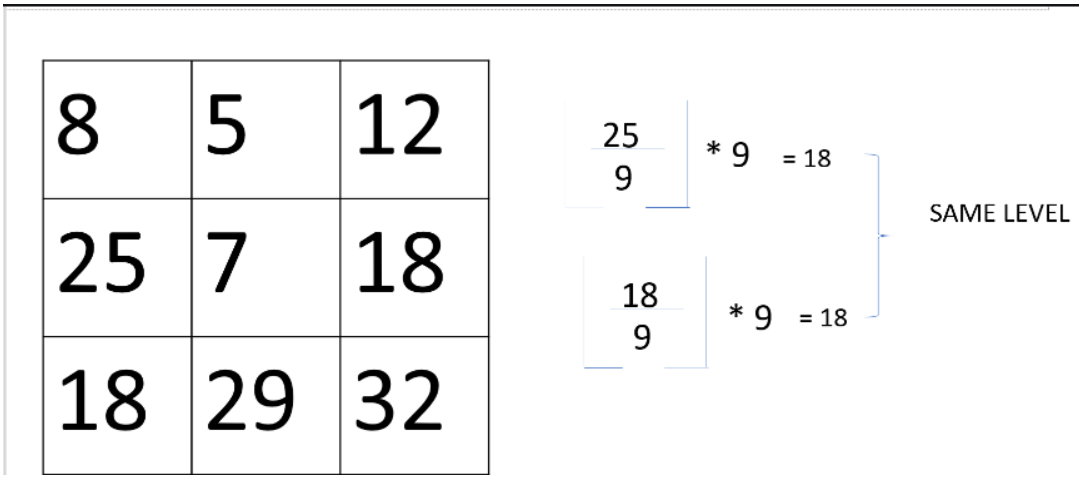


Рисунок 1.9 – квантування пікселів, де m=9.

Під час цього процесу втрачається деяка інформація, внаслідок чого роздільність та точність зображення можуть зменшитись. Наприклад ми маємо зображення, що має 256 рівні сірого кольору, і що разу ми будемо застосовувати фільтр і зменшитимемо рівні сірого у двічі. У такому випадку ми будемо стискати зображення за рахунок зменшення колірної моделі.

Рисунок 1.10 – Приклад якості зображення при зменшенні колірної моделі.

Із рисунку 1.10 бачимо, що від 256 до 32 рівня, для людського ока практично непомітна різниця у зображенні, хоч і було стиснуто зображення. При подальшому застосування фільтра, якісь погіршується. Якщо ми зменшимо

рівень до одного відтінку сірого ми втратимо повністю зображення. Отже з правильним вибором параметрів квантування можна забезпечити збереження досить непоганої якості зображення з прийнятним рівнем стиснення [8].

1.3.2 Кодування послідовностей символів

Основною ідеєю кодування послідовностей, це створення різних кодових слів для відповідного символу. Завдяки чому в кінцевому результаті досягається зменшення кількості бітів .

Такою технікою стиснення користуються такі алгоритми:

- Кодування Хафмана зі змінною довжиною,
- Кодування довжин серій.

Розглянемо детально кодування послідовностей символів на основі алгоритму кодування Хафмана. Алгоритм має дві складові: кодове дерево та відображення код-символ.

Кодове дерево– це бінарне дерево, де кожному листу відповідає символ вхідного алфавіту, а кожен внутрішній вузол має вагу, рівну сумі ваг свої дітей. Вага вузла може бути частотою або кількістю входжень символу в повідомленні. Для побудови кодового дерева алгоритм Хафмана використовує жадібний підхід: на кожному кроці він обирає вузол із найменшою вагою та об’єднує їх у новий вузол із новою вагою рівною їх сумі. Процес повторюється, поки не залишиться лише один вузол, це і є корінь дерева [9].

Для побудови відображення код-символ алгоритм Хафмана використовує шляхи від листів до кореня дерева: на кожній гілці він призначає біт від 0 до 1, так. що код символу складається із послідовності бітів, які відповідають гілкам на шляху.

Наприклад є повідомлення, для якого відповідає наступна таблиця частот

Таблиця 1– Таблиця частот символів

Кількість повторів	15	7	6	6	5
Символ	A	B	C	D	E

Процес побудови дерева Хафмана зображено на рисунку 1.3.3.

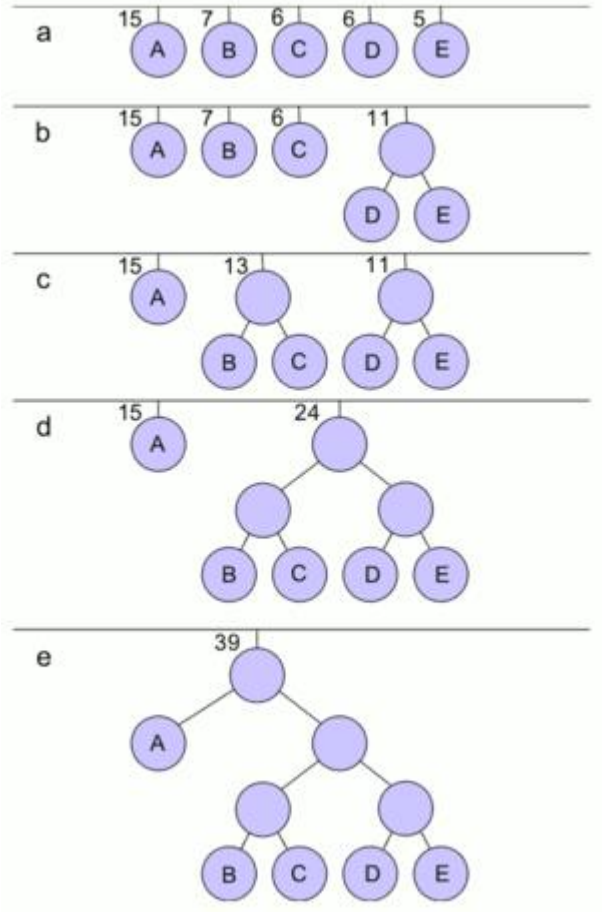


Рисунок 1.10 – Поетапна побудова дерева Хафмана.

Отже таким чином для даної таблиці символі (див табл.1), коди Хафмана відображені у таблиці 2.

Таблиця 2 – Таблиця кодів Хафмана

Символ	A	B	C	D	E
Кодування	0	100	101	110	111

Довжина тексту із таблиці 1 може без кодування займатиме 117 бітів, 3 біти на символ, а після кодування Хафмана довжина сягатиме 87 бітів. Проте однак для декодування обов’язково потрібно декодер повинен знати таблицю частот, отже до довжини 87 бітів ще потрібно попереду додати закодовану таблицю.

1.4 Критерії оцінки методів ущільнення

Для алгоритмів ущільнення даних виділяють наступні основні властивості:

- коефіцієнт (якість або ступінь) ущільнення, тобто відношення
- довжини (в бітах) ущільненого представлення даних до довжини вихідного представлення;
- швидкість кодування і декодування, що визначається часом, який витрачається на кодування і декодування даних;
- обсяг необхідної пам’яті.

В області ущільнення даних, досить часто зустрічається закон важеля (див. рис. 2): алгоритми, що використовують більше ресурсів (часу і пам’яті), зазвичай досягають кращої якості ущільнення, і навпаки: менш ресурсномісткі алгоритми за якістю ущільнення, я правило, поступаються більше ресурсномістким.

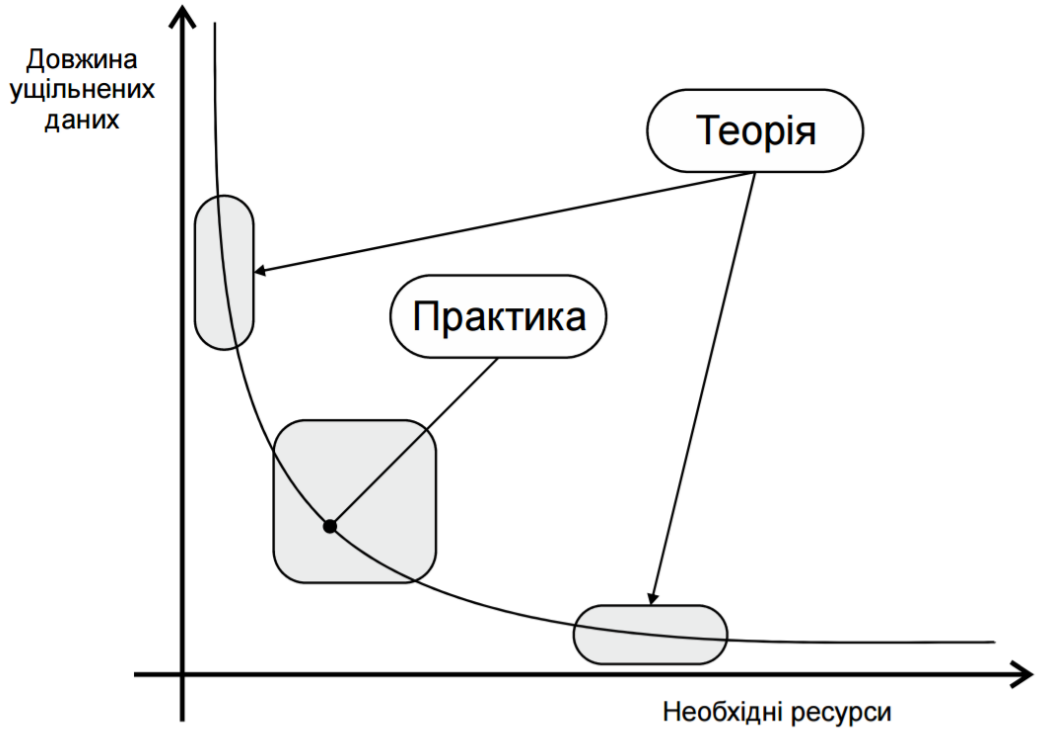


Рисунок 1.11 – Співвідношення якості ущільнення та використаних ресурсів

Таким чином, побудова оптимального з практичної точки зору алгоритму ущільнення даних є досить нетривіальним завданням, тому що необхідно домогтися досить високої якості ущільнення(необов’язково оптимального з теоретичної точки зору) при невеликому обсязі використаних ресурсів.

Звичайно, що критерії оціни методів ущільнення з практичної точки зору сильно залежить від передбачуваної області застосування. Наприклад, якщо використовувати у системах реального часу, тоді нам необхідно забезпечити високу швидкість кодування і декодування; для вбудованих систем критичний параметр є обсяг необхідної пам’яті; для систем довгострокового зберігання даних – коефіцієнт ущільнення або швидкість декодування або ж одночасно і коефіцієнт ущільнення та швидкість декодування і т. д.

1.5 Аналіз існуючих алгоритмів стиснення

1.5.1 JPEG стиснення

JPEG розшифровується як Joint Photographic Experts Group, це комітет стандартизації який і винайшов даний формат стиснення у 1992 році [6]. За мету комітет обрав досягти зменшити максимально розмір файлу природних фотографій, так щоб зберегти якісь зображення, яке б зорова система людини сприйняла на нормальному рівні.

Стиснення зображення форматом JPEG несе за собою втрати дані, коли ми розпаковуємо зображення, у розпакованому файлі присутня буде втрата деталей та чіткості порівнюючи із оригінальним зображенням.

Спрощений опис алгоритму кодування у форматі JPEG:

1. Перетворення зображення з колірної моделі RGB в YCrCb;
2. Субдискретизація компонентів колірності усередненням груп пікселів;
3. Застосування дискретних косинусних перетворень для зменшення надлишковості даних;
4. Квантування кожного блоку коефіцієнтів дискретних косинусних перетворень із застосуванням вагових функцій, що оптимізовані з урахуванням візуального сприйняття людиною;
5. Кодування результуючих коефіцієнтів(даного зображення) із застосуванням алгоритму Хафмена для видалення надлишковості інформації [6].

Стиснення зображення алгоритмом JPEG дозволяє стиснути у відношенні 20:1 та 25:1 . У деякі програми дозволяються вибрати ступінь стиснення, до прикладу Adobe Photoshop.

1.5.2 Flate/deflate стиснення

Алгоритм deflate це стиснення без втрат, в основі якого лежить два інші алгоритми стисненняб це кодування Хаффмана та стиснення LZ777. Розуміння роботи цих двох алгоритмів, дасть повне розуміння і алгоритму стиснення deflate/flate[Deflate].

Deflate – можна вважати це є розумний алгоритм, що вміє адаптувати спосіб стиснення даних до самих фактичних даних. Алгоритм має три варіанти стиснення:

- Практично не стиснутий. Вибір даних, які вже стиснено. Дані які зберігаються таким варіантом дещо розширяться, але проте не сильно, як якщо б вони вже були стиснуті та використали один із методів стиснення.
- Спочатку стиснення за допомогою LZ77, наступний крок – стиснення з модифікованою версією кодування Хаффмана. Дерева, що використовуються для стиснення визначаються специфікацією deflate. Таким чином, щоб зберегти ці дерева, не потрібно використовувати додаткову пам’ять.
- Спочатку стиснення за допомогою LZ77, далі стиснення знову алгоритмом Хаффмана з деревами, які створює алгоритм та зберігає їх разом із даними.

Після чого іде розбивка на блоки і кожен блок може використати один із режимів стиснення. Якщо алгоритм розуміє що наступні дані щоб стиснути потрібно змінити варіанти стиснення, він закриває поточний блок і починає новий обираючи новий варіант стиснення [10].

1.5.3 LZ77 стиснення

Алгоритм стиснення LZ77 є одним із методів стиснення без втрат, який використовується для зменшення об’єму даних шляхом виявлення і заміни повторюваних послідовностей символів у вихідному тексті. Цей алгоритм був розроблений Абрахом Лемпелом і Якобом Зівом у 1977 році і отримав назву їх імен [11].

Основна ідея алгоритму LZ77 полягає втому, щоб замінити повторювані блоки тексту короткими вказівниками на попередні блоки тексту, які вже зустрічались в поточному тексті. Це дозволяє досягнути стиснення без втрат даних [11].

Основні кроки алгоритму LZ77:

- Пошук в тексті найкоротшої підряд послідовності символів.
- Після знаходження такої послідовності. Створення вказівника, який містить відстань до початку цієї послідовності в поточному тексті довжину повторюваної послідовності.
- Додавання цього вказівника до стисненого потоку і знову повторення попередніх кроків для решти тексту

Алгоритм продовжує виконували ці кроки до тих пір, поки не пройде весь вихідний текст результатом роботи алгоритму LZ77 є стиснутий потік, який може бути подано як послідовність вказівників і літералів (символи, які не можуть бути стиснуті) [11].

1.5.4 Алгоритм стиснення QOI

Алгоритм стиснення QOI був створений у 2021 році Домініком Щаблевським. Алгоритм дивує своєю простотою та як заявляє автор за деяких обставин не гірше, а навіть краще алгоритму стиснення без втрат PNG. Якщо порівнювати розміри зображення PNG та QOI, то другий алгоритм дещо програє, але не багато [12].

Алгоритм проходиться по зображенню лише один раз. Під час проходу по пікселях він запам’ятовує певну кількість пройдених у буфер пам’яті.

Кожен піксель може бути закодований одним із 4-х варіантів [13]:

- Якщо попередній піксель дорівнює поточному, та записуємо у кінці кількість повторів, замість запису повторюваних пікселів;
- Якщо поточний піксель не рівний попередньому, перевіряємо чи він зустрічався у буфері перед цим, та записуємо його індекс;
- Якщо перші два варіанти не підходять. Враховуючи, що у більшості зображень сусідні пікселі практично не відрізняються по значеннях, ми записуємо тільки різницю кольорів із попереднім.
- На останок, якщо різниця велика, ми записуємо інформацію про піксель без змін та переходимо до наступного.

Висновки до розділу 1

Сфера обробки та передачі даних постійно розвивається. Зростання обсягів даних, особливо в мультимедійних форматах, робить актуальним пошук більш ефективних методів стиснення. Модифікація і покращення існуючих алгоритмів стиснення можуть допомогти зменшити розмір файлів та підвищити швидкість передачі.

Завдяки покращенню алгоритмів стиснення можна досягти кращої ефективності використання ресурсів, таких як обсяги сховищ та пропускної

здатності мережі. Це особливо важливо в умовах, коли ми маємо обмежені ресурси та потребу у максимальному використанні їхнього потенціалу.

З точки зору збереження інформації алгоритми стиснення дозволяють зберігати інформацію в більш компактному вигляді без втрати якості. Це може бути важливою задачею для збереження архівів, передачі великих обсягів даних через мережу або створення бекапів.

Алгоритми стиснення мають широке застосування в різних галузях, включаючи фотографію, відео, аудіо, медичну обробку даних та інші. Знання про модифікацію алгоритмів може бути корисним для вдосконалення та оптимізації різних проектів.

В даній роботі будемо вносити модифікації в алгоритм стиснення QOI, проведемо аналіз відсотку стиснення файлів формату QOI в застосунках для стиснення, в поєднанні з іншими алгоритмами стиснення.

2. АНАЛІЗ АЛГОРИТМУ QOI ТА АПАРАТНА РЕАЛІЗАЦІЯ

2.1 Детальний огляд алгоритму QOI

Розглянемо детально логіку роботи алгоритму стиснення QOI, його елементів та структур. Отже, згідно специфікації, для того щоб стиснути зображення достатньо лише одного проходу [13].

Основні складові :

- Заголовок – 14 байтів;
- Тіло (будь яка кількість фрагментів даних), розмір фрагментів – кратне 8 бітам ;
- Заключний маркер ;

Заголовок складається із наступних частин:

- magic – “магічні” байти “qoif”, ці байти є ідентифікатором для декодера.
- width – ширина зображення в пікселях, для визначення загальної кількості пікселів, необхідна умова для декодування .
- height – висота зображення в пікселях, для визначення загальної кількості пікселів, необхідна умова для декодування .
- channels – значення, що визначає кодування фрагмента коду, повністю залежить від інформації про піксель і варіюється в залежності від наявності альфа-каналу. Може містити два значення: 3 відповідає RGB, а 4 – RGBA.
- colorspace – 0 - sRGB з лінійною альфою ; 1 - всі канали лінійні,
- channels та colorspace - тільки інформативні поля.

Таблиця 3 – Таблиця розміру кожного елемента заголовку формату qoi

Magic’qoif’	Wight	Height	Channels	colorspace
4 Byte	4 Byte	4 Byte	1 Byte	1 Byte

На початку роботи алгоритм визначає наявність чи відсутність альфа-каналу. В залежності від цього використовується один із варіантів кодування піксельних послідовностей [13].

Якщо альфа-канал присутній, то фрагмент QOI_OP_RGBA кодується, як показано на рисунок 2.1.

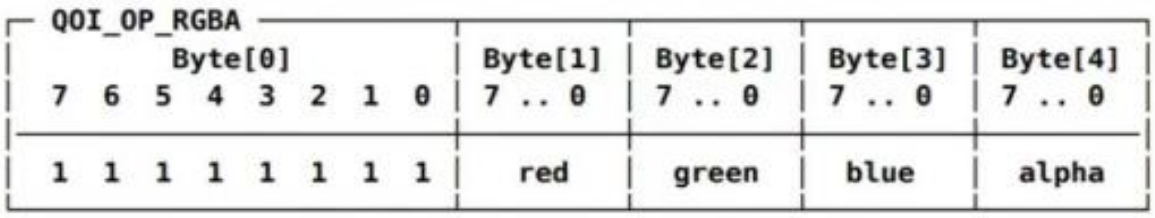


Рисунок 2.1 – Фрагмент QOI_OP_RGBA

Без альфа каналу фрагмент QOI_OP_RGB кодується так, як зображено на рисунку 2.2

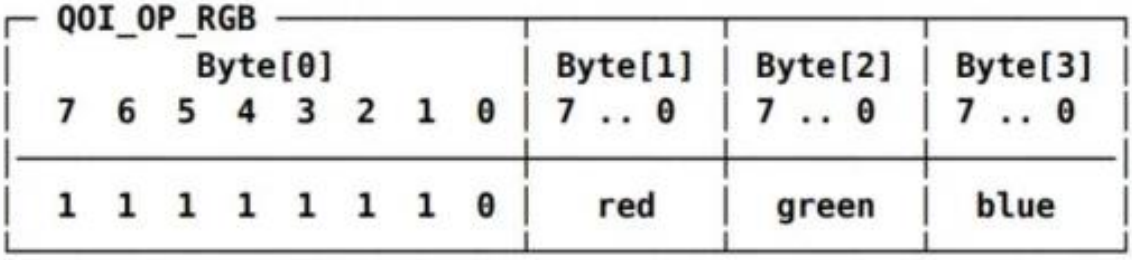


Рисунок 2.2 – Фрагмент QOI_OP_RGB

Проходячи по пікселях, алгоритм може кодувати поточний піксель одним із чотирьох варіантів. Також, під час проходження по поточних пікселях зображення в проміжному масиві завжди зберігається 64 попередньо пройдених значень пікселів.

Для першого варіанту перевіряється значення кольору поточного пікселя. Якщо воно дорівнює значенню попереднього пікселя зображення, то при цьому збільшується значення змінної пройденої довжини зображення − змінну run збільшуємо на одиницю та переходимо до наступного пікселя. Таку послідовність дій виконуємо до тих пір, поки поточний піксель не буде відрізнятися від попереднього. Після знаходження чергової відмінності від

попередніх пікселів, у фрагмент QOI_OP_RUN заноситься значення run – кількість повторів у бітовому представленні (рисунок 2.3).

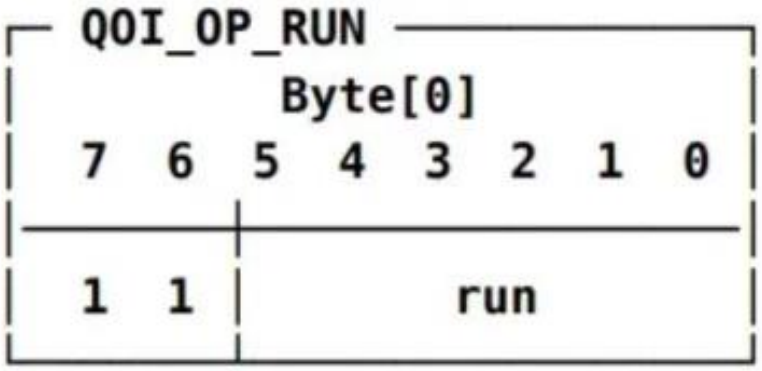


Рисунок 2.3 – Фрагмент QOI_OP_RUN

Другий варіант. Перевіряється попередні 64 пікселя, і в разі знаходження співпадіння у фрагмент QOI_OP_INDEX заноситься індекс пікселя з буфера (рисунок 2.4).

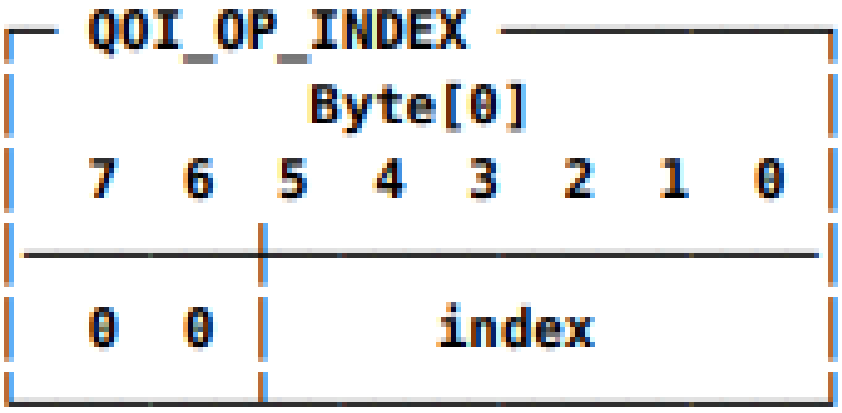


Рисунок 2.4 – Фрагмент QOI_OP_INDEX

Третій варіант. Оскільки у більшості випадків різниця між попереднім і поточним значенням пікселів невелика, то фрагмент дорівнюватиме різниці по модулю 255 (8-бітний канал) між поточним пікселем і попереднім (рисунок 2.5).

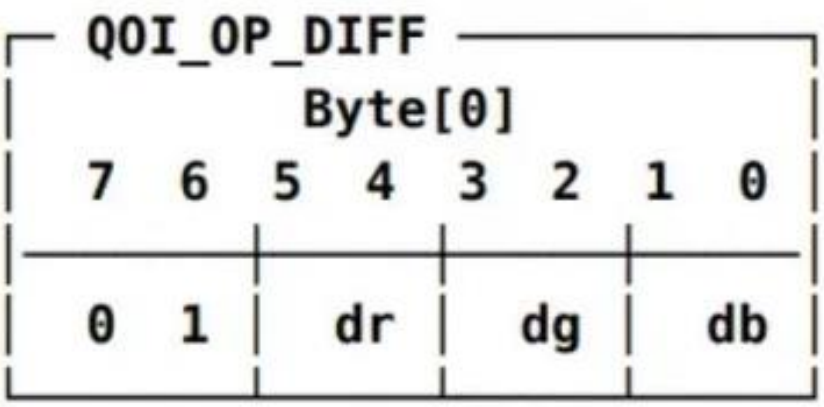


Рисунок 2.5 – Фрагмент QOI_OP_DIFF

Також можливим варіантом фрагменту є QOI_OP_LUMA зображено на рисунку 2.6.

Він представляє собою різницю між поточним пікселем і попереднім, в даному прикладі різниця по зеленому каналу використовується як базова різниця для інших кольорів. Цей метод, як було доведено автором алгоритму, збільшує ступінь стиснення в тестах бенчмарку [14].

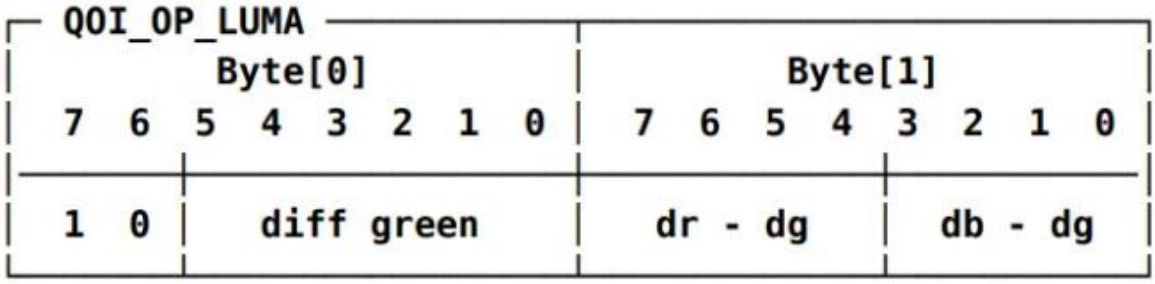


Рисунок 2.6 – Фрагмент QOI_OP_LUMA

Четвертий варіант. У випадку, коли жоден із попередніх варіантів не підходить, запам’ятовується останнє значення без змін та переходимо до наступного пікселя і виконуємо одну із чотирьох можливий дій.

Для запам’ятовування всієї інформації використовується фрагмент QOI_OP_RGB (див. рисунок 2.2) або QOI_OP_RGBA (див. рисунок 2.1).

2.2 Апаратна реалізація

Апаратна реалізація – це реалізація на фізичному рівні певної системи або пристрою. Апаратна частина може складається із пам’яті, мікросхеми, датчиків, провідники, процесори.

Апаратний прискорювач – це апаратний пристрій чи програмне забезпечення з основною функцією підвищення загальної продуктивності комп’ютера. Існують різні типи прискорювачів, які допомагають підвищити продуктивність різних аспектів функцій комп’ютера [15].

Прискорювач допомагає покращити якісні характеристики комп’ютера та його загальну продуктивність. Простим прикладом є відео-карта у комп’ютерах, що використовується для обробки графіки та виведення її, не завантажуючи центральний процесор.

Тобто апаратні прискорювачі це вузько-спеціалізовані схеми що мають свій процесор, пам’ять, канали вводу/виводу даних, і є незалежним від центрального процесора. Апаратне прискорення досить гарно підходить до будь-якого повторюваного, напруженого ключового алгоритму[16].

Апаратні прискорювачі можна розділити на наступні типи:

- За типом обчислень:
    - Прискорювачі із плаваючою комою – прискорюються обчислення з плаваючою комою , які часто використовуються в наукових та інженерних застосуваннях. Приклад : графічні процесори(GPU), тензорні процесори(TPU), фізичні процесори(PPU) тощо[16].
    - Прискорювачі цілих чисел – прискорюють обчислення з цілими числами, які часто використовуються в криптографії та компресії даних. Наприклад, копроцесори безпеки(SPU), копроцесори компресії(CPU) тощо [16].
    - Прискорювачі логічних операцій – прискорюють обчислення з логічними операціями, які часто використовуються в цифровій обробці сигналів та мікроконтролерах. Наприклад, програмовані

вентильні матриці(FPGA), програмовані логічні пристрої(PLD) тощо[16].

- За архітектурою:
    - Прискорювачі з різними інструкціями та різними даними (SIMD) – виконують одну і туж інструкцію на різних даних одночасно , використовуючи при цьому векторні або матричні регістри та операції. Наприклад, графічні процесори GPU, тензорні процесори(TPU), векторні процесори(VP) тощо[16].
    - Прискорювачі з різними інструкціями та однаковими даними(MIMD) – виконують різні інструкції на один і тих же даних одночасно, використовуючи багатоядерні або багато процесорні архітектури. Наприклад, симетричні багатопроцесорні системи (SMP), масиви паралельних процесорів(MPP), кластери комп’ютерів (CC) тощо[16].
    - Прискорювачі з однаковими інструкціями та однаковими даними(SISD) – виконують одну і ту ж інструкцію на одних і тих же даних послідовно, використовуючи скалярні або суперскалярні архітетури. Наприклад, центральні процесори (CPU), суперкомп’ютери (SC) тощо[16].

- За рівнем інтеграції:
    - Прискорювачі на рівні чіпа – інтегровані в той же чіп, що і основний процесор, що забезпечує високу швидкість та низьке споживання енергії [16]. Наприклад, копроцесри, які додаються до CPU, такі як MMX,SSE, AVX тощо.
    - Прискорювачі на рівні плати – підключені до основного процесора через шину або інтерфейс, що забезпечує більшу гнучкість та маштабованість[16]. Наприклад, розширювальні карти, які містять GPU, FPGA, PPU тощо.

- Прискорювачі на рівні системи – підключення до основного процесора через мережу або кластер, що забезпечує більшу паралельність та розподіленість. Наприклад віддалені сервери, які надають обчислювальні ресурси [16].

- За способом програмування:
  - Прискорювачі з фіксованою функцією – виконують попередньо визначену функцію, яка вже не може бути змінена користувачем. Наприклад, кодеки, шифратори, декодери тощо [16].
  - Прискорювачі з програмованою функцією, яка може бути змінена користувачем за допомогою спеціальної мови програмування або інтерфейсу. Наприклад, GPU, FPGA, TPU тощо [16].

2.3 Архітектура апаратних прискорювачів

2.3.1 Основні компоненти

Архітектура апаратних прискорювачів складається із наступних основних елементів, що працюючи у взаємодії для виконання спеціалізованих обчислень та завдань, з метою підвищення продуктивності комп’ютера в конкретних напрямках. Основні компоненти архітектури апаратних прискорювачів включають:

- Acceperator Proccesor (процесор апаратного прискорювача)
- Memory(пам’ять)
- Control Logic(керуюча логіка)
- I/O Interface(інтерфейс взаємодії із іншими система комп’ютера)
- Computation Controller (контролер обчислень)
- Data Bus, Control Bus (шина даних, шина керування)
- Vectorization Unit (модуль векторізації)
- Parallelism Unit (модуль паралельності)
- Power Management Module (модуль керування енергоспоживання)

Acceperator Proccesor (процесор апаратного прискорювача) – ядро, що спеціалізується на виконанні певного типу завдань або обчислень. Такі ядра виділяються високою продуктивністю у своєму напрямку, але будуть менш потужними у порівнянні із загально призначеними процесорами.

Memory(пам’ять) – зовнішня і внутрішня пам’ять, що необхідна для зберігання даних та інструкцій, які використовуються прискорювачам. Пам’ять характеризується високою швидкість доступу.

Control logic – логіка, що відповідає за керування потоком інструкцій, координацію роботи прискорювача та обробку введених /виведених даних.

I/O Interface – це сукупність засобів і правил, що забезпечують передачу, даних між прискорювачем і рештою системи.

Computation Controller(контролер обчислень) – це є компонентом апаратного прискорювача, який керує процесом обчислень. Він відповідає за розподіл завдань між різними обчислювальними блоками прискорювача, а також за координацію їх роботи. Контролер обчислень може включати в себе різні механізми, такі як планувальники завдань, контролери пам’яті, які керуються доступом до пам’яті прискорювача.

Data Bus, Control Bus (шина даних, шина керування) – слугують для передачі сигналів керування та передачі даних між різними компонентами прискорювача.

Vectorization Unit (модуль векторізації) – присутній, якщо апаратний прискорювач має здатність підтримувати векторизацію. Модуль відповідає за одночасну обробку декількох елементів даних, що сприяє паралельному виконанню операцій. Векторизація – це процес перетворення програми або алгоритму таким чином, щоб він міг використовувати векторні операції. У свою чергу векторні операції дозволяють обробляти одночасно декілька елементів даних

Parallelism Unit (модуль паралельності) – є компонентом апаратного прискорювача, який використовується для оптимізації обчислень шляхом

виконання декількох операцій одночасно. Паралельні обчислення ґрунтуються на особливості великих задач. Одну велику задачу розбивають на декілька менший які можна виконати незалежно один від одного.

Power Management Module (модуль керування енергоспоживання) – компонент апаратного прискорення, що забезпечує контроль над енергоспоживанням, та її оптимізації. Модуль ефективно розподіляє енергоспоживання наприклад в умовах обмеженого живлення, чи високих вимог енергоефективності.

2.3.2 Архітектура SIMD

Архітектура SIMD(одиночний потік команд, множинний потік даних) – процесор такої архітектури має матричну структуру, вузли якої включають в себе велику кількість порівняно простих високопродуктивних процесорних елементів, які можуть мати власну або спільну пам’ять даних. Всі процесорні елементи виконують одночасно одну і ту ж команду, але на різними операндами, що подаються в пам’яті мультизадачним потоком. Таким чином така архітектура значно збільшує швидкість роботи алгоритмів призначених для паралельної обробки даних. SIMD архітектура (рисунок 2.7) показує продуктивні результати на обробці графіки, зображень, 3D-графіки, відео. За своєю особливістю, ця архітектура під час роботи у пам’ять завантажує максимально більше потрібних даних для виконання над ними паралельних операцій. Враховуючи цю особливість зазначимо, що і споживання енергії теж збільшується. На жаль дана архітектура паралельність не завжди може підійти до всіх типів програм та алгоритмів [17].

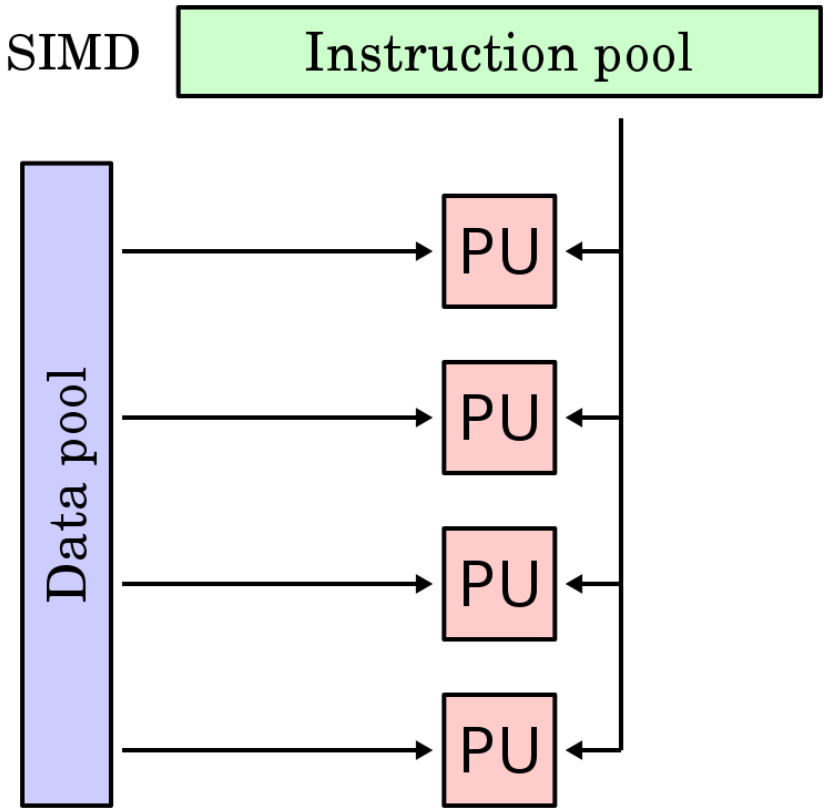


Рисунок 2.7 – Архітектура SIMD

Переваги архітектури SIMD:

- Швидко обробляє графічні дані(зображення, 3D-графіка, відео).
- Ефективний у застосуванні обробки великих масивів даних.
- Показує гарні результати на алгоритмах що призначені для паралельної обробки.

Недоліки архітектури SIMD:

- Значні вимоги у енергоспоживанні.
- Не всі алгоритми, підходять для даної архітектури, а саме ті що не призначені для паралельної обробки.

2.3.3 Архітектура MIMD

Архітектура MIMD (множинний потік команд, множинний потік даних) – є типом паралельної обчислювальної системи де кожен із обчислювальних пристроїв одночасно працюють над своїми даними та зі своїм набором команд [18].

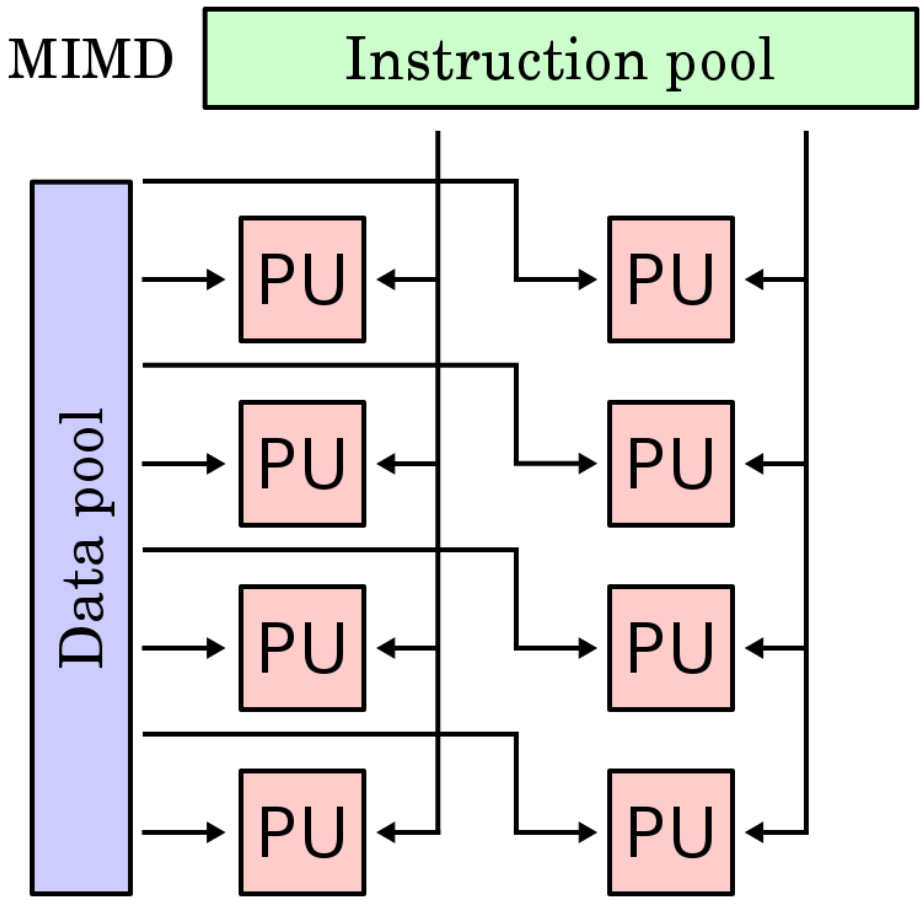


Рисунок 2.8 – Архітектура MIMD

На рисунку 2.8 зображено архітектуру MIMD.

Існують декілька реалізацій MIMD архітектури :

- Централізована архітектура
- Розподілена архітектура
- Спільна та розподілена пам’ять

Централізована архітектура – передбачає наявність одного головного процесора, що керує виконанням інструкцій та розподілом завдань між іншими

процесорами. Головний процесор координує дії інших процесорів та забезпечує їх синхронізацію[18].

Розподілена архітектура MIMD – включає набір незалежних процесорів, які працюють та виконують власні завдання. Кожен процесор може мати свою власну пам’ять та виконувати свою програму, в цій архітектурі немає головного процесора що контролює роботу інших[18].

Спільна пам’ять та розподілена пам’ять – архітектура може використовувати як спільну, так і розподілену пам’ять для обміну даних між процесорами. У спільній пам’яті всі процесори мають доступ до одного адресного простору, що спрощує обмін даними. У розподіленій пам’яті кожен процесор має свій власний набір пам’яті, і обмін даними відбувається через передачу повідомлень між процесорами[18].

Завдяки своїй особливості MIMD архітектура дозволяє виконувати безліч завдань паралельно, збільшує продуктивність системи та обчислення даних. Архітектура включає в себе різні рівні паралелізму від конвеєра до незалежних завдань і програм, то будь яка обчислювальна система цього класу в приватних додатках може виступати як SISD і SIMD-система[18].

MIMD архітектура дозволяє виконувати багато завдань паралельно, що пришвидшуватиме обчислення та збільшує продуктивність систем. Такі системи легко масштабуються, додаючи нові процесори або видаляти існуючі, при цьому вплив на продуктивність загальну не значний. Декілька процесорів які працюють з різними задачами одночасно, це забезпечує ефективність під час рішення задач, та забезпечує швидке виконання. Завдяки своїм властивостям дана архітектура часто використовується при рішенні наукових задач, де потрібно обчислювати великі масиви інформації. Варто зазначити що дану архітектуру успішно використовують у ігровій індустрії, криптографія, обробка зображень[18].

Переваги MIMD архітектури:

- Масштабованість
- Ефективність
- Гнучкість

- Висока продуктивність
- Паралельність

Серед недоліків можна виділити наступне:

- Не можливість виконання задач які не можливо розділити для паралельної обробки даних.

2.3.3 Архітектура SISD

Архітектура SISD (одна команда, одні дані) – архітектура із послідовним виконанням інструкцій. Це є звичні комп’ютери, які за один момент часу виконує одну дію(операцію) на одному ядрі [19].

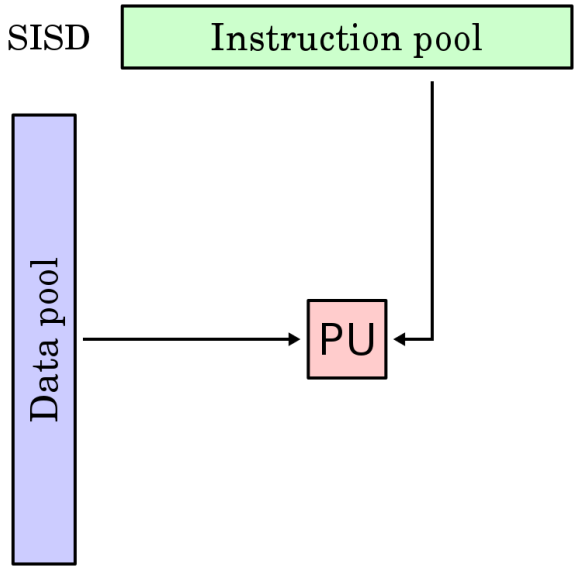


Рисунок 2.9 – Архітектура SISD

Така архітектура проста та не вимагає складних функціональних рішень, в ній зникає проблема із доступом до пам’яті чи набору операцій. Більшість поширених задач можна вирішити даною архітектурою навіть на нескладних система, адже вона не вимагає складної реалізації [19].

Отже можемо виділити наступні переваги :

- Простота реалізації
- Ефективність
- Сумісність

Серед недоліків можна виділити наступне:

- Не можливість виконання задач, які призначені для паралельної обробки даних [19].

2.4. Способи програмування прискорювачів

2.4.1 Прискорювачі із фіксованою функцією

Прискорювачі з фіксованою функцією – виконують попередньо визначену функцію, яка вже не може бути змінена користувачем. Такі прискорювачі вже із заводу виробника запрограмовані на виконання конкретних задач. Ці спеціалізовані прискорювачі вже оптимізовані під конкретні задачі. Яскравим приладом є шифратори та дешифратори кодеки.

2.4.2 Прискорювачі із програмованою функцією

Прискорювачі з програмованою функцією, це прискорювачі в яких програміст використовуючи додаткові програми та пристрої задають їх функціонал. За потреби програмісти може змінити призначення апаратного прискорювача, таким чином ми припускаємо що вони є більш універсальним в залежності від потреб. Прикладами є FPGA, TPU, GPU.

Програмовані прискорювачі типу FPGA (Field-Programmable Gate Array) –

є інтегрована схема, яку можна перепрограмувати для виконання певного завдання

2.5 Програмно-апаратна реалізація

Розглянемо теоретичну можливу реалізацію алгоритму QOI на апаратному рівні. Для реалізації не складна архітектура не потрібна, можна використати SISD, оскільки сам алгоритм передбачає тільки поступову обробку даних. Зумовлено, тим що обробка пікселів іде один за одним і кожен піксель при кодуванні залежить від уже пройдених пікселів перед цим. Аналогічно і при декодуванні, всі дії виконуються послідовно.

Елементи які потрібно врахувати:

- вхідний потік даних,
- модуль оперативна пам’ять,
- модуль фізична пам’ять,
- модуль обробки даних,
- вихідний потік даних.

Можемо теоретично розглянути реалізації алгоритму.

Якщо ми маємо дуже обмежені апаратно-фізичні властивості модулів чи всієї системи. У такому випадку можемо зробити, щоб на модуль обробки передавався тільки один піксель із фізичної пам’яті, і після обробки ми отримаємо на виході вже закодований фрагмент і пишемо її у виділену область у фізичній пам’яті.

2.5.1 Модуль обробки даних

Для обробки поточного пікселя виконуються наступні дії:

1. Порівняння пікселя із попереднім.
2. Порівняння пікселя із уже пройденими у буфері.
3. Виконання різниці
4. Зберігання даних

У зв’язку із тим що дії виконуються над одним поточним пікселем, отже, великого об’єму оперативної пам’яті не потрібно. Слід врахувати, що буфер пройдених пікселів, додаткові змінні такі як кількість повторів пікселів, значень різниці пікселів. Потрібно тримати в оперативній пам’яті. Буфер має розмір 64 пройдених пікселі, тобто від 192 до 256 байтів. Інформація про поточний і пройдений піксель від 6 до 8 байтів. Слід врахувати також інформацію про всі можливі варіанти фрагментів кодування в сумі виходить 12 байтів. Додаємо, ще мінімум 5 байтів для обчислення різниця колірних моделей у пікселів. Виходячи із вище наведених даних бачимо, що мінімальний об’єм це 300 байтів, але важливо розуміти, що в реальності буде потрібно більший об’єм на додаткові ресурси для обчислення та оптимальної роботи системи.

Ключовий момент, що на виході повинен бути як готовий результат вже повністю сформований фрагмент.

На модуль обробки буде передаватися тільки поточний піксель і виконуватися операції над ним.

2.5.2 Модуль фізична пам’ять

У модулі фізичної пам’яті можемо зберігати вхідне початкове зображення, на модуль обробки пам’яті, відсилатимемо кожен піксель послідовно, також у фізичній пам’яті потрібно буде виділити такий же розмір як і вхідне зображення для зберігання вже стиснутого зображення. Виділяємо пам’ять більше, щоб забезпечити об’єм для зберігання, якщо пам’яті буде забагато виділеної, різницю можна буде очистити. Модуль обробки буде так само послідовно віддавати вже готові стиснуті фрагменти і записуватися у модуль фізичної пам’яті.

Отже можемо зобразити схематично який може мати вигляд кодер (рисунок 2.10) та декодер для даного алгоритму.

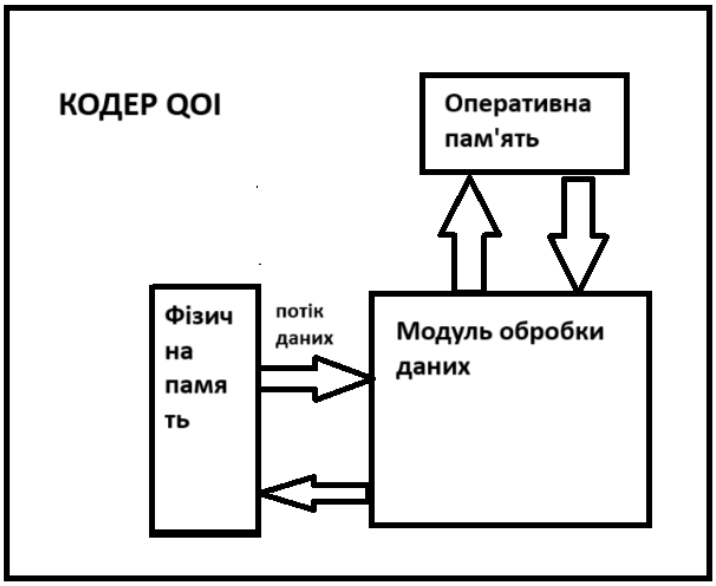


Рисунок 2.10 – Кодер QOI

Декодер (рисунок 2.11) матиме такий же вигляд, різниця буде тільки в програмуванні модуля обробки даних.

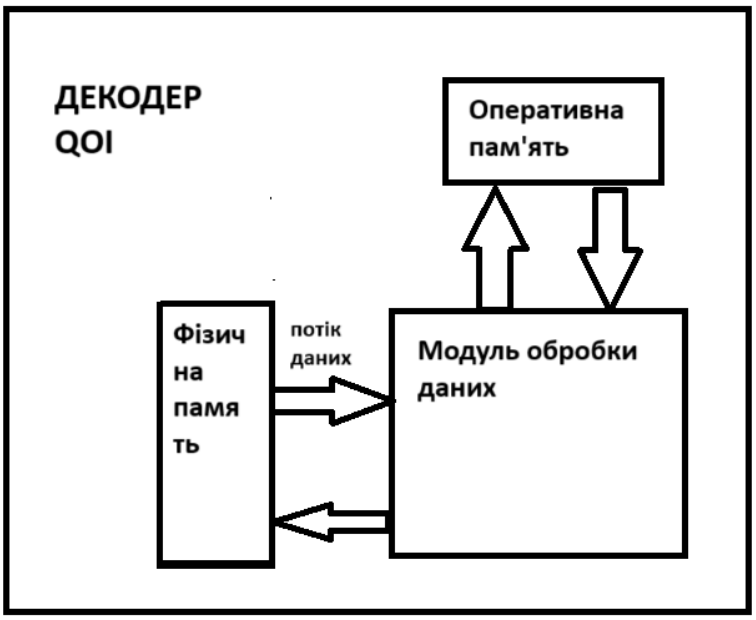


Рисунок 2.11 – Декодер QOI

2.6. Аналіз алгоритму QOI для виявлення можливостей удосконалення

2.6.1 Модифікацій алгоритмів

Алгоритми стиснення як і всі алгоритми мають свої переваги та недоліки. Створити універсальний алгоритм, котрий буде показувати на будь яких даних найкращі результати неможливо. Один алгоритм швидко стискає дані, але має низький ступінь стиснення. Інший алгоритм максимально можливо стисне дані, та програє по часу кодування. Досить довге виконання може бути спричинене алгоритмом на який виконує обробку даних він основі є надто складним тощо. Якість стиснення та швидкість залежить і від вхідних даних, всі дані різні і для різних типів даних є свої найкращі формати.

Будь який алгоритм, що винайдені завжди модифікуються та удосконалюються, це може робити як і сам розробник, так і інші бажаючі, що виявили проблему і вирішили його вдосконалити, або пристосувати до конкретних умов використання. Кожне удосконалення спрямоване на покращення характеристик алгоритму, пришвидшення, збільшити якісні показники, наприклад ступінь стиснення, тощо.

Для початку зміни оригінального алгоритму.

Перед тим, як модифікувати алгоритм слід виконати наступні дії:

- Провести аналіз роботи кожного кроку алгоритму.
- Розглянути можливість залежність даних, які використовує алгоритм для своєї роботи, та проаналізувати їх залежність між собою.
- Розглянути можливість модифікації кожного елементу алгоритму окремо.

Також, для модифікації алгоритму або покращення його роботи, можна використати додаткові алгоритми, наприклад, попередню обробку вхідних даних, далі використати алгоритм, що модифікується (оригінальний або модифікований). Можна застосувати комплексний підхід до модифікації, із використанням декількох алгоритмів та модифікацію окремих елементів роботи алгоритму.

2.6.2 Аналіз алгоритму QOI

Для початку модифікації коду алгоритму визначимо ключові елементи алгоритму(буфер або те як кодуються символи тощо) та проведемо аналіз можливості модифікації кожного елементу окремо чи разом.

Для алгоритму стиснення QOI можемо виділити такі елементи [20]:

- Буфер із попередніми значеннями 64 пройдених пікселів.
- Чотири методи шифрування пікселі:
    - Записати оригінальні значення колірної моделі поточного пікселя. QOI_OP_RGB або QOI_OP_RGBA
    - Запам’ятовування якщо піксель теперішній рівний попередньому(QOI_OP_RUN)
    - Запам’ятати різницю значень колірної моделі із по попереднім пікселем(QOI_OP_DIFF або QOI_OP_LUMA)
    - Записати оригінальні значення колірної моделі поточного пікселя. QOI_OP_RGB або QOI_OP_RGBA.

- Віднайти індекс такого ж пікселя у буфері пройдених пікселів(QOI_OP_INDEX)
- Записати оригінальні значення колірної моделі поточного пікселя. QOI_OP_RGB або QOI_OP_RGBA.

Кожен із методів шифрування записуються у фрагменти. Розмір одного фрагменту залежить від його типу (таблиця 4).

Таблиця 4 – таблиця розміру кожного із фрагментів, які містить алгоритм QOI.

Назва	Довжина
QOI_OP_RUN	1 байт
QOI_OP_DIFF	1 байт
QOI_OP_LUMA	2 байти
QOI_OP_INDEX	1 байт
QOI_OP_RGBA	5 байтів
QOI_OP_RGB	4 байти

Аналіз фрагментів QOI_OP_RGBA та QOI_OP_RGB. Фрагменти QOI_OP_RGBA та QOI_OP_RGB мають схожу структуру, кожен фрагмент запам’ятовує повну інформацію про значення кольорів r, g, b та наявність альфа-каналу. В роботі алгоритму ці фрагменти є основними, тому що подальші фрагменти залежать від них. Під час роботи алгоритму, особливо на початку коли ми перевіряємо перший піксель зображення та завжди запам’ятовується повна його інформація та наступний спосіб кодування ґрунтується на порівнянні із попереднім. Також варто зазначити що не завжди всі пікселі можна закодувати іншими варіантами, тому мати повну інформацію про піксель без змін це важливо, тому ці фрагменти ми не будемо модифікувати [20].

Аналіз фрагментів QOI_OP_RUN. Фрагмент QOI_OP_RUN слугує для запам’ятовування кількості повторів однакових пікселів підряд. Старші біти мають дві одиниці, які є ідентифікатором фрагменту, на запам’ятовування

відводиться інші шість бітів. Згідно документації є обмеження в максимальному числі, 62 – значення. Тобто 62 повтори пікселів максимально, якщо буде більша кількість кодування буде відбуватися іншим варіантом. Сучасні зображення маю розміри по декілька тисяч пікселів або десятки тисяч, тому ми розглянемо можливість зміни даного фрагменту для запам’ятовування більшої кількості повторюваних пікселів, чим і зможемо менше використати пам’яті [20].

Аналіз фрагментів QOI_OP_DIFF та QOI_OP_LUMA. Фрагменти мають розмір у 8 бітів та 16 бітів відповідно, кожному випадку старші два біти це ідентифікатори, інші біти для запам’ятовування корисної інформації. Ці фрагменти запам’ятовують різницю значень кожного кольору між поточний і попереднім. За допомогою фрагменту QOI_OP_DIFF можна закодувати піксель коли різниця значень коливається в межа від -2 до 1. А за допомогою фрагменту QOI_OP_LUMA закодувати піксель якщо різниця значень для зеленого кольору коливається від -32 до 31, різниця значень червоний-зелений, та синій-зелений від -8 до 7. Враховуючи, що різниця між суміжними пікселями завжди не велика і ці фрагменти достатньо ефективно перекодуватиме пікселі. Модифікація даних фрагментів не вважаємо доцільно [20].

Аналіз фрагменту QOI_OP_INDEX. Фрагмент QOI_OP_INDEX, у старші два біти – ідентифікатори, і шість бітів на зберігання індексу по якому даний піксель знаходить у буфері пройдених пікселів. Буфер пройдених пікселів тримає в собі інформацію про шістдесят чотири пройдені пікселі. Записуються згідно функції, параметрами якої є значення колірної моделі пікселя. Одна якщо ми збільшимо буфер пройдених пікселів, ми матимемо більше даних на які зможемо посилатися та менше буде використовуватися типи фрагментів QOI_OP_RGB, QOI_OP_RGBA [20].

Обмеження для модифікації алгоритму. Ідентифікація фрагментів QOI_OP_RGBA та QOI_OP_RGB особлива, їх можна ідентифікувати коли значення байту, рівна 11111111 або 11111110 відповідно. Інші фрагменти ідентифікуються тільки по 2 старшим бітам цього достатньо, тому що всього

лишилось 4 види фрагменту, відповідно множини варіацій значень двох бітів цілком достатньо [20].

Зауважимо, що у наслідок особливості ідентифікації фрагментів QOI_OP_RGBA та QOI_OP_RGB, для фрагменту QOI_OP_RUN є обмеження. Ці обмеження пов’язані із тим, що для ідентифікації фрагменту QOI_OP_RUN, у двох старших бітах встановлюється по 1, решта 6 бітів використовується на запам’ятовування кількості повторів. І якщо у нас повторів буде 62 або 63, у бітовому передавлені це 1111110 або 111111 відповідно, до цих значень потрібно ще додати по старші біти які рівні 11, і результатом матимемо кодування QOI_OP_RGB та QOI_OP_RGBA. Під час модифікації потрібно буде врахувати цю особливість для щоб уникнути помилок при декодуванні. Додаймо, що така особливість обмежує кількість пройдених однакових пікселів і у окремих випадках може негативно впливати на розмір кодування зображення [20].

2.6.3 Модифікація фрагменту QOI_OP_INDEX

Візьмемо до уваги QOI_OP_INDEX це фрагмент довжина якого вісім бітів, два з яких відносять до заголовку і шість бітів на запам’ятовування, що можу вмістити в собі 64 варіанти. Формула запам’ятовування для запису в буфер на конкретну позицію(1):

$$\text{index} = (r * 3 + g * 5 + b * 7 + a * 11)\%64 \qquad (1)$$

Вище ми вже ознайомились із з ключовими елементами алгоритму та розглянули що зміна довжини буферу пройдених пікселів може дати позитивний результат в кінцевих розмірах стиснутого зображення. Розглянемо одну із ситуацій, коли у нас після проходження 80 різних між собою пікселів, ми знаходимо піксель, який був 14 пікселем у пройдених 80 пікселях. Цей піксель не підходить до жодного із кодування окрім варіанту запису повної інформації про нього, і на що піде від чотирьох до п’яти байтів. У такій ситуації було зручно мати більший буфер пройдених пікселів, це виграє нам від трьох до чотирьох байтів. Враховуючи, що сучасні зображення мають досить великі розміри, до

прикладу зображення розміром 2560 на 1280 пікселів містить в собі 3 276 800 пікселів. Для такого розміру зображення буфер у шістдесят чотири пікселі є недостатнім. Ситуація коли ми будемо записувати повну інформацію про піксель досить часто буде зустрічатися під час обробки трьох мільйонів пікселів. І тому три або чотири байти різниці у кодуванні фрагментів можуть зіграти свою роль, вплинувши позитивно на кінцевий розмір вже стиснутого зображення [20].

Для того, щоб збільшити розмір буферу, потрібно і збільшити розмір фрагменту QOI_OP_INDEX(див Рис. 4). Збільшувати будемо на один байт, тобто фрагмент матиме вже два байти. При збільшенні розміру фрагменту ми повинні врахувати, що зменшиться і можливий виграш у розмірі стиснутому зображенні з трьох-чотирьох до двох-трьох байтів. Кількість виграшних байтів також залежить і від типу зображення, а саме чи міститься альфа канал чи ні. Після збільшення фрагменту на один байт, на запам’ятовування індексу відводитиметься чотирнадцять бітів і два біти на ідентифікатор. Вирахуємо максимальну можливе значення індексу, це 2^{14} і дорівнює 16384. Тоді як для попереднього варіанту максимальна кількість символів була 64 [20].

Після зміни довжини буферу, нам потрібно змінити хеш-функцію яка записує індекс, яка б в залежності від значень кольорів зеленого, червоного та синього видавала лише одне відповідне значення індексу у буфері.

Ми прагнемо модифікувати функцію таким чином, щоб вона оперувала виключно цілими числами для коефіцієнтів “r”, “g”, “b” та “a”. Це вимагає від нас визначити такі значення цих коефіцієнтів, які не тільки є цілими числами, але й, якщо можливо, є найменшими спільними множниками (НСМ) для чисел 3, 5, 7 і 11. Ми виявили, що НСМ для цих чисел становить 1155. Тому ми вирішили встановити коефіцієнти на рівні 1155, 2310, 8085, і 12615 відповідно [20]. Отримана формула буде мати наступний вигляд(2):

$$\text{index} = (r * 1155 + g * 2310 + b * 8085 + a * 12615)\%16384 \quad (2)$$

Вигляд даного фрагменту зображено на рисунку 2.12.

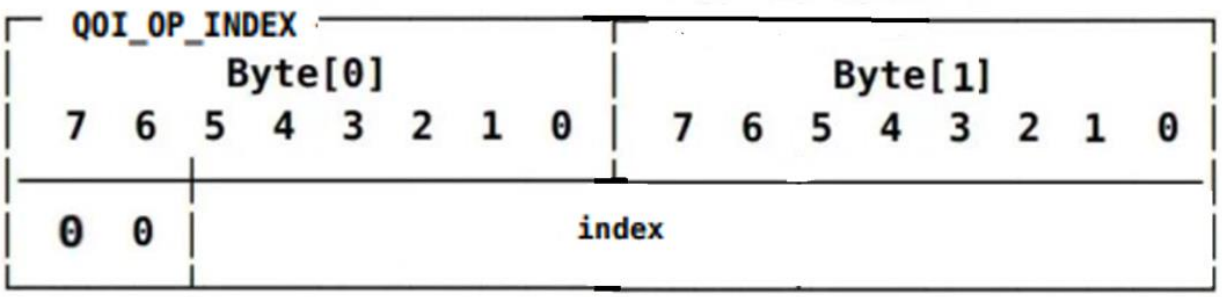


Рисунок 2.12 – Фрагмент QOI_OP_INDEX

2.6.4 Модифікація фрагменту QOI_OP_RUN

Розглянемо фрагмент QOI_OP_RUN, що записує кількість повторів попереднього пікселя. Отже, якщо у нас виникне ситуація коли буде однакових пікселів по підряд більше ніж 62. Тоді для наступного пікселя потрібно буде використати один із 3 інших варіантів. Далі ми знову кодуємо наступний піксель за допомогою QOI_OP_RUN. Внаслідок ми матимемо 3 байти інформації. І кожні наступні 62 однакових пікселів, даний варіант буде добавляти 2 байти. Так, як більшість зображень мають по декілька сотень тисяч пікселів, тому така ситуація буде повторюватися досить часто. Як і в попередній модифікація ми тут збільшимо довжину фрагмента на 8 бітів тобто 1 байт. Такими діями ми зможемо закодувати до 15872 однакових пікселів, враховуючи обмеження, що у 16 бітовому числі, у восьми старший бітах не можна використовувати комбінації що рівні 11111111 та 11111110 відповідно, оскільки, тоді декодер не ідентифікує даний фрагмент як QOI_OP_RUN, а як фрагмент QOI_OP_RGB чи QOI_OP_RGBA. і ми втратимо оригінальне зображення. Така модифікація дає нам виграш у декілька десятків можливо і сотні байт [20].

Новий модифікований фрагмент зображено на рисунку 2.13.

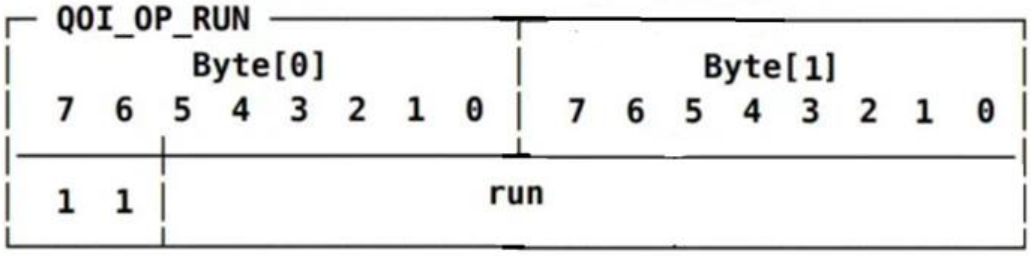


Рисунок 2.13 – Фрагмент QOI_OP_RUN

Висновки до розділу 2

У розділі було розглянуто детально роботу алгоритму QOI. Він виконує кодування та декодування зображення за один прохід по пікселям. Він не вимогливий до програмного забезпечення, завдяки чому він може реалізований на апаратному рівні. Під час ознайомлення із апаратними реалізаціями було зображено потенційно можливу реалізацію кодера та декодера.

Під час аналізу алгоритму ми виявили області для поліпшення, а саме збільшення розміру буферу пройдених пікселів, що призведе до змінення фрагменту QOI_OP_INDEX, а також можливість розширення фрагменту QOI_OP_RUN. Ці модифікації не є критичним адже вони і так повноцінно функціонують, однак мають потенціал для їх подальшого вдосконалення та покращення.

Модифікація із збільшеним буфером дає нам можливість зменшити використання фрагментів QOI_OP_RGB та QOI_OP_RGBA, що на кожному використанні модифікованому фрагменту QOI_OP_INDEX, на два або три байти ніж при записі повної інформації.

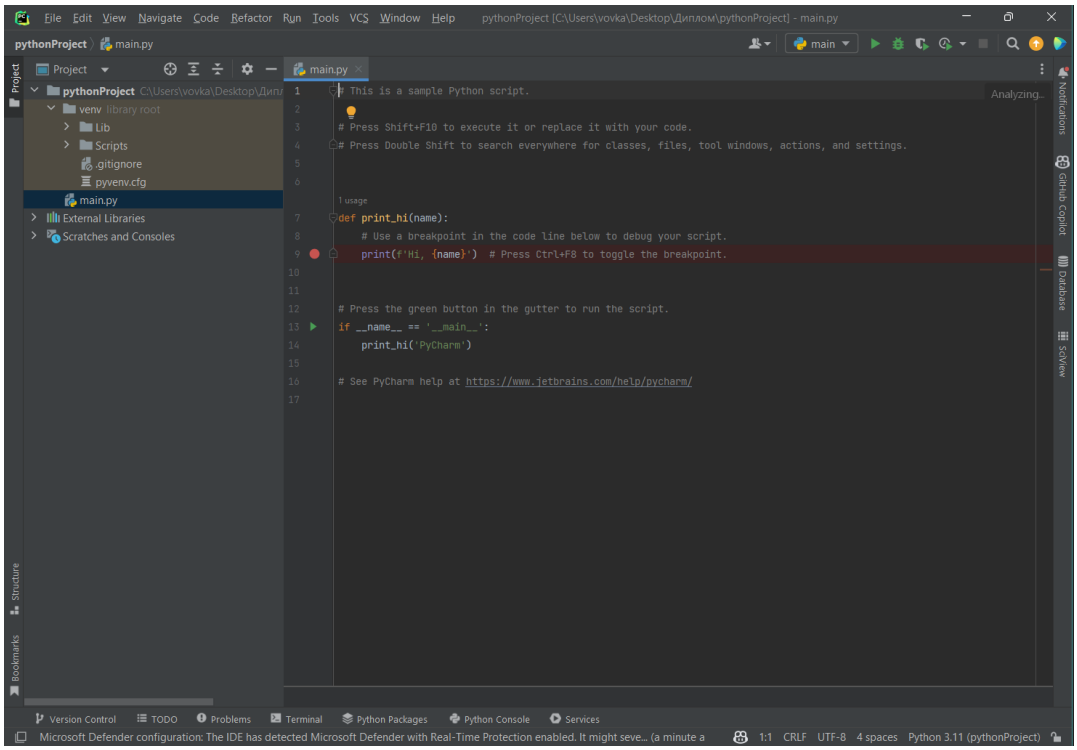
Модифікація фрагменту QOI_OP_RUN із розширеним діапазоном значень для запам’ятовування потворів пікселів. Це дозволяє за допомогою одного фрагменту запам’ятати більшу кількість повторів. Така модифікація досить ефективно може себе показати для зображень з великими однорідними областями.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ, ОПИС ПРОГРАМНОГО СЕРЕДОВИЩА

3.1 Програмне середовище для розробки PyCharm

PyCharm – є потужним інструментом для розробки на мові Python. Це продукт який створила компанія JetBrains, який є підтримується на багатьох платформах, що робить його вибором для багатьох розробників.

Інтерфейс програмного середовище наведено на рисунку 3.1



Рисунку 3.1 – Інтерфейс середовища розробки PyCharm

Інтерфейс PyCharm складається із наступних основних частин :

1. **Меню**: Містить команди для виконання різних операцій, таких як створення проекту, запуск програми, збереження файлів налаштування рисунок 3.2)

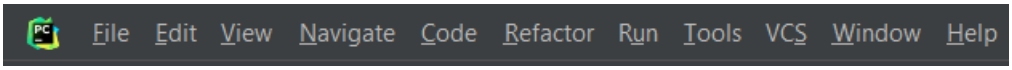


Рисунок 3.2 – Меню панель середовища розробки PyCharm

2. **Панель інструментів**: Містить кнопки для швидкого доступу до найбільш часто використовуваних команд.

Рисунок 3.3 – панель інструментів середовища розробки PyCharm

3. **Дерево проектів**: Показує структуру вашого проекту, включаючи всі файли та папки(рисунок 3.4).

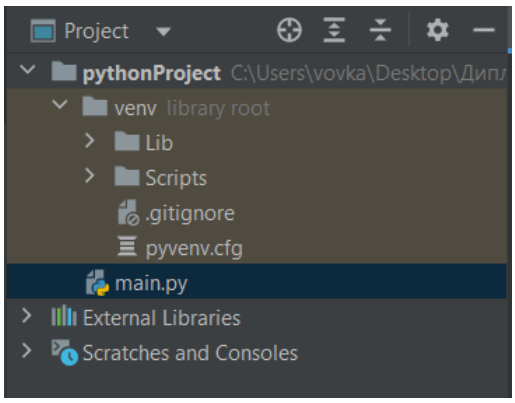


Рисунок 3.4 – Дерево проектів.

4. **Редактор коду**: Це місце, де ви пишете та редагуєте свій код. Він має багато корисних функцій, таких як підсвічування синтаксису, автозавершення коду та інше(рисунок 3.5).

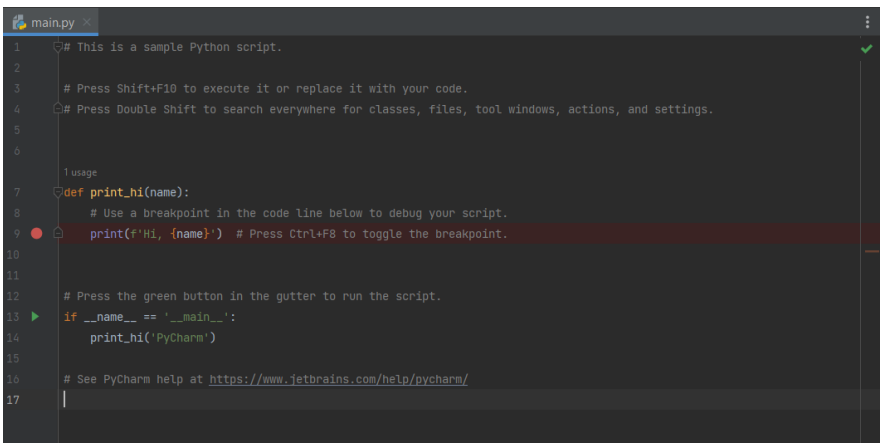


Рисунок 3.5 – Редактор коду.

5. **Консоль**: Відображає вивід вашої програми під час виконання(рисунок 3.6). Тут ви також можете вводити команди в режимі інтерактивного виконання. Поруч можна також переглянути вкладку контролю версій.

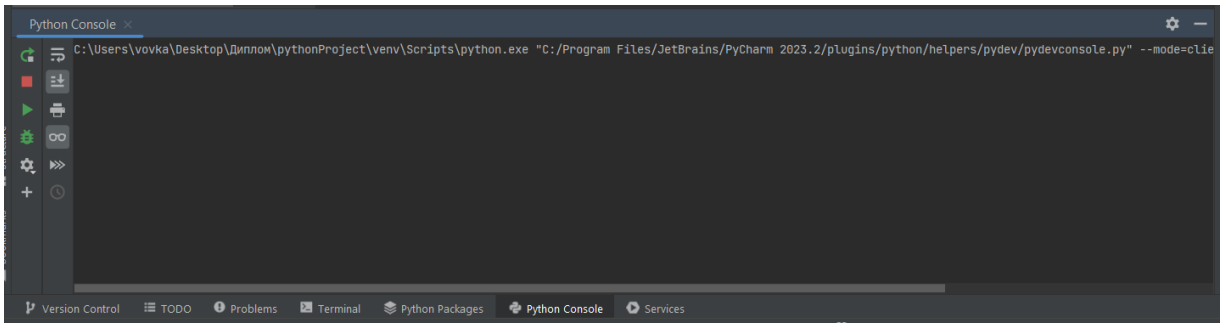


Рисунок. 3.6 – Консоль

Ці частини можна налаштувати відповідно до ваших потреб, і ви можете додавати або видаляти панелі інструментів відповідно до ваших потреб.

Автодоповнення та синтаксичне виділення підтримуються редактором коду, що робить написання коду простим і швидким. Для ефективного виявлення та виправлення помилок відлагодження PyCharm включає встановлення точок зупинки та перегляд значень змінних рис 3.7.

Дерево проектів PyCharm полегшує управління проектами, надаючи ієрархічний перегляд файлів і тек. Крім того, кодова база легко редагується за допомогою вбудованих інструментів пошуку та заміни. PyCharm підтримує віртуальні середовища Python, що робить управління залежностями простішим і дозволяє відокремлювати проекти один від одного. Інтеграція з системами контролю версій, такими як Git, дозволяє використовувати простий інтерфейс для комітів і гілок, що полегшує роботу з кодом у команді. PyCharm має підтримку плагінів, що дозволяє IDE розширювати свої функції відповідно до потреб користувача. Також PyCharm містить інструменти для веб-розробки, такі як підтримка мов JavaScript, HTML і CSS і робота з різними фреймворками. Що стосується тестування, PyCharm пропонує інтегровані інструменти для ефективного написання, запуску та аналізу тестів, що допомагає розробникам

розробляти програми, які є надійними та стійкими. Одним із важливих можливостей PyCharm є просте перемикання та налаштування між різними версіями Python, що робить його гнучким інструментом для роботи над проектами, які мають різні вимоги до версій мови програмування.

3.2 Реалізація оригінального алгоритму на мові Python

3.2.1 Реалізація стиснення

Алгоритму було реалізовано на мові Python. Для читання зображення по пікселях було використано бібліотеку Pillow.

Бібліотека Pillow – це яка створена для обробки графіки. Вона дозволяє створювати, обробляти, аналізувати зображення різних форматів.

Створено модель що характеризує піксель для запам’ятовування його значень та має один метод для порівняння

```
   class Pixel:
 def __init__(self, r=0, g=0, b=0, a=255):
    self.r = r
    self.g = g
    self.b = b
    self.a = a

  def __eq__(self, other):
      if not isinstance(other, Pixel):
          return NotImplemented

      return self.r == other.r and self.g == other.g and \
          self.b == other.b and self.a == other.a
```

Для зчитування та запису чотирьох байтів, було написано функції, write_32_bits(value, bytes_array) та read_32_bits(bytes_array).

```
def write_32_bits(value, bytes_array):
    bytes_array.append((0xff000000 & value) >> 24)
    bytes_array.append((0x00ff0000 & value) >> 16)
    bytes_array.append((0x0000ff00 & value) >> 8)
    bytes_array.append((0x000000ff & value))

def read_32_bits(bytes_array):
    b1, b2, b3, b4 = bytes_array
    return b1 << 24 | b2 << 16 | b3 << 8 | b4
```

Ці дві функції забезпечують запис та читання перших чотирьох бітів, коли ми записуємо магічні значення до заголовку, чи іншу інформацію.

Окрема функція hash_f(px), що повертає значення індексу у буфері для даного пікселя.

Функція create_image_from_pixels(decoded, size, mode) – яка зберігає стиснуте зображення після обробки.

```
def create_image_from_pixels(decoded, size, mode):
    img_qoi = Image.new(mode, size)

    if mode == 'RGBA':
        for i in range(len(decoded) // 4):
            x = i % size[0]
            y = i // size[0]
            r, g, b, a = decoded[i*4:(i+1)*4]
            img_qoi.putpixel((x, y), (r, g, b, a))
    else:
        for i in range(len(decoded) // 3):
            x = i % size[0]
            y = i // size[0]
            r, g, b = decoded[i*3:(i+1)*3]
            img_qoi.putpixel((x, y), (r, g, b))

    return img_qoi
```

Основні дві функції, це кодування та декодування, відповідно encode(pixels) та decode(pixels). Аргументи цієї функції це масив пікселів зображення, у випадку кодування – це масив не стиснутого зображення, випадку декодування, це масив стиснутого зображення.

Функція декодування можна розділити на декілька частин, перша це оголошення змінних потрібних необхідних для роботи алгоритму, наприклад run – змінна, що запам’ятовує кількість повторів однакових пікселів підряд, далі це значення буде записано у бітовому представленні у фрагмент QOI_OP_RUN. Також оголошення буферу пройдених пікселів.

Далі формування заголовку, основна частина це цикл який проходиться по всіх пікселях зображення і в якому іде аналіз на спосіб кодування пікселя, та цикл що формує заключний маркер.

Нижче наведено код де ми зберігаємо інформацію про розміри висота та ширина, та підраховуємо загальну кількість пікселів у зображенні. Оголошено

додаткові змінні та буфер пройдених пікселів. Створюємо також бітовий масив, в якому будемо зберігати вже закодоване зображення.

```
h = pixels.size[1]
w = pixels.size[0]

total_size = w * h
run = 0
index = [Pixel() for _ in range(64)]
bytes_array = bytearray()
```

Використовуючи функції запису write_32_bits, ми записуємо починаємо записувати у масив байтів заголовок, це магічне число ідентифікатор, що сигналізуватиме яким алгоритмом закодоване зображення, та інформацію про висоту та ширину і пікселях. Фрагмент коду представлений нижче.

```
write_32_bits(QOI_MAGIC, bytes_array)
write_32_bits(w, bytes_array)
write_32_bits(h, bytes_array)
```

Наступний крок це перевірити наявність альфа каналу, та записати інформацію до заголовку.

```
channels = 4 if pixels.mode == 'RGBA' else 3

bytes_array.append(4) if channels == 4 else bytes_array.append(3)
```

Після створення заголовку переходимо до обробки пікселів зображення. Зчитуємо послідовно пікселі, перевіряємо присутність альфа каналу для того щоб визначити який тип фрагменту використовувати якщо ми будемо записувати повну інформацію про піксель.

```
for h_i in range(h):
    for w_i in range(w):
        pixel = pixels.getpixel((w_i,h_i))
        px.r = pixel[0]
        px.g = pixel[1]
        px.b = pixel[2]

        if channels == 4: px.a = pixel[3]
```

Перша перевірка полягає в тому, чи є поточний піксель рівним попередньому. Якщо умова виконується, змінна `run` збільшується на одиницю. Потім перевіряється, чи є поточний піксель останнім, а також чи досягнуто максимального значення повторів згідно з алгоритмом. Якщо виконується хоча

б одна з цих умов, ми кодуємо змінну `run` у фрагмент `QOI_OP_RUN` та зберігаємо його у байтовий масив результату. У випадку коли поточний піксель не є рівним попередньому а значення змінної run не дорівнює нулю, ми зберігаємо значення run у фрагмент QOI_OP_RUN, записуємо фрагмент у байтовий масив результату, змінюємо значення змінної run на нуль, і переходимо до наступної перевірки поточного пікселя.

```
if px == px_prev:
    run += 1
     if run == 62 or w_i * h_i >= total_size:
         bytes_array.append(QOI_OP_RUN | (run - 1))
         run = 0
else:
     if run > 0:
         bytes_array.append(QOI_OP_RUN | (run - 1))
         run = 0
```

Наступний крок це визначити індекс, який піксель займав би у буфері пройдених пікселів. Визначивши індекс, ми порівнюємо між собою піксель який записаний у буфері пройдених пікселів за цією позицією і поточний піксель. Якщо у нас умова виконується і пікселі рівні ми зберігаємо індекс у фрагмент QOI_OP_INDEX та переходимо до наступного пікселя. Якщо пікселі не рівні, записуємо за знайденим індексом поточний піксель у буфер пройдених, та перевіряємо на інші можливі варіанти кодування що лишились.

```
index_pos = hash_f(px) % 64

if (index[index_pos] == px):
    bytes_array.append(QOI_OP_INDEX | index_pos)
else:
    index[index_pos] = cp.copy(px)
```

Далі ми перевіряємо чи альфа канал поточного пікселя рівний попередньому, якщо що зображення не має альфа-каналу то ці значення рівні 0, і умова виконується. У разі рівності альфа-каналів, ми знаходимо різницю значень кожного кольору, тобто різницю значень зеленого кольору поточного пікселя із попереднім, та аналогічно і різницю в червоному та синьому кольорі. Також знаходимо значення різниці значень, а саме різниця червоних кольорів із різницею зеленого, та різницю значень між різницею синіх та різницею зеленого.

```
if px.a == px_prev.a:
    vr = px.r - px_prev.r
```

```
    vg = px.g - px_prev.g
    vb = px.b - px_prev.b
    vg_r = vr - vg
    vg_b = vb - vg
```

Оскільки ми маємо два фрагменти які кодують різницю значень колірної палітри, ми перевіряємо яка із них краще підходить. Якщо різниця значень кожного кольору лежить в межах -2 до 1 включно, тоді ми використовуємо фрагмент QOI_OP_DIFF. Якщо значення лежать в більшому діапазоні, тоді ми перевіряємо чи лежить значення різниць синій-зелений та різниці червоний-зелений лежить в діапазоні від -8 до 7 включно, і різниця зеленого у двох пікселях є в межах -32 до 31 включно. У такому випадку ми зберігаємо ці різниці у фрагментів QOI_OP_LUMA.

```
if vr > -3 and vr < 2 and vg > -3 and vg < 2 and vb > -3 and vb < 2:
    bytes_array.append(QOI_OP_DIFF | (vr + 2) << 4 | (vg + 2) << 2 | (vb + 2))
elif vg_r > -9 and vg_r < 8 and vg > -33 and vg < 32 and vg_b > -9 and vg_b <
8:
    bytes_array.append(QOI_OP_LUMA | (vg + 32))
    bytes_array.append((vg_r + 8) << 4 | (vg_b + 8))
```

Якщо жодний із варіантів не підійшов ми запам’ятовуємо повну інформацію про піксель використовуючи фрагмент QOI_OP_RGBA або QOI_OP_RGB.

```
    else:
        bytes_array.append(QOI_OP_RGB)
        bytes_array.append(px.r)
        bytes_array.append(px.g)
        bytes_array.append(px.b)
else:
    bytes_array.append(QOI_OP_RGBA)
    bytes_array.append(px.r)
    bytes_array.append(px.g)
    bytes_array.append(px.b)
    bytes_array.append(px.a)
```

Після проходження по всіх пікселях ми записуємо заключний маркер із семи байтів із нульовим значення та останній байт з одиницею та повертаємо результат.

```
for                    i                    in                    range(7):
    bytes_array.append(0)
bytes_array.append(1)

return bytes_array
```

3.2.2 Реалізація декодера

У реалізації декодера використовуємо функції читання та запису, клас pixel. Спершу зчитуємо заголовок зображення, його розмір, наявність альфа каналу. Створюємо буфер пройдених пікселів

Також ми визначаємо константи із якими ми будемо порівнювати байти

```
run = 0
index = [Pixel() for _ in range(64)]

qoi_magic = read_32_bits(bytes_array[:4])
del bytes_array[:4]
w = read_32_bits(bytes_array[:4])
del bytes_array[:4]
h = read_32_bits(bytes_array[:4])
del bytes_array[:4]
channels = bytes_array.pop(0)
```

для ідентифікації типів фрагментів

```
QOI_OP_INDEX = 0x00
QOI_OP_DIFF = 0x40
QOI_OP_LUMA = 0x80
QOI_OP_RUN = 0xc0
QOI_OP_RGB = 0xfe
QOI_OP_RGBA = 0xff
QOI_MASK_2 = 0xc0
```

Після читання заголовку переходимо до зчитування байтів зображення та ідентифікацією типом фрагментів.

```
if run > 0:
    run -= 1
elif len(bytes_array) > 8:
    b1 = bytes_array.popleft()
```

Перевіряємо на початку змінну run, для можливого розковування повторюваних пікселів, за замовчуванням її значення нуль. Після перевіряємо довжину кількості байтів яку потрібно розкодувати, якщо довжина менше 8 тоді ми дійшли кінця зображення і зустріли заключний маркер і кодування можна завершити, в іншому випадку ми продовжуємо розшифровування зображення.

Оскільки перший піксель зображення буде завжди фрагмент із повною інформацією про піксель, тобто QOI_OP_RGBA або QOI_OP_RGB. Якщо умова виконується зчитуємо наступні три чи чотири байти та запам’ятовуємо їх.

```
if b1 == QOI_OP_RGB:
    px.r = bytes_array.popleft()
    px.g = bytes_array.popleft()
    px.b = bytes_array.popleft()
elif b1 == QOI_OP_RGBA:
```

```
    px.r = bytes_array.popleft()
    px.g = bytes_array.popleft()
    px.b = bytes_array.popleft()
    px.a = bytes_array.popleft()
```

Після визначення фрагменту ми записуємо у буфер пройдених пікселів значення пікселів, та переходимо до зчитування наступного байту та проведення над ним аналізу.

```
index[hash_f(px) % 64] = cp.copy(px)
```

Якщо умова перевірки на фрагменти QOI_OP_RGB або QOI_OP_RGBA не виконується, далі перевіряємо чи байти містить у двох старших бітах ідентифікатор фрагменту QOI_OP_INDEX. При позитивній перевірці, ми з буферу пройдених пікселів беремо піксель і додаємо його до результуючого масиву та переходимо до наступного байту.

```
elif (b1 & QOI_MASK_2) == QOI_OP_INDEX:
    px = cp.copy(index[b1])
```

Якщо всі вище загаді перевірки не виконуються, відбувається пошук ідентифікаторів фрагменту QOI_OP_DIFF та QOI_OP_LUMA. При успішному пошуку фрагменту QOI_OP_LUMA зчитується додатковий байт, та змінюється значення r, g, b, відповідно до значень. Для фрагменту QOI_OP_DIFF зчитування додаткового байту не потрібно. Перехід до аналізу наступного байту.

```
elif (b1 & QOI_MASK_2) == QOI_OP_DIFF:
    px.r += ((b1 >> 4) & 0x03) - 2
    px.g += ((b1 >> 2) & 0x03) - 2
    px.b += (b1 & 0x03) - 2
elif (b1 & QOI_MASK_2) == QOI_OP_LUMA:
    b2 = bytes_array.popleft()
    vg = (b1 & 0x3f) - 32
    px.r +=  vg - 8 + ((b2 >> 4) & 0x0f)
    px.g += vg
    px.b += vg - 8 + (b2 & 0x0f)
```

Останнім випадком ми ідентифікуємо фрагмент QOI_OP_RUN. Значення що несе у собі байт записуємо у змінну run, та дублюємо попередній піксель потрібну кількість разів.

Результатом функції декодування це масив пікселів із результатом, ширина та висота зображення, а також присутність альфа каналу.

```
mode = 'RGBA' if channels == 4 else 'RGB'

return pixels, (w, h), mode
```

3.3 Реалізація модифікованого алгоритму зі збільшеним буфером пройдених пікселів

Для реалізації модифікації алгоритму із збільшеним буфером пройдених пікселів, внесли наступні зміни в код.

```
def hash_f(px):
    return (px.r * 1155 + px.g * 2310 + px.b * 8085 + px.a * 12615)
```

Зміни в циклі:

```
index_pos = hash_f(px) % 16384

if (index[index_pos] == px):
    # bytes_array.append(QOI_OP_INDEX | index_pos)
    combined_value = QOI_OP_INDEX | (index_pos & 0x3FFF)
    bytes_array.append((combined_value >> 8) & 0xFF)
    bytes_array.append(combined_value & 0xFF)
```

Зміни для декодера:

```
elif (b1 & QOI_MASK_2) == QOI_OP_INDEX:
    index_b1 = ((b1 & 0x3f) << 8) | (bytes_array.popleft() & 0xff)
px = cp.copy(index[index_b1])
```

3.4 Реалізація модифікованого алгоритму зі збільшеним кусочком запам’ятовування кількості повторюваних пікселів

Для реалізації модифікації алгоритму зі збільшеним кусочком запам’ятовування кількості повторюваних пікселів. Зміни вплинули на перевірку максимального значення та кодування фрагменту. Зміни для функції декодера:

```
elif (b1 & QOI_MASK_2) == QOI_OP_RUN:

    run_length = ((b1 & 0x3f) << 8) | (bytes_array.popleft() & 0xff)
    run = run_length
```

Зміни для функції кодування

```
if px == px_prev:
    run += 1
    if run == 15872 or w_i * h_i >= total_size:
        run = run - 1
        combined_value = QOI_OP_RUN | (run & 0x3FFF)
```

```
        bytes_array.append((combined_value >> 8) & 0xFF)
        bytes_array.append(combined_value & 0xFF)
        run = 0
else:
    if run > 0:
        run = run - 1
        combined_value = QOI_OP_RUN | (run & 0x3FFF)
        bytes_array.append((combined_value >> 8) & 0xFF)
        bytes_array.append(combined_value & 0xFF)
        run = 0
```

Висновки до розділу 3

У цьому розділі було розглянуто програмне середовище PyCharm, в якому відбувалось тестування створення програмного коду початкового алгоритму та його двох модифікацій.

Розібрали ключову моменти програмного коду, та чим відрізняються між собою у реалізації кожна із модифікацій.

4. ЕКСЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДИФІКОВАНИХ АЛГОРИМТІВ СТИСНЕННЯ

4.1 Опис тестових наборів даних

Набір даних для тестування включає в себе такі зображення.

- Текстури, різного характеру та розміру,
- Одноколірні зображення,
- Фото різного характеру та розміру,
- Знімки екрану під час гри, та перегляду інтернету,
- Іконки, логотипи.

Для аналізу якості стиснення та порівняння результатів будемо порівнювати характеристики, такі як:

- Час кодування,
- Час декодування,
- Розмір стиснутого зображення у байтах

Враховуючи що зображення для зберігання чи передачі файлів, зображення групуються у папки і стискаються у архіви. Тож ми також перевіримо якість стиснення зображення у zip-архів.

Одноколірні зображення, це зображення що мають розмір 1920*1080 пікселів. Зафарбовані у різні кольори білий, червоний, жовтий та інші. Приклад наведено на рисунку 4.1

Рисунок 3.1 – Приклад одноколірного зображення

Текстури маю різних характери, які від розміру так і до інформації яку вони містять. Данні мають від 256*256 пікселів до 1024*2048 пікселів.

На них зображено листки, дерева, так і просто лінії, комп’ютери каміння, цегла та інші. Приклад текстур зображено на рисунку 4.2.

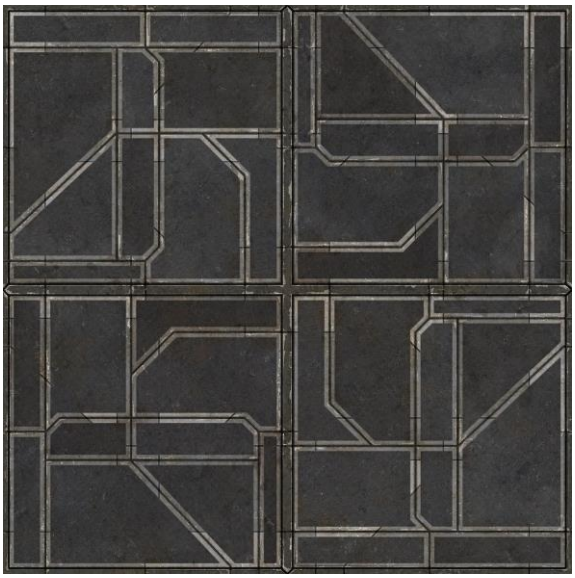


Рисунок 4.2 – Приклад текстурного зображення.

Фото зображення мають також різних розмір від 512*768 до 1428*714 пікселів. До фото також відносяться і зображення що були взяти із вікіпедії. Інформація на них досить різна від портретів та пейзажів до архітектури та зображень техніки чи репортажної зйомки(рисунок 4.3).

Рисунок 4.3 - Приклад фото зображення.

Також варто зазначити що були взяти і зображення створені графічними редакторами (рисунок 4.4).

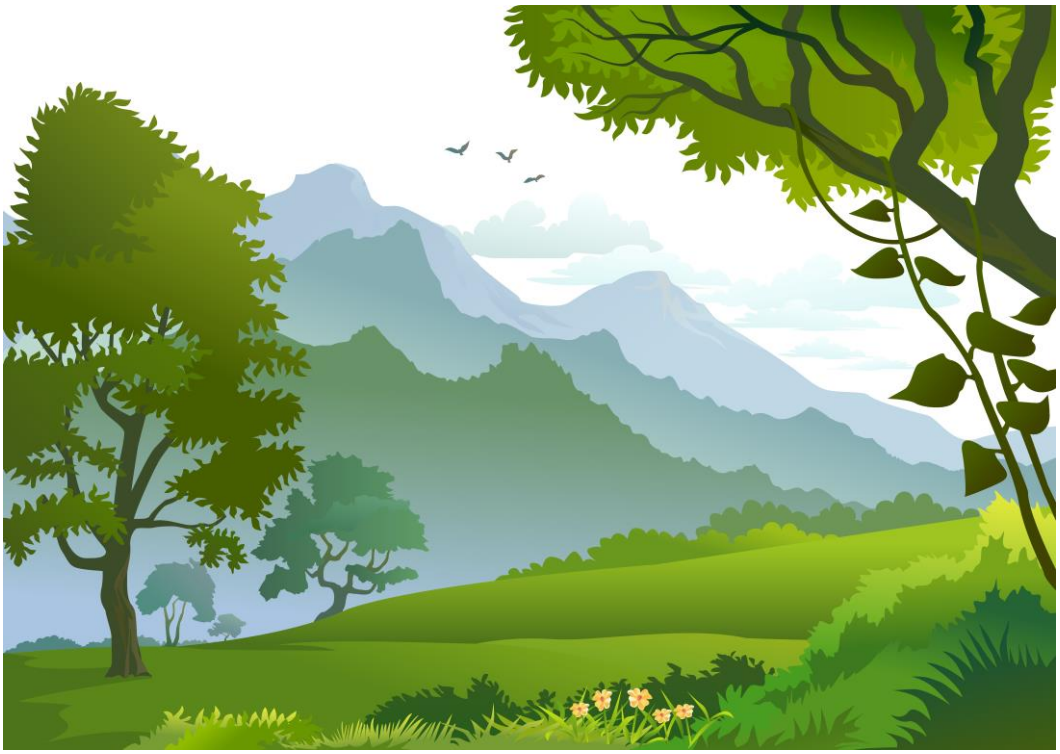


Рисунок 4.4 – Приклад зображення створеного графічними засобами

Іконки та логотипи це зображення що мають малий розмір від 64*64 до 512*512 пікселів, це ті елементи що використовуються для зображення кнопки зберігання у програмі Microsoft Word, чи оновлення сторінки тощо. Приклад зображено на рисунку 4.5.

Рисунок 4.5 – Приклад зображення іконки

4.2 Дослідження алгоритмів на швидкість кодування та декодування

На рисунку 4.6 та рисунку 4.7 зображено час стиснення та декодування зображень трьома алгоритмами, де index – це алгоритм де ми модифікували QOI_OP_INDEX, run – алгоритм, де було модифіковано фрагмент QOI_OP_RUN.

Рисунок 4.6 – Час стиснення одноколірних зображень.

Рисунок 4.7 – Час стиснення одноколірних зображень

Згідно графіку зображеного на рисунку 4.6 для одноколірних зображень середній час кодування рівний близько 12 секунд.

Середній час декодування зображень є меншим, ніж час кодування і становить близько 9 секунд.

В таблиці 5 відображено результати стиснення одноколірного зображення в чорному кольорі.

Таблиця 5 – Результат стиснення одноколірного зображення

Модифікація алгоритму	Зображення	Розмір, байт	Стиснутий розмір, байти	Відсоток стиснення	Час стиснення, секунди	Час декодування, секунди
ORIGINAL	black_1920_1080	8294457	36746	99,56	12,39	8,54
RUN	black_1920_1080	8294457	383	100	9,12	9,57
INDEX	black_1920_1080	8294457	36746	99,56	11,08	10,72

Згідно результатів бачимо, що оригінальний алгоритм та алгоритм з модифікацією QOI_OP_INDEX, де що програють алгоритму з модифікацією QOI_OP_RUN.

Скільки для кодування іконок були взяті два варіанти вхідних даних розміром 64*64 та 512*512 пікселів.

На рисунку 4.8 та рисунку 4.9 зображено графіки середнього часу стиснення та декодування.

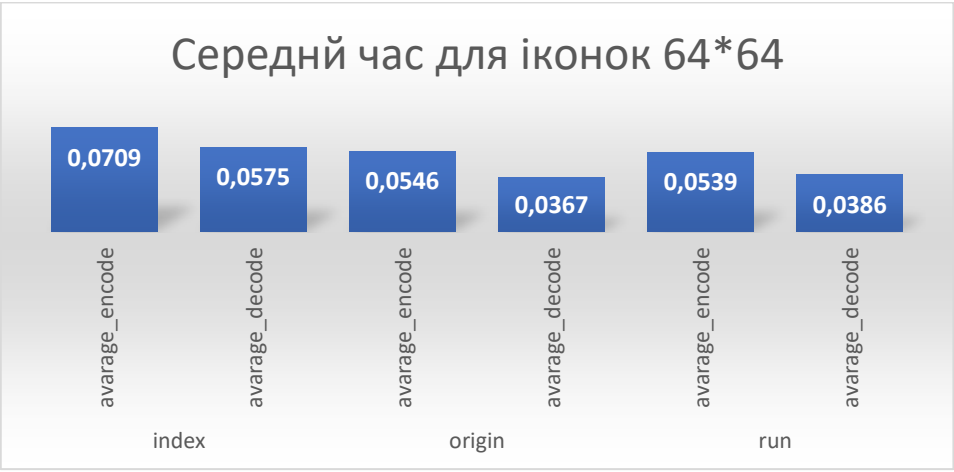


Рисунок 4.8 – Середній час стиснення та декодування зображень іконок 64*64 пікселі

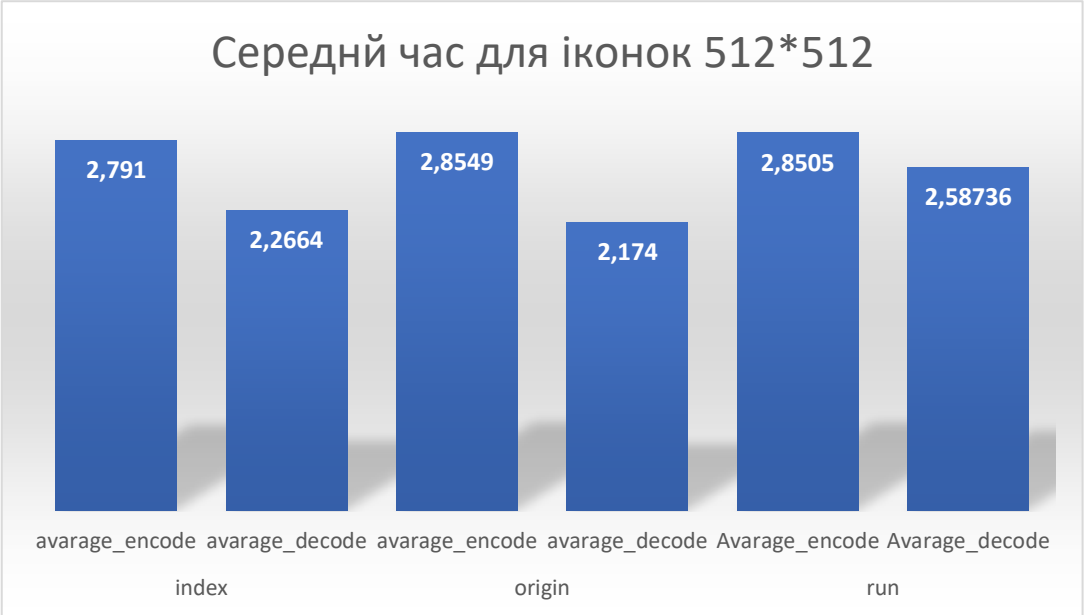


Рисунок 4.8 – Середній час стиснення та декодування зображень іконок 512*512 пікселі.

Як показуються там середні значення алгоритм із модифікацією QOI_OP_INDEX показує гірші результати, ніж оригінальний чи алгоритм із модифікацією QOI_OP_RUN. Останній близький до оригінального алгоритму і лише програє на декілька десятих секунд, як при кодуванні так і при декодуванні.

Для наступного тесту ми об’єднали знімки, які було зроблено на фотоапарат та зображення які були завантажені із загальнодоступної вікіпедії. Всі фотографії у даній вибірці мають різні розміру файлів та різні розширення, завдяки чому ми можемо чітко прослідкувати залежність часу кодування кожного алгоритму від його розміру, звичайно що є в плив і від змісту зображення. Аналогічно буде зображено також і результати декодування кожного із алгоритму. Всі дані наведені на рисунку 4.9 де зображено зміну часу кодування та декодування в залежності від розміру зображення.

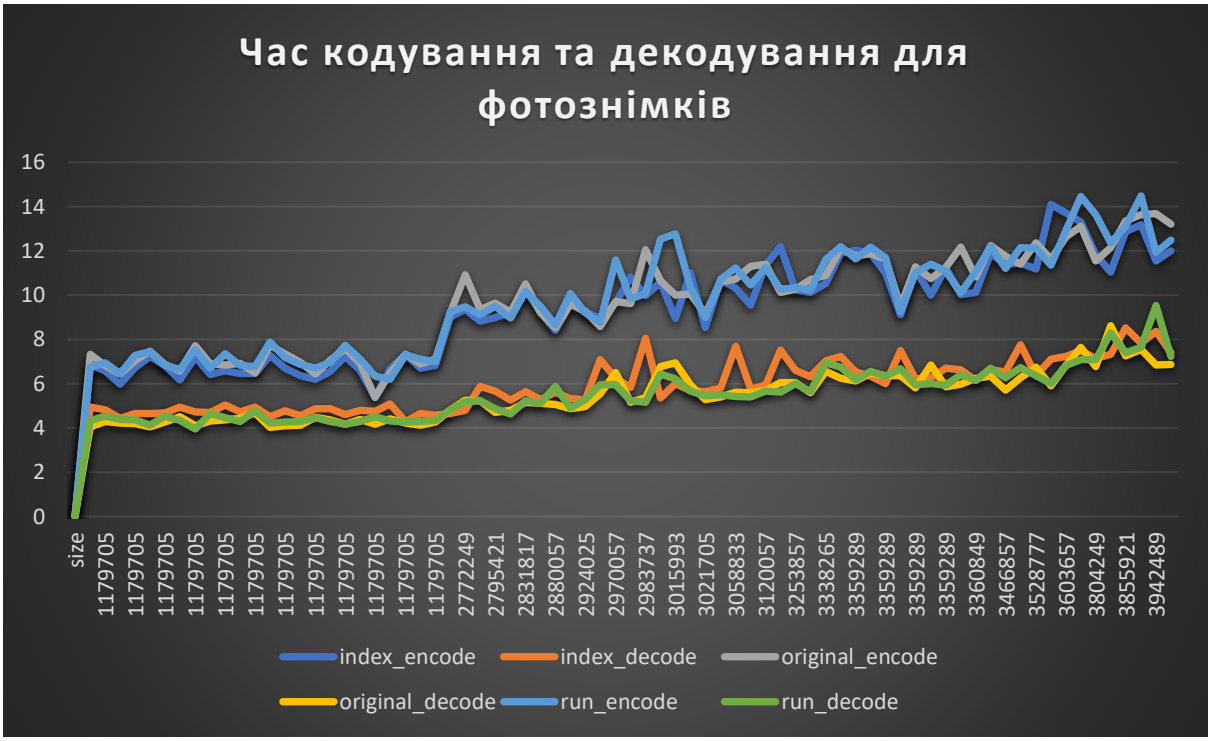


Рисунок 4.9 – Графік залежності часу кодування та декодування від розміру зображення

Результати графіку чітко показують що при збільшені розміру вхідного зображення час на виконання стиснення збільшується не залежно від алгоритму. При порівнянні між собою три алгоритми, у всіх час кодування та декодування практично не відрізняються між собою, проте час на кодування більш у всіх алгоритмах ніж час на декодування.

При тестуванні стиснення скріншотів зображень із інтернет ресурсів, спостерігалась схожа поведінка по часу кодування і декодування як і попередньому варіанті, всі алгоритми кодують дещо повільніше ніж декодують.

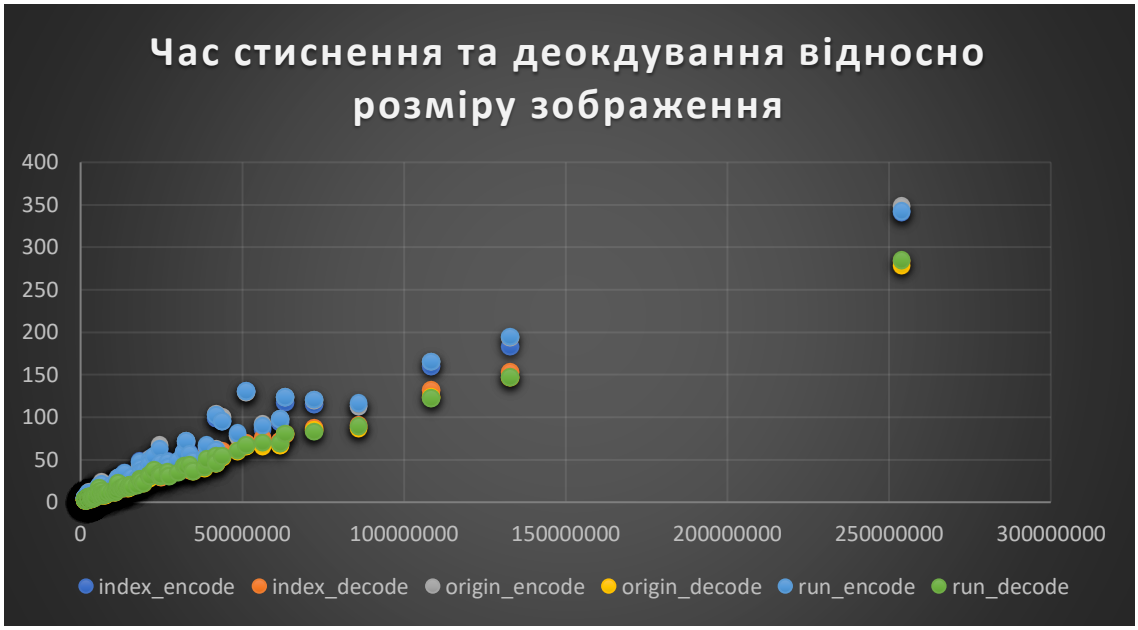


Рисунок 4.10 – Графік залежності часу кодування та декодування від розміру зображення

Отже, наші модифікації не покращують час кодування і декодування, водночас і не погіршують результат, тому що різниці між кодування і декодування у трьох алгоритмів невеликі.

4.3 Дослідження алгоритмів на якість стиснення

Отже дослідивши, модифікації практично не впливають на час кодування та декодування, порівняємо алгоритми між собою на якість стиснення.

Результат стиснення одноколірних зображень одного розміру наведені в таблиці 4.1, такі результати а саме, що найкращий результат буде у алгоритму QOI_OP_RUN, буде завжди прослідковуватися при стисненні одноколірного зображення. Перейдемо для аналізу результатів стиснення іконок розміром 64*64 та 512*512 пікселів. Враховуючи, що розміри зображення сталі, то якість стиснення буде насамперед залежить від алгоритму стиснення та якого характеру є інформація що відображається на іконці. Результати стиснення ста зображень розміром 64*64 наведені на рисунку 4.11.

Рисунок 4.11 – Графік коефіцієнтів стиснення для іконок

На рисунку 4.11 із графіком коефіцієнтів стиснення для іконок розміром 64*64 пікселі, схожі результати має і графік коефіцієнтів стиснення для іконок розміром 512*512 пікселів(рисунок 4.12).

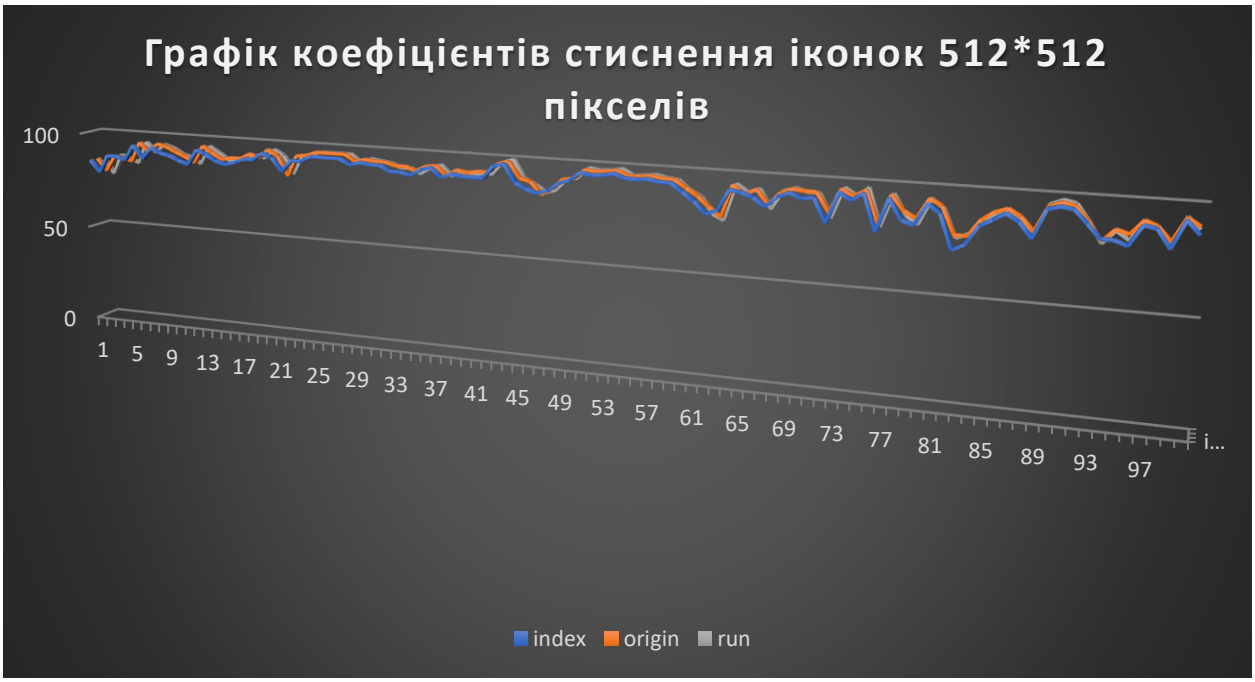


Рисунок 4.12 – Графік коефіцієнтів стиснення для іконок розміром 512*512

Для кожного із алгоритмів можемо побачити, що різниця у значеннях не суттєва. Всі значення практично не відрізняються між собою, але можемо виділити до алгоритм із модифікацією фрагменту QOI_OP_INDEX показує де що гірші результати. Оригінальний алгоритм та алгоритм із модифікацією QOI_OP_RUN практично відтворюють графік один одного. Слід звернути увагу, що при стисненні зображень розміром 512*512 значення самого коефіцієнту виросло і коливається в межах від 80% до 99% стиснення.

На рисунку 4.13 зображено графік залежності коефіцієнта стиснення від розміру зображень. Для тестування використовувались фото із портретами, архітектура, що людина може робити під час звичайного життя.

Рисунок 4.13 – Графік коефіцієнту стиснення від розміру зображення

При тестуванні на зображеннях, що були створені у графічних редакторах, або це є знімок екрану, отримали наступні результати.

Рисунок 4.14 – Графік коефіцієнту стиснення від розміру зображення.

Результати які отримали при стисненні тексту наведені на рисунку 4.15.

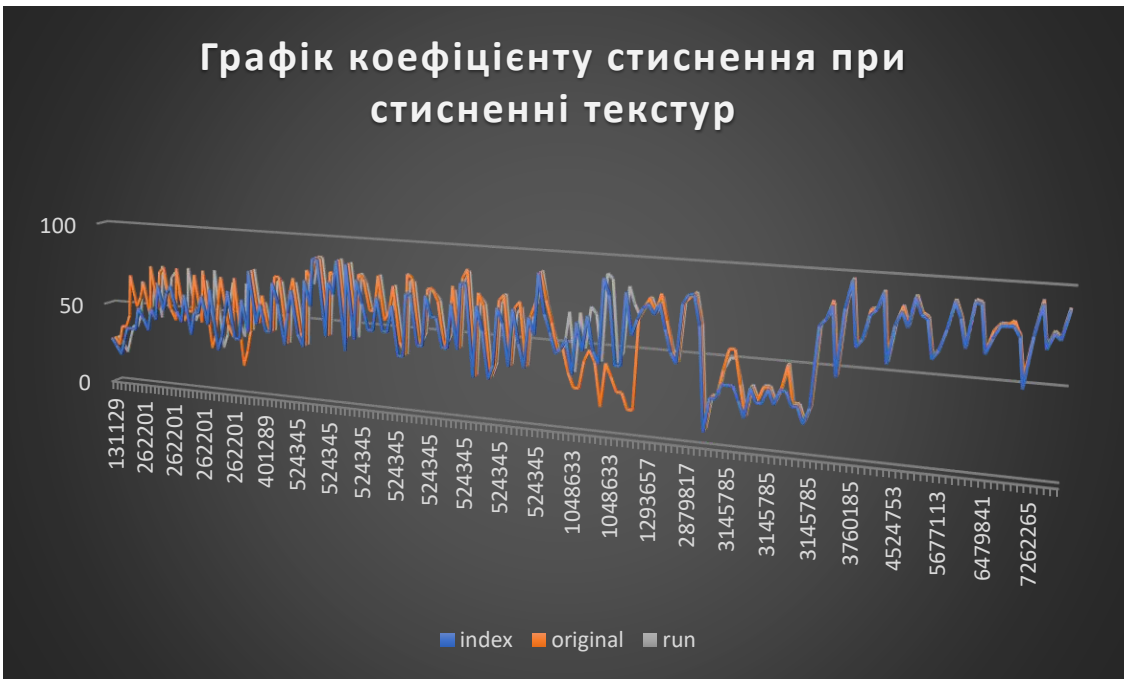


Рисунок 4.15 – Графік коефіцієнту стиснення від розміру зображення(текстури)

Бачимо, що в більшості випадків алгоритми стискають зображення досить ефективно і на одному рівні між собою. Згідно графіку бачимо, що при розмірі

зображення від 1048633 байт до 1293657 байт, оригінальний показав найгірші порівняно його модифікаціями. І в період коли розмір текстур був від 2879817 байт до 3145785 байт бачимо, що оригінальний алгоритм і алгоритм із модифікацією QOI_OP_RUN показали набагато кращі результати, ніж модифікація QOI_OP_INDEX.

Приклад одного із зображення, під час стиснення яких оригінальний алгоритм показав гірші результати, ніж обидві модифікації наведені на рисунку 4.16.

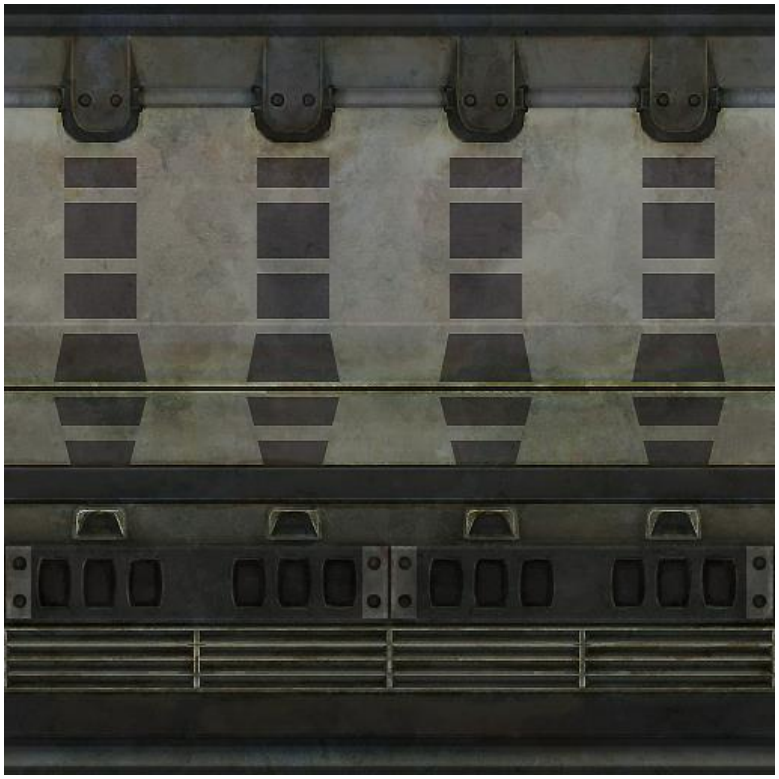


Рисунок 4.16 – Приклад зображення

Також на рисунку 4.17 наведено приклад зображення на якому бачимо що алгоритм із модифікацією QOI_OP_INDEX показує найгірші результати у порівнянні із іншими версіями алгоритму.

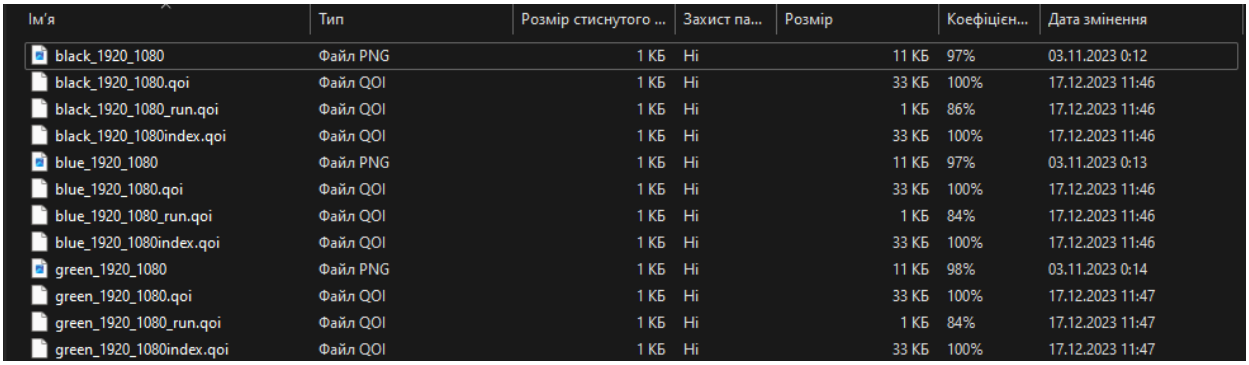
Рисунок 4.17–Приклад зображення

4.4 Дослідження на якість стиснення зображення у форматі QOI у zip-архівах

Вважаючи, що більшість користувачів зберігає дані у архівах, було проведено тестування для порівняння, насамперед, наскільки добре стискається даний формат у zip-архівах, та додатково порівняли коефіцієнти стиснення, початкові розміри стиснутих зображень за допомогою алгоритму QOI та популярного формату PNG.

У більшості випадків зображення у форматі PNG займають менше місця ніж зображення стиснуті за допомогою QOI, але є окрема категорія де PNG впорався набагато гірше ніж QOI.Варто зазначити що при стисненні у zip-архіви, файли із розширенням QOI стискалися набагато краще, у деяких випадках у стиснутому вигляді мали менший розмір ніж теж фото у PNG.

Отже при стисненні однокольорових зображень отримали результати, які зображені на рисунку 4.18.

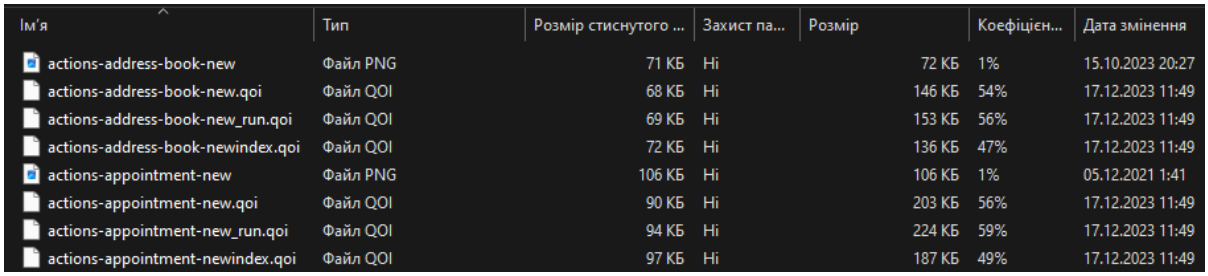


Ім'я	Тип	Розмір стиснутого ...	Захист па...	Розмір	Коефіцієн...	Дата змінення
black_1920_1080	Файл PNG	1 КБ	Ні	11 КБ	97%	03.11.2023 0:12
black_1920_1080.qoi	Файл QOI	1 КБ	Ні	33 КБ	100%	17.12.2023 11:46
black_1920_1080_run.qoi	Файл QOI	1 КБ	Ні	1 КБ	86%	17.12.2023 11:46
black_1920_1080index.qoi	Файл QOI	1 КБ	Ні	33 КБ	100%	17.12.2023 11:46
blue_1920_1080	Файл PNG	1 КБ	Ні	11 КБ	97%	03.11.2023 0:13
blue_1920_1080.qoi	Файл QOI	1 КБ	Ні	33 КБ	100%	17.12.2023 11:46
blue_1920_1080_run.qoi	Файл QOI	1 КБ	Ні	1 КБ	84%	17.12.2023 11:46
blue_1920_1080index.qoi	Файл QOI	1 КБ	Ні	33 КБ	100%	17.12.2023 11:46
green_1920_1080	Файл PNG	1 КБ	Ні	11 КБ	98%	03.11.2023 0:14
green_1920_1080.qoi	Файл QOI	1 КБ	Ні	33 КБ	100%	17.12.2023 11:47
green_1920_1080_run.qoi	Файл QOI	1 КБ	Ні	1 КБ	84%	17.12.2023 11:47
green_1920_1080index.qoi	Файл QOI	1 КБ	Ні	33 КБ	100%	17.12.2023 11:47

Рисунок 4.18 – Стиснення у zip-архівах однокольорових зображень.

При порівнянні між собою алгоритмів, то модифікація QOI_OP_RUN має найкращий результат без стиснення. Та при стисненні у архів розмір як і всіх інших алгоритмів в тому числі PNG зменшується до 1 кілобайти.

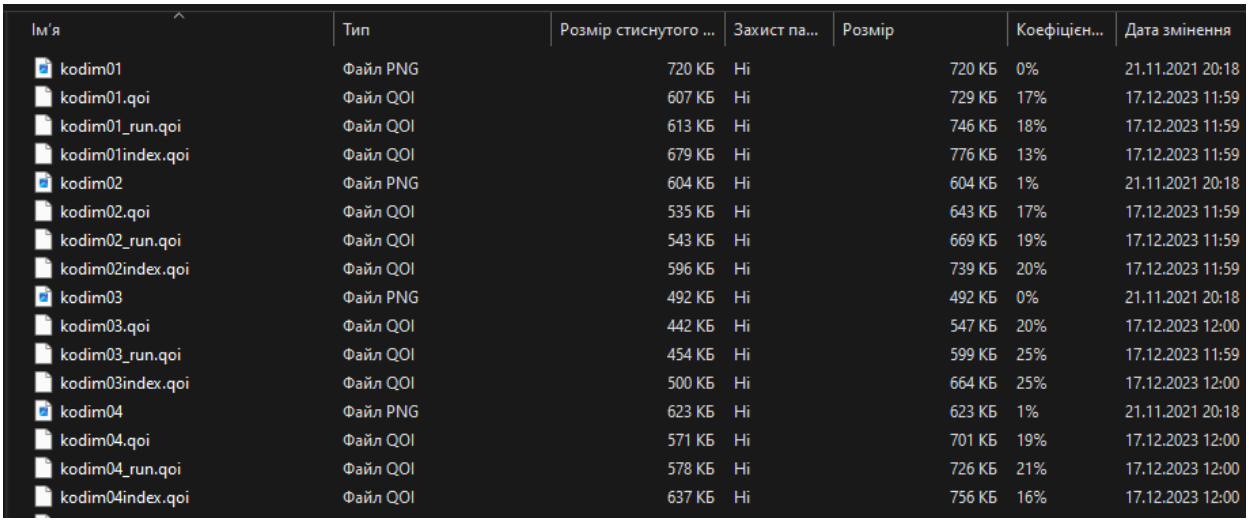
При стисненні зображень – іконок маємо схожий результат, що розміри одного і того ж зображення у форматі QOI, де що більші PNG, проте підчас стиснення розмір стиснутого зображення менший ніж PNG. Результат рисунку 4.19.



Ім'я	Тип	Розмір стиснутого ...	Захист па...	Розмір	Коефіцієн...	Дата змінення
actions-address-book-new	Файл PNG	71 КБ	Ні	72 КБ	1%	15.10.2023 20:27
actions-address-book-new.qoi	Файл QOI	68 КБ	Ні	146 КБ	54%	17.12.2023 11:49
actions-address-book-new_run.qoi	Файл QOI	69 КБ	Ні	153 КБ	56%	17.12.2023 11:49
actions-address-book-newindex.qoi	Файл QOI	72 КБ	Ні	136 КБ	47%	17.12.2023 11:49
actions-appointment-new	Файл PNG	106 КБ	Ні	106 КБ	1%	05.12.2021 1:41
actions-appointment-new.qoi	Файл QOI	90 КБ	Ні	203 КБ	56%	17.12.2023 11:49
actions-appointment-new_run.qoi	Файл QOI	94 КБ	Ні	224 КБ	59%	17.12.2023 11:49
actions-appointment-newindex.qoi	Файл QOI	97 КБ	Ні	187 КБ	49%	17.12.2023 11:49

Рисунок 4.19 – Стиснення у zip-архівах зображень іконок.

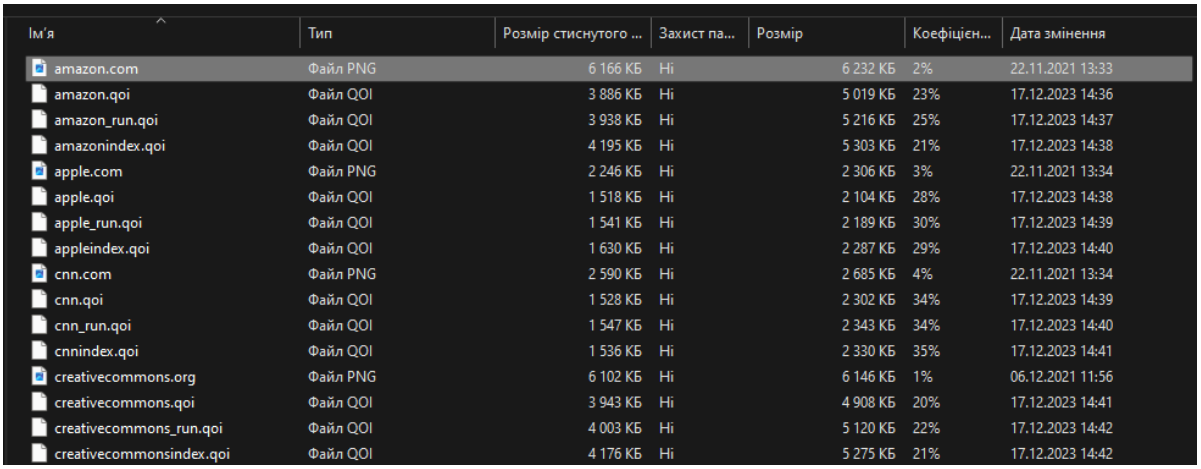
При досліджені стиснення фотографій, зображень завантажених із інтернет ресурсів результати зображені на рисунку 4.20.



Ім'я	Тип	Розмір стиснутого ...	Захист па...	Розмір	Коефіцієн...	Дата змінення
kodim01	Файл PNG	720 КБ	Ні	720 КБ	0%	21.11.2021 20:18
kodim01.qoi	Файл QOI	607 КБ	Ні	729 КБ	17%	17.12.2023 11:59
kodim01_run.qoi	Файл QOI	613 КБ	Ні	746 КБ	18%	17.12.2023 11:59
kodim01index.qoi	Файл QOI	679 КБ	Ні	776 КБ	13%	17.12.2023 11:59
kodim02	Файл PNG	604 КБ	Ні	604 КБ	1%	21.11.2021 20:18
kodim02.qoi	Файл QOI	535 КБ	Ні	643 КБ	17%	17.12.2023 11:59
kodim02_run.qoi	Файл QOI	543 КБ	Ні	669 КБ	19%	17.12.2023 11:59
kodim02index.qoi	Файл QOI	596 КБ	Ні	739 КБ	20%	17.12.2023 11:59
kodim03	Файл PNG	492 КБ	Ні	492 КБ	0%	21.11.2021 20:18
kodim03.qoi	Файл QOI	442 КБ	Ні	547 КБ	20%	17.12.2023 12:00
kodim03_run.qoi	Файл QOI	454 КБ	Ні	599 КБ	25%	17.12.2023 11:59
kodim03index.qoi	Файл QOI	500 КБ	Ні	664 КБ	25%	17.12.2023 12:00
kodim04	Файл PNG	623 КБ	Ні	623 КБ	1%	21.11.2021 20:18
kodim04.qoi	Файл QOI	571 КБ	Ні	701 КБ	19%	17.12.2023 12:00
kodim04_run.qoi	Файл QOI	578 КБ	Ні	726 КБ	21%	17.12.2023 12:00
kodim04index.qoi	Файл QOI	637 КБ	Ні	756 КБ	16%	17.12.2023 12:00

Рисунок 4.20 – Стиснення у zip-архівах зображень іконок.

Тут чітко прослідковуються вище згадані твердження, та і зауважимо що різниця між PNG на більших розмірах зображеннях зменшується. Візьмемо до уваги зображення kodim01. Розміри стиснутого файлу на 20% менші ніж від початкового. Така тенденція зберігається у більшості випадків які були протестовані, але на одному із наборі даних QOI показав найкращі результати не тільки у розмірі стиснутого зображення під час zip-архівування але і після обробки зображення алгоритмами QOI всіх модифікацій. До цієї категорії належать знімки екранів веб-ресурсів, та в деяких випадках текстури. Зазначимо, що розміри зображень веб-ресурсів коливається від 1313 пікселів * 6097пікселів, або 1313 пікселів * 4755 пікселів та інші. Тобто ці зображення мають досить великі розміри. Результати стиснення у zip-архів наведено на рисунку 4.21.



Ім'я	Тип	Розмір стиснутого ...	Захист па...	Розмір	Коефіцієн...	Дата змінення
amazon.com	Файл PNG	6 166 КБ	Ні	6 232 КБ	2%	22.11.2021 13:33
amazon.qoi	Файл QOI	3 886 КБ	Ні	5 019 КБ	23%	17.12.2023 14:36
amazon_run.qoi	Файл QOI	3 938 КБ	Ні	5 216 КБ	25%	17.12.2023 14:37
amazonindex.qoi	Файл QOI	4 195 КБ	Ні	5 303 КБ	21%	17.12.2023 14:38
apple.com	Файл PNG	2 246 КБ	Ні	2 306 КБ	3%	22.11.2021 13:34
apple.qoi	Файл QOI	1 518 КБ	Ні	2 104 КБ	28%	17.12.2023 14:38
apple_run.qoi	Файл QOI	1 541 КБ	Ні	2 189 КБ	30%	17.12.2023 14:39
appleindex.qoi	Файл QOI	1 630 КБ	Ні	2 287 КБ	29%	17.12.2023 14:40
cnn.com	Файл PNG	2 590 КБ	Ні	2 685 КБ	4%	22.11.2021 13:34
cnn.qoi	Файл QOI	1 528 КБ	Ні	2 302 КБ	34%	17.12.2023 14:39
cnn_run.qoi	Файл QOI	1 547 КБ	Ні	2 343 КБ	34%	17.12.2023 14:40
cnnindex.qoi	Файл QOI	1 536 КБ	Ні	2 330 КБ	35%	17.12.2023 14:41
creativecommons.org	Файл PNG	6 102 КБ	Ні	6 146 КБ	1%	06.12.2021 11:56
creativecommons.qoi	Файл QOI	3 943 КБ	Ні	4 908 КБ	20%	17.12.2023 14:41
creativecommons_run.qoi	Файл QOI	4 003 КБ	Ні	5 120 КБ	22%	17.12.2023 14:42
creativecommonsindex.qoi	Файл QOI	4 176 КБ	Ні	5 275 КБ	21%	17.12.2023 14:42

Рисунок 4.21 – Стиснення у zip-архівах зображень веб-сторінок.

Наприклад зображення із назвою amazon.com, в розширені PNG має розмір більше 6 кілобайт пам'яті, тоді як у всіх варіантах кодування алгоритмом QOI має, менший розмір і не більший ніж 5303 кілобайти, при цьому якість зображення не змінна. Розмір стиснутого зображення у zip-архіві, також виграє, тому що коефіцієнт стиснення для PNG у даному випадку сягає 2%, тобто все ж і досі займає більше шести кілобайт пам'яті, а розмір стиснутого зображення у форматі qoi в zip-архіві займає практично у півтори рази менше.

Висновки до розділу 4

У даному розділі ми провели тестування та аналіз результатів щодо стиснення зображень за допомогою алгоритму QOI, та двох модифікацій із збільшенням фрагменту та буфера QOI_OP_INDEX та із збільшеним фрагментом QOI_OP_RUN.

Провели тести, що показують як ефективно стискаються зображення із розширенням qoi у zip-архівах. Та порівняли якість стиснення із цим ж зображеннями у форматі png.

Таким чином можемо зробити наступні висновки по модифікаціях алгоритму. За часом стиснення та декодування всі три алгоритми справляються практично одночасно, їх часові результати лежать один біля одного.

За коефіцієнтом стиснення алгоритм із модифікацією фрагменту QOI_OP_INDEX програє оригінальному алгоритму та алгоритму із модифікацією фрагменту QOI_OP_RUN. Якщо порівнювати останні два алгоритми між собою то алгоритм із модифікацією фрагменту QOI_OP_RUN практично однаково гарно стискає як і оригінальний та у випадку одноколірних зображень показує кращі результати ніж оригінальний алгоритм.

При стисненні зображень у zip-архів, алгоритм показує краще відсоткове стиснення ніж у формат PNG.

ВИСНОВКИ

Метою роботи було підвищення ефективності алгоритму стиснення без втрат QOI. Проаналізувавши алгоритм на можливість модифікацій для збільшення його ефективності, ми визначили дві модифікації: розширення фрагменту QOI_OP_RUN і збільшення буферу пройдених пікселів.

Перша модифікація із збільшення фрагменту QOI_OP_RUN з одного до двох байт, дала змогу збільшити діапазон кількості повторюваних пікселів поспіль з 62 до 15872. Ця модифікація покращила результати стиснення одноколірних зображень. Зображення набагато ефективніше стискаються, ніж притисненні оригінальним алгоритмом. При стисненні звичайних зображень(текстур, знімки екрану, фото і т. д.), модифікований алгоритм показує практично одні і ті ж результати, що і оригінальний.

Друга модифікація із збільшення буферу пройдених пікселів, в наслідок якого було збільшено фрагмент QOI_OP_INDEX з одного до двох байтів, таким чином буфер збільшився із 64 пікселів до 16384 пікселів. Однак ця модифікація на всіх тестах не показала очікуваного результату. Алгоритм потребує більше часу на стиснення в порівнянні із оригінальним алгоритмом та модифікацією фрагменту QOI_OP_RUN.

Стиснуті зображення алгоритмом QOI, незалежно від модифікацій, демонструють кращу ефективність у ZIP-архівах порівняно із форматом PNG.

Заміна оригінального алгоритму на модифікований, зокрема з розширеним фрагментом QOI_OP_RUN, виявилася більш продуктивною.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. Fitria, L.A., Purboyo, T.V. A review of data compression techniques. International Journal of Applied Engineering Research. 2017. Вип .12 pp. 8956-8963

2. Демиденко М.Г., Федченко О.В. Крос-платформні мови програмування, Суми, Міністерство освіти і науки України, Сумський державний університет, 2010 - 5 с.

3. В. Г. Маценко. Комп'ютерна графіка. Чернівці, «Рута», 2009 – 184-188 с.

4. Бондар І. О. ТЕОРІЯ КОЛЬОРУ. Харків, ХНЕУ ім. С. Кузнеця, 2016 - 18-19 с.

5. Інформатика 8 клас / Ривкінд Й.А., Лисенко Т. І., Чернікова Л.А., Шакотько В. В., Київ, 2021, 12-14 с

6. Caplan P. «What Is a JPEG? The Invisible Object You See Every Day» ULR: https://www.theatlantic.com/technology/archive/2013/09/what-is-a-jpeg-the-invisible-object-you-see-every-day/279954/ (дата звернення: 20.11.2023)

7. «What is Image Compression?» URL : https://www.geeksforgeeks.org/what-is-image-compression/ (дата звернення: 20.11.2023)

8. «Concept of Quantization» URL: https://www.tutorialspoint.com/dip/concept_of_quantization.htm (дата звернення 20.11.2023)

9. Білинський Й. «Динамічне кодування методом Хаффмена» URL: https://web.posibnyky.vntu.edu.ua/firen/6bilynskyj_elektronni_systemy/564.htm (дата звернення: 20.11.2023)

10. «Flate/deflate compression» URL: https://www.prepressure.com/library/compression-algorithm/flate-deflate (дата звернення: 22.11.2023)

11.«Lempel-Ziv-77 (LZ77)» URL: https://web.archive.org/web/20100407084220/http://www.binaryessence.com/dct/en000138.htm (дата звернення: 22.11.2023)

12. Szablewski D. «The Quite OK Image Format for Fast, Lossless Compression» URL: https://qoiformat.org/ (дата звернення: 22.11.2023)

13. Szablewski D. «THE QUITE OK IMAGE FORMAT» URL: https://qoiformat.org/qoi-specification.pdf (дата звернення: 22.11.2023)

14. Szablewski D. «QOI - The Quite OK Image Format» URL: https://qoiformat.org/benchmark/ (дата звернення: 22.11.2023)

15. «Що таке прискорювач?» URL: https://uk.theastrologypage.com/accelerator (дата звернення: 12.12.2023)

16. Ilyashov Oleksandr Класифікація даних апаратними прискорювачами fpga у центрах обробки даних та хмарах / Oleksandr Ilyashov, Kostiantyn Pokora, Vladislav Diachenko, Andriy Kovalenko // *Системи управління, навігації та зв'язку. Збірник наукових праць*. – Полтава: ПНТУ, 2023. – Т. 2 (72). – С. 106-112. – doi:https://doi.org/10.26906/SUNZ.2023.2.106.

17. «SIMD архітектура» URL: https://ua5.org/technol/2807-simd-arhitektura.html (дата звернення: 12.12.2023)

18. «MIMD архітектура» URL: https://ua5.org/technol/2811-mimd-arhitektura.html (дата звернення: 12.12.2023)

19. «SISD комп'ютери» URL: http://um.co.ua/1/1-1/1-134245.html (дата звернення: 12.12.2023)

20. Клятченко, Я., Голуб, В. (2023). ЕФЕКТИВНІСТЬ МОДИФІКАЦІЇ АЛГОРИТМУ УЩІЛЬНЕННЯ ДАНИХ БЕЗ ВТРАТ. *Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології*, (2 (10), 67–72. https://doi.org/10.20998/2079-0023.2023.02.10