МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Кафедра радіофізики та комп’ютерних технологій

КУРСОВА РОБОТА

на тему:

«Застосування фракталів для стиснення інформації»

Виконав:
студент групи ФеІм-11с
Гаращак А. Б.

Науковий Керівник:
проф. Болеста І. М.

Львів – 2023 рік

Зміст

АНОТАЦІЯ 3
ANOTATION 4
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 5
1.1 Вступ 5
1.2 Історія винайдення фракталів та їх основні характеристики 5
1.3. Приклади фракталів 6
1.4 Фрактали в інформаційних технологіях. Опис та сфери застосування. 8
1.5 Наукометричний аналіз теми роботи 9
РОЗДІЛ 2. ЗАСТОСУВАННЯ ФРАКТАЛІВ ДЛЯ СТИСНЕННЯ ІНФОРМАЦІЇ 11
2.1. Опис підходу 11
2.2. Переваги та недоліки фрактального стиснення 12
2.3. Сфери застосування 13
РОЗДІЛ 3. ОПИС ТА РЕЗУЛЬТАТИ РОБОТИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ФРАКТАЛЬНОГО СТИСНЕННЯ ЗОБРАЖЕНЬ 15
3.1 Постановка задачі 15
3.2. Опис використаних технологій 15
3.3. Опис розробленого алгоритму стиснення 15
3.4. Опис розробленого алгоритму декодування 16
3.5 Опис роботи програми 17
3.6 Аналіз отриманих результатів 18
ВИСНОВКИ 20
СПИСОК ЛІТЕРАТУРИ 21
ДОДАТКИ 22
Додаток А – код алгоритму стиснення 22
Додаток В - код алгоритму декодування зображення 29

АНОТАЦІЯ

В цій курсовій роботі мною розроблено застосунок, який дозволяє стиснути зображення, використовуючи фрактальні алгоритми стиснення. Мною проаналізовано тему роботи та наявні алгоритми, та реалізовано програму, яка виконує поставлену задачу.

Програму написано мовою програмування C# використовуючи фреймворк для графічних інтерфейсів WPF (Windows Presentation Foundation).

В записці детально описано алгоритм, який було реалізовано, а також історію фрактальної графіки та її застосування.

Також мною проаналізовано результати роботи розробленої програми та наведено висновки.

ANOTATION

In this term paper, I have developed an application that allows you to compress images using fractal compression algorithms. I analyzed the topic of work and existing algorithms, and implemented a program that performs the task.

The program was written in C# using the WPF (Windows Presentation Foundation) framework for graphical interfaces.

The note describes in detail the algorithm that was implemented, as well as the history of fractal graphics and its application.

I also analyzed the results of the developed program and drew conclusions.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вступ

Фрактал - це геометрична фігура, що містить детальну структуру на як завгодно малих масштабах.

Фрактали можна побудувати за допомогою математичних формул, які описують їх структуру та взаємодію їх частин. Один з найвідоміших фракталів - це так звана зірка Коха, яка складається з багатьох тріедрів, додаткових до кожного кута.

Інші відомі фрактали: множина Мандельброта – множина компоексних чисел що відповідають певному закону, килим Серпінського, який формується поділом квадрата на менші квадрати за певним законом.

Фрактали мають важливу роль у науці та технології, вони використовуються для моделювання складних систем, таких як клітинні структури, економічні тенденції, метеорологічні процеси та інші.

Також фрактали часто зустрічаються в природі, мистецтві, архітектурі. Наприклад, у вигляді віток дерев, хмар або гірських ланцюгів.

1.2 Історія винайдення фракталів та їх основні характеристики

Історія винайдення та розвитку фракталів має багато важливих етапів. Ось деякі ключові моменти в історії фракталів:

Математичні корені: Перші посилання на фрактальні об'єкти знаходяться у роботах англійського математика Леонарда Ейлера у XVIII столітті. Він розглядав ряди і функції, які виявляли фрактальну природу.

Поліноми Жюліа та Мандельброта: У 19-му столітті французький математик Гастон Жюліа та польський математик Бенуа Мандельброт досліджували поліноміальні функції з рекурентними правилами. Вони відкрили складну структуру і взаємозв'язок між числовими послідовностями, що призвело до поняття фракталу.

Теорія фракталів: В 1975 році Бенуа Мандельброт запропонував термін "фрактал" для опису складних структур, які мають самоподібність

на різних масштабах. У своїй книзі "Фрактальна геометрія природи" він розробив систематичну теорію фракталів і визначив поняття фрактальної геометрії.

Комп'ютерна графіка: В 1980-х роках фрактальні зображення стали популярними завдяки розробці алгоритмів комп'ютерної графіки. Фрактальна графіка дозволяла генерувати складні та реалістичні образи з великою деталізацією, що привернуло увагу громадськості.

Розширення досліджень: Після розробки теорії Мандельбротом та поширення використання фракталів у комп'ютерній графіці, дослідження в галузі фрактальної геометрії розширилися на різні наукові дисципліни, включаючи фізику, біологію, економіку та інші галузі.

1.3. Приклади фракталів

Зірка Коха: одною з основних особливостей фрактальної кривої Коха є те, що в кожній точці вона не має похідної, а сама крива і кожна її частина має нескінченну довжину. Алгоритм побудови кривої Коха виглядає наступним чином: береться одиничний інтервал, поділяється на три рівні частини і замінюється середній інтервал рівностороннім трикутником без цього сегмента. В результаті утворюється ламана, що складається з чотирьох ланок довжиною 1/3 (крива Коха першого порядку). На наступному кроці повторюються аналогічні дії для кожної з чотирьох одержаних ланок (крива Коха другого порядку). Ці кроки можуть повторюватись нескінченну кількість разів.

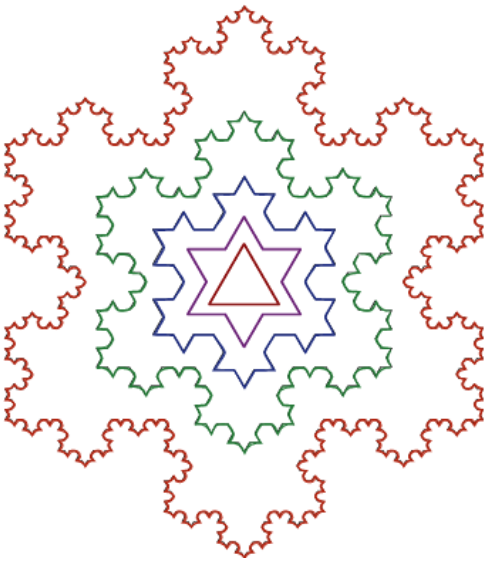

Рис. 1.1. Приклад сніжинки Коха різних порядків (1-5)

Килим Серпінського: це плоский фрактал, вперше описаний Вацлавом Серпінським в 1916 році. Килим є одним із прикладів множини Кантора у двох вимірах (у більших вимірах — хмари Кантора). Серпінський продемонстрував, що цей фрактал є універсальною кривою, де будь-який можливий одновимірний граф, спроектований на двовимірну площину, гомеоморфний до підмножини серветки Серпінського.

Побудова килима Серпінського починається із квадрата. Квадрат розрізається на 9 конгруентних підквадратів, що утворюють сітку три на три, і центральний підквадрат видаляється. Та ж процедура нескінченно рекурсивно застосовується до вісьмох квадратів, що залишилися.

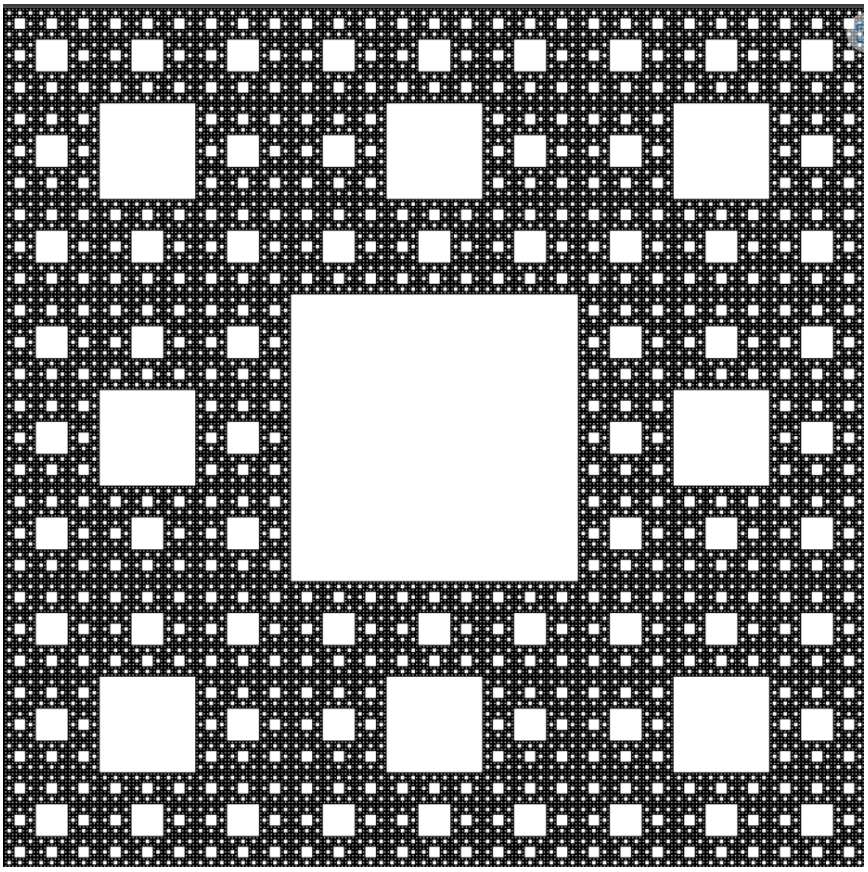

Рис. 1.2. Приклад фракталу «Килим Серпінського»

1.4 Фрактали в інформаційних технологіях. Опис та сфери застосування.

З розвитком інформаційних технологій збільшується складність та об’єм різноманітних даних, які опрацьовуються. Також невпинно збільшується обсяг інформації, яку потрібно зберігати. Це все призводить до того, що потрібно знаходити зручні та ефективні методи для обробки інформації. Саме тому в інформаційних технологіях почали використовувати фрактали та їх властивості. Основна їх перевага полягає в тому, що завдяки властивостям самоподібності можна набагато спростити процес обробки чи опису складних даних чи об’єктів.

Нижче наведено приклади застосування фракталів в різних галузях інформаційних технологій:

1. Графіка: Фрактали дозволяють створювати складні інтерактивні графічні ефекти. Наприклад, алгоритми використовуються для створення реалістичних ландшафтів, обробки зображень і створення геометричних форм.

2. Комп'ютерне моделювання: Фракталами можна описувати складні структури в природі або соціальних системах. Вони використовуються для моделювання різноманітних систем, таких як клітинні автомати, популяції тварин, фінансові ринки та інші.

3. Стиснення даних: алгоритми стиснення з використанням фракталів дозволяють ефективно стискувати дані шляхом виявлення самоподібних (фрактальних) закономірностей та кодування їх з меншою кількістю бітів.

4. Криптографія: Фрактальна криптографія є новим підходом до захисту даних та інформації. Вона базується на математичних властивостях фракталів, що забезпечують високий рівень криптографічної стійкості та варіативності.

5. Музика та звук: Фрактальна музика використовує фрактальні алгоритми для створення музичних композицій зі складними структурами та варіаціями. Також фрактальні алгоритми можуть бути використані для синтезу звуків та обробки аудіо.

6. Штучний інтелект: З використанням фракталів моделюються складні структури нейронних мереж, а також фрактали використовуються для тренування нейронних мереж в алгоритмах машинного навчання.

1.5 Наукометричний аналіз теми роботи

Наукометрія – є галуззю науки, яка вивчає наукову діяльність і наукову продуктивність. Вона зосереджується на вимірюванні та аналізі наукових публікацій, цитувань, авторської продуктивності, впливу наукових журналів і дослідників, а також на розробці метрик і інструментів для оцінки наукової продуктивності та впливу.

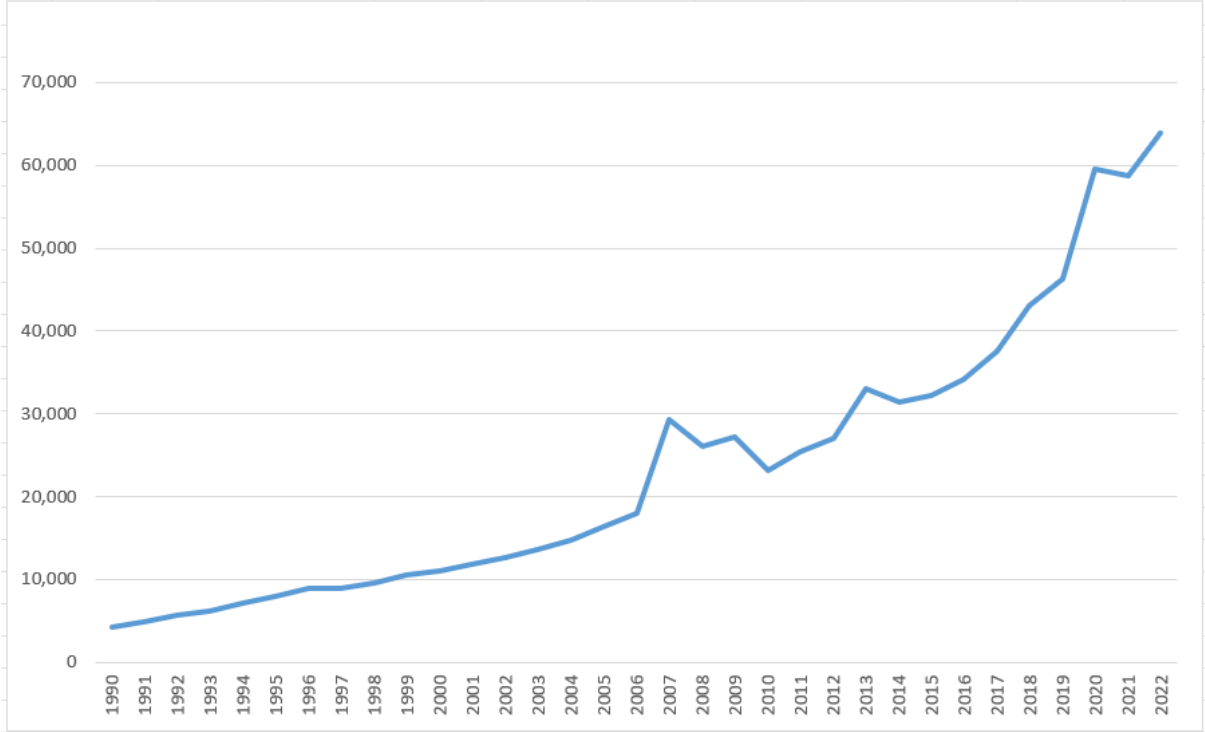

Рис. 1. Кількість публікацій на тему фракталів в рік в період з 1990 по 2022 роки

Дивлячись на рис 1 можна чітко побачити експотенційну залежність. Цей ріст кількості публікацій означає розвиток тематики фракталів та збільшення їх застосування в науці і технологіях.

В період з 2005 по 2009 роки можна бачити логістичну залежність, це означає, що в той період було досягнуто насичення – відсутність росту та після цього спад. Це вказує на те, що в той період був менший інтерес до теми фракталів. Це може бути пов’язано з розвитком інших галузей науки або відсутністю нових досліджень.

РОЗДІЛ 2. ЗАСТОСУВАННЯ ФРАКТАЛІВ ДЛЯ СТИСНЕННЯ ІНФОРМАЦІЇ

2.1. Опис підходу

Для стиснення інформації за допомогою фракталів використовується техніка під назвою “Фрактальне стиснення”. Воно базується на ідеї самоподібності, коли невелика частина зображення або набору даних нагадує загальну структуру цілого.

Процес стиснення в загальному складається з таких кроків:

1. Розбиття на частини: Дані розділяються на блоки або області невеликого розміру. Наприклад, при стисненні зображень їх ділять на менші квадратні блоки.

2. Початкове наближення: Для кожного блоку вибирається початкове наближення. Це робиться шляхом вибору репрезентативного блоку з набору даних або використовуючи простий алгоритм, наприклад, усереднення значень пікселів у блоці.

3. Зіставлення та кодування: Порівнюється кожен блок з іншими блоками в наборі даних, щоб знайти подібні або ідентичні шаблони. Зазвичай це робиться за допомогою методів, таких як середня квадратична похибка (MSE) або сума абсолютних різниць (SAD).

4. Заміна та доопрацювання: Подібні або ідентичні блоки замінюються математичним перетворенням або кодуванням еталонного блоку. Це кодування має бути меншим за розміром, ніж вихідний блок. Еталонний блок можна масштабувати, повертати та позиціонувати так, щоб він максимально відповідав оригінальному блоку.

5. Ітеративне вдосконалення: Повторюються кроки 3 і 4 для декількох ітерацій, покращуючи якість апроксимації з кожною ітерацією. Це дозволяє отримати більш точне представлення вихідних даних.

6. Кодування та декодування: Кодування перетворених блоків разом з будь-якою необхідною інформацією, необхідною для реконструкції

вихідних даних. Процес кодування може включати такі методи, як кодування довжини прогону, кодування Хаффмана або інші методи ентропійного кодування.

7. Реконструкція: Щоб розпакувати дані, закодовані блоки декодуються і повертаються до своїх початкових позицій. Потім перетворені блоки об'єднуються для відновлення вихідних даних.

Фрактальне стиснення вимагає значних обчислень і не завжди перевершує інші методи стиснення, такі як JPEG або MPEG. Ефективність залежить від самоподібності та фрактальних властивостей даних, що стискаються.

2.2. Переваги та недоліки фрактального стиснення

Фрактальне стиснення базується на самоподібності. Це означає, що алгоритм має високу залежність від властивостей даних: самоподібності та можливості їх стиснення та подальшого відновлення.

Можна визначити такі переваги алгоритму:

- Висока ступінь стиснення. Якщо дані маюсь високу ступінь самоподібності, то їх можна зберігати з високою точністю витрачаючи як найменший обсяг пам’яті. Це стосується в першу чергу зображень.
- Відновлення якості. Стиснені дані можуть бути відновлені з низьким рівнем втрат. Це означає, що відновлені дані будуть дуже близькими до початкових даних.
- Масштабованість. Фрактальне стиснення дозволяє зберігати дані на різних масштабах без втрати деталей. Це означає, що дані можуть бути збережені у високій роздільній здатності, а також можуть бути масштабовані для відтворення на різних пристроях .

Недоліки алгоритмів фрактального стиснення

- Висока обчислювальна складність. Хоча алгоритми даюсь досить високу ступінь стиснення, та це вимагає багато обчислень, що може бути досить не вигідно для великого об'єму даних – обсяг витрачених обчислювальних потужностей переважає отримані переваги в обсягу зайнятої пам'яті завдяки стисненню.
- Відносна ефективність. Алгоритми фрактального стиснення дуже залежні від даних, які потрібно стиснути. Якщо дані мають низький ступінь самоподібності, то обсяг зекономленої пам'яті дуже малий, що робить застосування алгоритму недоцільним. Також можлива ситуація, що під час стиснення може втратитись багато деталей (в випадку зображень) чи інформації (в випадку інших даних), що також не є добре і нівелює переваги алгоритму.

2.3. Сфери застосування

В загальному розумінні, фрактальне стиснення може бути застосоване в будь-якій галузі, де важливе стиснення даних з мінімальною втратою інформації. Проте можна виділити декілька основних галузей де цей підхід застосовується найбільше:

Зображення: Алгоритм використовується для стиснення цифрових зображень - вони поділяються на менші блоки, знаходиться фрактальний код для кожного блоку та зберігається лише цей код замість фактичного зображення.

Відео: Фрактальне стиснення також може бути використане для стиснення відеоданих. Алгоритми використовуються для стиснення послідовностей кадрів шляхом знаходження фрактального коду для кожного кадру та зберігання лише цього коду замість повних піксельних значень кожного кадру.

Звук: Використовуючи фрактальні алгоритми аналізується аудіо сигнал та знаходяться фрактальні закономірності, які можуть бути використані для подальшого синтезу аудіо з меншою кількістю даних.

Комп'ютерна графіка: У галузі комп'ютерної графіки фрактальне стиснення може бути використане для стиснення текстур та інших графічних елементів. Це дозволяє ефективно зберігати та передавати графічні ресурси, зменшуючи обсяг даних, необхідних для їх представлення.

РОЗДІЛ 3. ОПИС ТА РЕЗУЛЬТАТИ РОБОТИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ФРАКТАЛЬНОГО СТИСНЕННЯ ЗОБРАЖЕНЬ

3.1 Постановка задачі

Завдання полягає в розробці програмного забезпечення для стиснення зображень методом фрактального стиснення.

Програма повинна містити такий функціонал:

1. Вибір зображення для стиснення.
2. Задання параметрів стиснення, а саме розміру блоку.
3. Стиснення зображення алгоритмом фрактального стиснення зображення.
4. Збереження результату стиснення в файл, що дозволяє потім відновити початкове зображення з мінімальними втратами.
5. Можливість відтворити зображення на основі збережених даних.

Програма повинна працювати на комп’ютерах з операційною системою Windows.

3.2. Опис використаних технологій

Для розробки ПЗ було використано такі технології та засоби:

- Мова програмування C#.
- Фреймворк для розробки графічного інтерфейсу Windows Presentation Foundation.
- Середовище розробки – Microsoft visual studio 2022.
- Комп’ютер з операційною системою Windows 11.

3.3. Опис розробленого алгоритму стиснення

Алгоритм стиснення з вхідного зображення формує об’єкт, який містить інформацію про блоки вхідного зображення, на основі яких в

подальшому можна відновити початкове зображення з мінімальними втратами.

Послідовність дій:

1. Отримання даних пікселів для подальшого порівняння. Використано такі параметри як яскравість, відтінок та насиченість.
2. Обчислення середніх значень для блоків на основі вибраного розміру блоку.
3. На основі даних кожного блоку проводиться порівняння всіх блоків та відбувається процес трансформації – найбільш схожі блоки замінюється на один блок.
4. Збереження отриманої інформації про трансформації та про характеристики блоку – це є результат роботи алгоритму, стиснені дані.

Через те, що відбувається повний перебір всього зображення цей процес досить повільний.

Результат роботи алгоритму зберігається в файл, який використовується в алгоритмі декодування.

Код алгоритму поданий в додатку А

3.4. Опис розробленого алгоритму декодування

Алгоритм декодування працює на основі результату кодування. Послідовність дій алгоритму:

1. Визначення параметрів алгоритму на основі закодованих даних та зображення для декодування (зображення, на якому будуть виконані перетворення).
2. Знаходження матриць перетворення
3. На основі запису даних про оригінальне зображення та його трансформації відбувається відтворення параметрів закодованого

зображення (яскравість, відтінок та насиченість) – використовуються матриці перетворень та спеціальні формули.

4. З обчислених даних формуються дані про пікселі зображення
5. Використовуючи отримані дані формується вихідне зображення, яке з певним рівнем точності (він залежить від властивостей закодованого зображення) відтворює його.

Код алгоритму поданий в додатку В

3.5Опис роботи програми

Розроблена програма запускається в вигляді вікна. Серед доступних елементів керування – кнопка для вибору зображення, яке потрібно стиснути поле вводу для вказування розміру блоку, а також кнопка, яка запускає алгоритм стиснення зображення та збереження результатів.

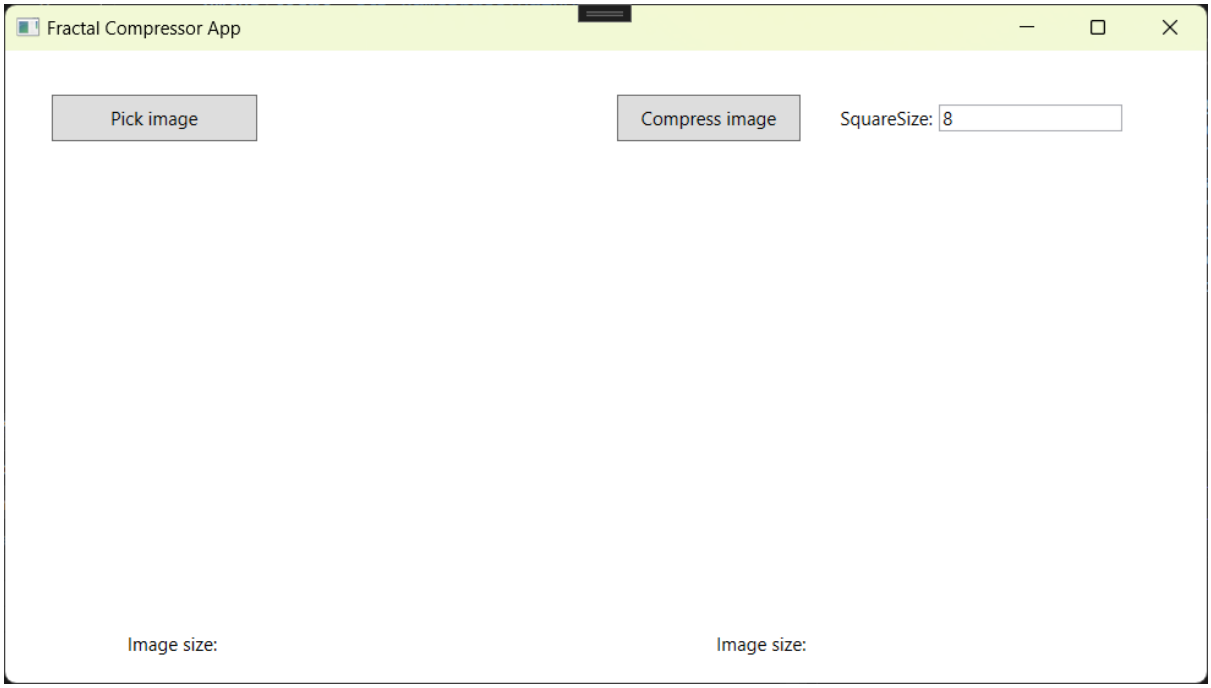

Рис. 1. Головне вікно програми.

Після вибору зображення через системний діалог воно буде відображено в інтерфейсі програми та також внизу сторінки буде вказано його розмір в байтах.

Для стиснення зображення потрібно натиснути на кнопку “Compress image”. Перед цим необхідно впевнитись в правильності розміру блоку – по замовчуванню це 8 пікселів, проте значення може бути змінено користувачем.

Програма автоматично відкриє діалог для збереження стиснених даних зображення та декодує зображення для наглядності роботи алгоритму.

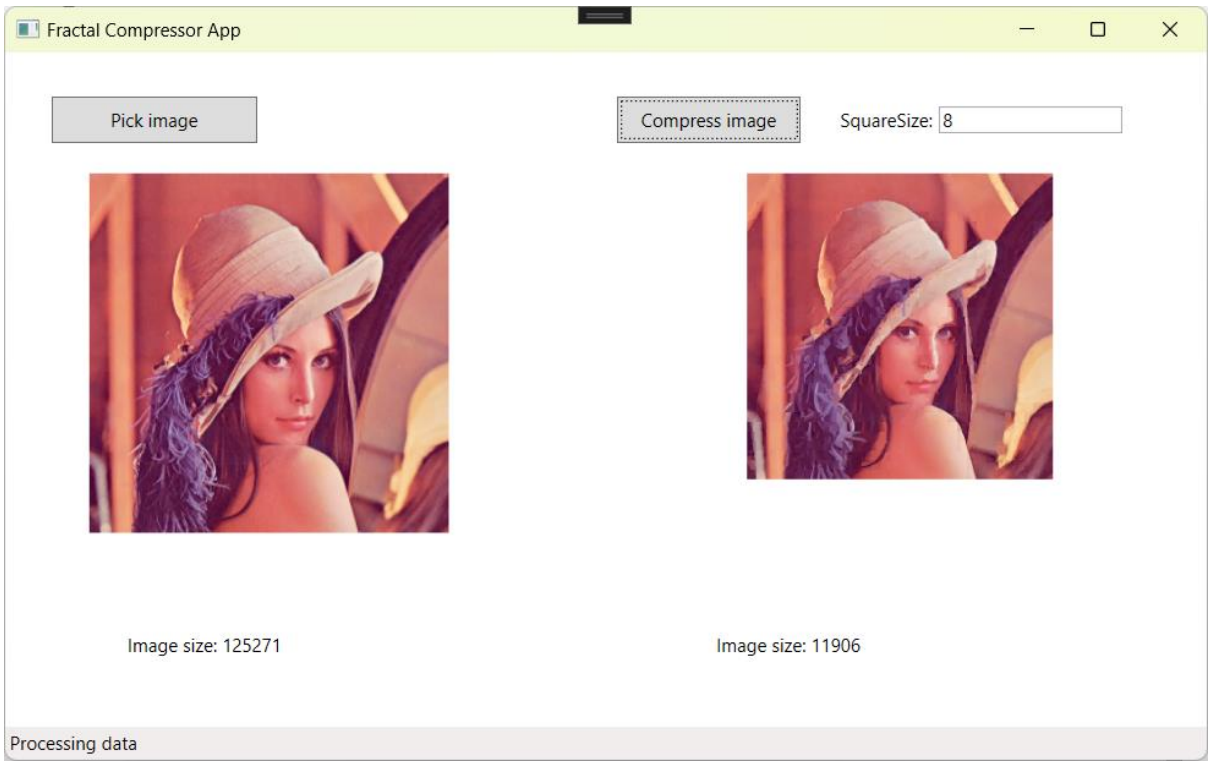

Рис. 2. Результат роботи програми

Також програма відображає поточний стан її роботи в рядку стану. Користувач може переглянути як початкове так і декодоване зображення та порівняти їх розміри.

3.6Аналіз отриманих результатів

Для тесту було використано зображення, яке представлено на рисунку вище. Було отримано такі дані в процесі роботи:

Процес кодування триває 2хв.

Процес декодування – 20 секунд.

Розмір вхідного файлу – 122кб.

Розмір файлу з даними стиснення – 12 кб.

Розмір вихідного файлу зображення– 121 кб.

Можна побачити, зо завдяки високому ступеню самоподібності зображення було збережено з використанням всього 10% пам’яті відносно розміру вхідного зображення. Це показує високу ефективність алгоритму на заданих даних.

ВИСНОВКИ

При виконанні курсової роботи мною вивчено основні принципи та засоби фрактальної графіки та написано програму яка виконує фрактальне стиснення зображення, та подальше його декодування.

Використання фракталів для стиснення інформації є потужним інструментом з рядом переваг. Фрактальне стиснення дозволяє ефективно зменшити розмір інформації, зберігаючи при цьому важливі деталі. Цей метод заснований на принципах самоподібності і повторюваності фрактальних структур у набагато меншому масштабі. Використання фракталів дозволяє досягти великого ступеня стиснення, що робить його корисним для передачі та зберігання великих обсягів даних.

Недоліками фрактального стиснення є висока обчислювальна складність процесу стиснення та декомпресії, що може вимагати значних обчислювальних ресурсів. Крім того, фрактальне стиснення не завжди ефективно для всіх типів даних і може призводити до втрати якості та точності при відновленні оригінального зображення чи даних.

Однак, за правильного використання, фрактальне стиснення може бути корисним інструментом для ефективного зберігання та передачі інформації, особливо в тих випадках, коли великі обсяги даних потребують економії простору без значної втрати якості. Для подальшого розвитку використання фракталів у стисненні інформації слід проводити додаткові дослідження для поліпшення ефективності та швидкодії алгоритмів.

СПИСОК ЛІТЕРАТУРИ

1. Качмарський Ю. Л., Черняк В. І., Яценко М. В. Фрактали: Конспект лекцій. - Київ: Видавничий дім "Слово", 2001.
2. Харченко В. Ф. Фрактальний аналіз: Методи та застосування. - Київ: Видавництво НТУУ "КПІ", 2008.
3. Кузьменко О. А., Баран О. В., Сліпченко О. І. Фракталогія. - Київ: Видавництво НТУУ "КПІ", 2010.
4. Стринадюк В. М., Кузнєцов В. І., Забіяка І. В. Фрактальний аналіз у прикладних задачах. - Київ: Наукова думка, 2006.
5. Голикова О. М., Криворучко О. П., Макаров В. Ю. Фракталы в задачах и управлении. - Харьков: ХНУ им. В. Н. Каразина, 2003.
6. Макаров В. Ю. Фрактальна математика та моделі фізичних процесів. - Харків: Видавництво Харківського національного університету, 2015.
7. Глущенко В. В. Фрактальний аналіз зображень. - Київ: Наукова думка, 2014.
8. Квітка В. О., Забіяка І. В., Кузнєцов В. І. Методи фрактального аналізу у прикладних дослідженнях. - Київ: Видавництво НТУУ "КПІ", 2002.
9. Кондратов О. В. Фрактальна геометрія і комп'ютерні графічні системи. - Київ: Видавництво НТУУ "КПІ", 2004.
10. Качмарський Ю. Л., Мироненко В. Д. Фрактальні методи у моделюванні динамічних систем. - Київ: Видавничий дім "Слово", 2004.

ДОДАТКИ

Додаток А – код алгоритму стиснення

```
public FractalCompressionResult Compress(CustomImage image)
{
    //Validate the image and the params
    if (image.Width % _options.RSquareSize > 0 || image.Height % _options.RSquareSize > 0 || _options.DSquareRatio <= 1)
        throw new InvalidOperationException("The compression options specified can't compress this image as the R squares size is not a divisor of height and width. Ri Sizes must be divisor of Di sizes too.");

    object finalResultLock = new object();
    int rXCount = image.Width / _options.RSquareSize; //Number of R Squares in X
    int rYCount = image.Height / _options.RSquareSize; //Number of R Squares in Y
    int totalRCount = rXCount * rYCount;
    byte dSquareSize = Convert.ToByte(_options.RSquareSize * _options.DSquareRatio);
    int maxDxPos = image.Width - dSquareSize + 1;
    int maxDyPos = image.Height - dSquareSize + 1;
    byte dSize = dSquareSize;
    byte rSize = _options.RSquareSize;
    int n = _options.RSquareSize * _options.RSquareSize;
    double drRatio = (double)_options.RSquareSize / (double)dSquareSize;
    int drInverseRatio = dSquareSize / _options.RSquareSize;
    double pixelAverageSquareSize = (double)drInverseRatio * (double)drInverseRatio;
    var translateMatrixes = GetTranslateMatrixes(Convert.ToSByte(rSize));
    int transformationsCount = _transformMatrixes.Length;
    var transformations = new FractalCompressionResult.FractalTransformation[totalRCount, 3]; //Transformations (2 dimension to keep transformations for each ri in each color component)
    int rx;
    var rawImgData = image.ImageData;
    var imgDataY = new byte[image.Width, image.Height];
    var imgDataI = new short[image.Width, image.Height];
    var imgDataQ = new short[image.Width, image.Height];
    byte[,][][,] tempDiY = new byte[maxDxPos, maxDyPos][][,];
    short[,][][,] tempDiI = new short[maxDxPos, maxDyPos][][,];
    short[,][][,] tempDiQ = new short[maxDxPos, maxDyPos][][,];
    int[,] sumsOfDY = new int[maxDxPos, maxDyPos];
    int[,] sumsOfDYSquared = new int[maxDxPos, maxDyPos];
    int[,] sumsOfDI = new int[maxDxPos, maxDyPos];
    int[,] sumsOfDISquared = new int[maxDxPos, maxDyPos];
    int[,] sumsOfDQ = new int[maxDxPos, maxDyPos];
    int[,] sumsOfDQSquared = new int[maxDxPos, maxDyPos];
```

```
//Fill imgData Y, I and Q
Parallel.For(0, image.Width, dx =>
{
   for (int dy = 0; dy < image.Height; dy++)
   {
      imgDataY[dx, dy] = rawImgData[dx, dy].Y;
      imgDataI[dx, dy] = rawImgData[dx, dy].I;
      imgDataQ[dx, dy] = rawImgData[dx, dy].Q;
   }
});

//Average pixels of each Di to size of Ri
Parallel.For(0, maxDxPos, dx =>
{
   int i, x, y, j, k, newX, newY, indexX = 0, indexY = 0;
   sbyte[,] tempTransformMatrix;
   sbyte[] tempTranslateMatrix;
   byte[,] tempPixelY = null;
   short[,] tempPixelI = null;
   short[,] tempPixelQ = null;
   int tempSumY = 0, tempSumI = 0, tempSumQ = 0;
   byte tempAveragePixelY = 0;
   short tempAveragePixelI = 0, tempAveragePixelQ = 0;
   byte[,] tempDY = new byte[rSize, rSize];
   short[,] tempDI = new short[rSize, rSize];
   short[,] tempDQ = new short[rSize, rSize];

   for (int dy = 0; dy < maxDyPos; dy++)
   {
      tempDiY[dx, dy] = new byte[transformationsCount][,];
      tempDiI[dx, dy] = new short[transformationsCount][,];
      tempDiQ[dx, dy] = new short[transformationsCount][,];

      for (i = 0; i < transformationsCount; i++)
      {
         tempPixelY = new byte[rSize, rSize];
         tempPixelI = new short[rSize, rSize];
         tempPixelQ = new short[rSize, rSize];

         for (x = 0; x < dSize; x += drInverseRatio) //Increment of inverse ratio
to get only parts of Di
         {
            for (y = 0; y < dSize; y += drInverseRatio)
            {
               //Get transformation
               tempTransformMatrix = _transformMatrixes[i];
               tempTranslateMatrix = translateMatrixes[i];

               //Average pixels on first transform
               if (i == 0)
```

```
{
    //Reset temp variables
    tempSumY = 0;
    tempSumI = 0;
    tempSumQ = 0;

    //Average pixels
    for (j = 0; j < drInverseRatio; j++)
    {
        for (k = 0; k < drInverseRatio; k++)
        {
            tempSumY += imgDataY[dx + x + j, dy + y + k];
            tempSumI += imgDataI[dx + x + j, dy + y + k];
            tempSumQ += imgDataQ[dx + x + j, dy + y + k];
        }
    }

    //Calculating average Pixel Components
    tempAveragePixelY = Convert.ToByte(tempSumY / pixelAverageSquareSize);
    tempAveragePixelI = Convert.ToInt16(tempSumI / pixelAverageSquareSize);
    tempAveragePixelQ = Convert.ToInt16(tempSumQ / pixelAverageSquareSize);

    //Keep reference to subsampled Di.
    tempDY[x / drInverseRatio, y / drInverseRatio] = tempAveragePixelY;
    tempDI[x / drInverseRatio, y / drInverseRatio] = tempAveragePixelI;
    tempDQ[x / drInverseRatio, y / drInverseRatio] = tempAveragePixelQ;

    //Sum of "a" and "a^2" for future RMS calculation
    sumsOfDY[dx, dy] += tempAveragePixelY;
    sumsOfDI[dx, dy] += tempAveragePixelI;
    sumsOfDQ[dx, dy] += tempAveragePixelQ;
    sumsOfDYSquared[dx, dy] += tempAveragePixelY * tempAveragePixelY;
    sumsOfDISquared[dx, dy] += tempAveragePixelI * tempAveragePixelI;
    sumsOfDQSquared[dx, dy] += tempAveragePixelQ * tempAveragePixelQ;
}

newX = tempTransformMatrix[0, 0] * (x / drInverseRatio) + tempTransformMatrix[0, 1] * (y / drInverseRatio) + tempTranslateMatrix[0];
newY = tempTransformMatrix[1, 0] * (x / drInverseRatio) + tempTransformMatrix[1, 1] * (y / drInverseRatio) + tempTranslateMatrix[1];
```

```
                    indexX = x / drInverseRatio;
                    indexY = y / drInverseRatio;

                    //Store pixel components for transformed Di (Fi)
                    tempPixelY[newX, newY] = tempDY[indexX, indexY];
                    tempPixelI[newX, newY] = tempDI[indexX, indexY];
                    tempPixelQ[newX, newY] = tempDQ[indexX, indexY];
                }
            }

            //Store pixels of Di
            tempDiY[dx, dy][i] = tempPixelY;
            tempDiI[dx, dy][i] = tempPixelI;
            tempDiQ[dx, dy][i] = tempPixelQ;
        }
    }
});

//Find the Transformations to do...
//For each Ri
for (rx = 0; rx < rXCount; rx++)
{
    //Parallel execution of for loop, so it runs faster.
    Parallel.For(0, rYCount, ry =>
    {
        //Calculate the sums of components in Ri for future RMS metric
calculation
        int sumOfRY = 0;
        int sumOfRYSquared = 0;
        int sumOfRI = 0;
        int sumOfRISquared = 0;
        int sumOfRQ = 0;
        int sumOfRQSquared = 0;
        int sumOfDAndRY;
        int sumOfDAndRI;
        int sumOfDAndRQ;
        int x, y, indexX, indexY, dx, dy, i, sYInt, oYInt, sIInt, oIInt, sQInt, oQInt,
            bestRmsYIndexX = -1, bestRmsYIndexY = -1,
bestRmsYTransformationIndex = -1,
            bestRmsIIndexX = -1, bestRmsIIndexY = -1,
bestRmsITransformationIndex = -1,
            bestRmsQIndexX = -1, bestRmsQIndexY = -1,
bestRmsQTransformationIndex = -1,
            bestOY = 0, bestSY = 0,
            bestOI = 0, bestSI = 0,
            bestOQ = 0, bestSQ = 0;
        byte[,] tempPixelY = null;
        short[,] tempPixelI = null;
        short[,] tempPixelQ = null;
        double sY, oY,
```

```
                    sI, oI,
                    sQ, oQ,
                    tempRmsY, tempRmsI, tempRmsQ,
                    tempSYDenominator, tempSIDenominator, tempSQDenominator,
                    finalRmsY = Double.MaxValue, finalRmsI = Double.MaxValue,
finalRmsQ = Double.MaxValue;

                for (x = 0; x < rSize; x++)
                {
                    for (y = 0; y < rSize; y++)
                    {
                        //Sum of "b" and of "b^2"
                        indexX = rx * rSize + x;
                        indexY = ry * rSize + y;
                        sumOfRY += imgDataY[indexX, indexY];
                        sumOfRI += imgDataI[indexX, indexY];
                        sumOfRQ += imgDataQ[indexX, indexY];
                        sumOfRYSquared += imgDataY[indexX, indexY] *
imgDataY[indexX, indexY];
                        sumOfRISquared += imgDataI[indexX, indexY] * imgDataI[indexX,
indexY];
                        sumOfRQSquared += imgDataQ[indexX, indexY] *
imgDataQ[indexX, indexY];
                    }
                }

                //For Each Di
                for (dx = 0; dx < maxDxPos; dx++)
                {
                    for (dy = 0; dy < maxDyPos; dy++)
                    {
                        tempSYDenominator = n * sumsOfDYSquared[dx, dy] -
sumsOfDY[dx, dy] * sumsOfDY[dx, dy];
                        tempSIDenominator = n * sumsOfDISquared[dx, dy] - sumsOfDI[dx,
dy] * sumsOfDI[dx, dy];
                        tempSQDenominator = n * sumsOfDQSquared[dx, dy] -
sumsOfDQ[dx, dy] * sumsOfDQ[dx, dy];

                        //For Each possible transformations
                        for (i = 0; i < transformationsCount; i++)
                        {
                            sumOfDAndRY = 0;
                            sumOfDAndRI = 0;
                            sumOfDAndRQ = 0;
                            tempPixelY = tempDiY[dx, dy][i];
                            tempPixelI = tempDiI[dx, dy][i];
                            tempPixelQ = tempDiQ[dx, dy][i];

                            //Compute sums necessary for calculating RMS metric
                            for (x = 0; x < rSize; x++)
```

```
{
   indexX = rx * rSize + x;

   for (y = 0; y < rSize; y++)
   {
      //Calculate indexX and indexY
      indexY = ry * rSize + y;

      //Sum of "ri*di"
      sumOfDAndRY += tempPixelY[x, y] * imgDataY[indexX,
indexY];
      sumOfDAndRI += tempPixelI[x, y] * imgDataI[indexX,
indexY];
      sumOfDAndRQ += tempPixelQ[x, y] * imgDataQ[indexX,
indexY];
   }
}

//Compute contrast, brightness and RMS metric
sY = CalculateS(tempSYDenominator, n, sumOfDAndRY,
sumsOfDY[dx, dy], sumOfRY, out sYInt);
sI = CalculateS(tempSIDenominator, n, sumOfDAndRI,
sumsOfDI[dx, dy], sumOfRI, out sIInt);
sQ = CalculateS(tempSQDenominator, n, sumOfDAndRQ,
sumsOfDQ[dx, dy], sumOfRQ, out sQInt);

oY = CalculateO(n, sY, sumOfRY, sumsOfDY[dx, dy],
Y_LEVELS, out oYInt);
oI = CalculateO(n, sI, sumOfRI, sumsOfDI[dx, dy], I_LEVELS,
out oIInt);
oQ = CalculateO(n, sQ, sumOfRQ, sumsOfDQ[dx, dy],
Q_LEVELS, out oQInt);

tempRmsY = Math.Sqrt((sumOfRYSquared + sY * (sY *
sumsOfDYSquared[dx, dy] - 2 * sumOfDAndRY + 2 * oY * sumsOfDY[dx, dy]) + oY * (n *
oY - 2 * sumOfRY)) / n);
tempRmsI = Math.Sqrt((sumOfRISquared + sI * (sI *
sumsOfDISquared[dx, dy] - 2 * sumOfDAndRI + 2 * oI * sumsOfDI[dx, dy]) + oI * (n * oI -
2 * sumOfRI)) / n);
tempRmsQ = Math.Sqrt((sumOfRQSquared + sQ * (sQ *
sumsOfDQSquared[dx, dy] - 2 * sumOfDAndRQ + 2 * oQ * sumsOfDQ[dx, dy]) + oQ * (n *
oQ - 2 * sumOfRQ)) / n);

if (Double.IsNaN(tempRmsY))
   throw new Exception("SQRT IS NAN!");
if (Double.IsNaN(tempRmsI))
   throw new Exception("SQRT IS NAN!");
if (Double.IsNaN(tempRmsQ))
   throw new Exception("SQRT IS NAN!");
```

```
                    //For each component, keep best transformation
                    if (!Double.IsNaN(tempRmsY) && finalRmsY > tempRmsY)
                    {
                        finalRmsY = tempRmsY;
                        bestRmsYIndexX = dx;
                        bestRmsYIndexY = dy;
                        bestRmsYTransformationIndex = i;
                        bestSY = sYInt;
                        bestOY = oYInt;
                    }

                    if (!Double.IsNaN(tempRmsI) && finalRmsI > tempRmsI)
                    {
                        finalRmsI = tempRmsI;
                        bestRmsIIndexX = dx;
                        bestRmsIIndexY = dy;
                        bestRmsITransformationIndex = i;
                        bestSI = sIInt;
                        bestOI = oIInt;
                    }

                    if (!Double.IsNaN(tempRmsQ) && finalRmsQ > tempRmsQ)
                    {
                        finalRmsQ = tempRmsQ;
                        bestRmsQIndexX = dx;
                        bestRmsQIndexY = dy;
                        bestRmsQTransformationIndex = i;
                        bestSQ = sQInt;
                        bestOQ = oQInt;
                    }
                }
            }
        }
        //Save Di retained for Ri
        transformations[ry * rXCount + rx, 0] = new
FractalCompressionResult.FractalTransformation(bestRmsYIndexX, bestRmsYIndexY,
Convert.ToByte(bestOY), Convert.ToByte(bestSY),
Convert.ToByte(bestRmsYTransformationIndex));
        transformations[ry * rXCount + rx, 1] = new
FractalCompressionResult.FractalTransformation(bestRmsIIndexX, bestRmsIIndexY,
Convert.ToByte(bestOI), Convert.ToByte(bestSI),
Convert.ToByte(bestRmsITransformationIndex));
        transformations[ry * rXCount + rx, 2] = new
FractalCompressionResult.FractalTransformation(bestRmsQIndexX, bestRmsQIndexY,
Convert.ToByte(bestOQ), Convert.ToByte(bestSQ),
Convert.ToByte(bestRmsQTransformationIndex));
    });
}
```

```
        return new FractalCompressionResult(transformations, Convert.ToByte(Math.Ceiling(Math.Log(Math.Max(maxDxPos, maxDyPos), 2))), Convert.ToInt16(rXCount));
    }
```

Додаток В - код алгоритму декодування зображення

```
public CustomImage Uncompress(FractalCompressionResult compressionResult, CustomImage decodingImage)
    {
        int rXCount = compressionResult.RCountX;
        int rYCount = compressionResult.Transformations.GetLength(0) / compressionResult.RCountX;
        int imgSizeX = rXCount * _options.RSquareSize;
        int imgSizeY = rYCount * _options.RSquareSize;
        Pixel[,] pixels = new Pixel[imgSizeX, imgSizeY];
        int rSize = _options.RSquareSize;

        const int dimensions = 3;

        byte dSquareSize = Convert.ToByte(_options.RSquareSize * _options.DSquareRatio);
        double drRatio = (double)_options.RSquareSize / (double)dSquareSize;
        int drInverseRatio = _options.DSquareRatio;
        int maxDxPos = imgSizeX - dSquareSize + 1;
        int maxDyPos = imgSizeY - dSquareSize + 1;
        int dSize = dSquareSize;
        var imgData = decodingImage.ImageData;
        int pixelAverageSquareSize = drInverseRatio * drInverseRatio;
        var translateMatrixes = GetTranslateMatrixes(Convert.ToSByte(dSize));

        Parallel.For(0, rXCount, rx =>
        {
            int dimIndex, indexX, indexY, newX, newY, i, j, y;
            sbyte[,] tempTransform;
            sbyte[] tempTranslate;
            double[,,] tempComponents = new double[dSize, dSize, dimensions];
            double s, o;
            FractalCompressionResult.FractalTransformation transformation;
            double componentSumY;
            double componentSumI;
            double componentSumQ;

            for (int ry = 0; ry < rYCount; ry++)
            {
                //Get transformed Di (Ri)
                for (indexX = 0; indexX < dSize; indexX++)
                {
                    for (indexY = 0; indexY < dSize; indexY++)
                    {
                        for (dimIndex = 0; dimIndex < dimensions; dimIndex++)
                        {
                            transformation = compressionResult.Transformations[ry * rXCount + rx, dimIndex];
                            tempTransform = _transformMatrixes[transformation.TransformationType];
                            tempTranslate = translateMatrixes[transformation.TransformationType];

                            newX = Convert.ToInt32(tempTransform[0, 0] * indexX + tempTransform[0, 1] * indexY + tempTranslate[0]);
                            newY = Convert.ToInt32(tempTransform[1, 0] * indexX + tempTransform[1, 1] * indexY + tempTranslate[1]);
```

```
                    s = transformation.ContrastTransformation / (double)(S_BITS_VALUE) * (2d * S_MAX) -
S_MAX;

                    if (dimIndex == 0)
                    {
                        o = (transformation.BrightnessTransformation / (double)(O_BITS_VALUE - 1) * ((1d +
Math.Abs(s)) * Y_LEVELS));

                        if (s > 0d)
                            o -= s * Y_LEVELS;

                        tempComponents[newX, newY, dimIndex] = imgData[transformation.DiX + indexX,
transformation.DiY + indexY].Y * s + o;
                    }
                    else if (dimIndex == 1)
                    {
                        o = (transformation.BrightnessTransformation / (double)(O_BITS_VALUE - 1) * ((1d +
Math.Abs(s)) * I_LEVELS));

                        if (s > 0d)
                            o -= s * I_LEVELS;

                        tempComponents[newX, newY, dimIndex] = imgData[transformation.DiX + indexX,
transformation.DiY + indexY].I * s + o;
                    }
                    else
                    {
                        o = (transformation.BrightnessTransformation / (double)(O_BITS_VALUE - 1) * ((1d +
Math.Abs(s)) * Q_LEVELS));

                        if (s > 0d)
                            o -= s * Q_LEVELS;

                        tempComponents[newX, newY, dimIndex] = imgData[transformation.DiX + indexX,
transformation.DiY + indexY].Q * s + o;
                    }
                }
            }
        }

        //Populate pixels from Ri calculated
        for (indexX = 0; indexX < dSize; indexX += drInverseRatio)
        {
            for (indexY = 0; indexY < dSize; indexY += drInverseRatio)
            {
                componentSumY = 0;
                componentSumI = 0;
                componentSumQ = 0;

                for (i = 0; i < drInverseRatio; i++)
                {
                    for (j = 0; j < drInverseRatio; j++)
                    {
                        componentSumY += tempComponents[indexX + i, indexY + j, 0];
                        componentSumI += tempComponents[indexX + i, indexY + j, 1];
                        componentSumQ += tempComponents[indexX + i, indexY + j, 2];
                    }
                }
```

```
                    pixels[rx * rSize + indexX / drInverseRatio, ry * rSize + indexY / drInverseRatio] =
Pixel.FromYIQ(
                        Convert.ToByte(Math.Max(Math.Min(componentSumY / (drInverseRatio *
drInverseRatio), 255), 0)),
                        Convert.ToInt16(Math.Max(Math.Min(componentSumI / (drInverseRatio * drInverseRatio),
151), -151)),
                        Convert.ToInt16(Math.Max(Math.Min(componentSumQ / (drInverseRatio *
drInverseRatio), 133), -133)));
                    }
                }
            }
        });

        if (_options.SmoothImage)
            pixels = SmoothImage(pixels, rSize);

        return new CustomImage(pixels);
            }
```