Міністерство освіти і науки України
Відкритий міжнародний університет розвитку людини «Україна»

Кваліфікаційна наукова праця
на правах рукопису

ЗУБКО РОМАН АНАТОЛІЙОВИЧ

УДК 004.627

ДИСЕРТАЦІЯ

Удосконалення методів підвищення часової ефективності фрактального стиснення зображень

01.05.03 – «Математичне та програмне забезпечення обчислювальних машин і систем»

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.
_____ Р.А. Зубко

Науковий керівник Забара Станіслав Сергійович, доктор технічних наук, професор

Київ – 2021

АНОТАЦІЯ

Зубко Р.А. Удосконалення методів підвищення часової ефективності фрактального стиснення зображень. - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 01.05.03 "Математичне та програмне забезпечення обчислювальних машин і систем" – Відкритий міжнародний університет розвитку людини "Україна", Київ, 2021.

Актуальність дослідження обумовлена високим практичним значенням і недостатнім опрацюванням проблеми застосування фрактальних методів кодування зображень в інформаційних технологіях та комп'ютерній графіці. Особлива увага приділяється математичній моделі, системі ітерованих функцій (Iterated Function System - IFS), яка використовується при фрактальному стисненні зображень.

Досліджено фрактальний метод стиснення зображень. Серед різноманітних методів кодування він дозволяє отримувати найбільші коефіцієнти стиснення. Наведено класичний алгоритм кодування-декодування зображень фрактальним методом. Суть його полягає в пошуку самоподібних ділянок зображення на підставі параметрів стиснення. Процес кодування цим методом вимагає значних обчислювальних витрат, відповідно швидкодія його невисока. Потрібно відмітити, що декодування зображень не потребує таких великих потужностей та ресурсів операційної системи. Низька швидкодія стиснення пов’язана з тим, що для отримання високої якості вихідного зображення необхідно обробляти велику кількість доменних областей. Тому виконано дослідження з пошуку критеріїв, що дозволили б для даної рангової області підібрати відповідну доменну, яка після афінних перетворень найбільш точно апроксимує рангову область.

Проведено стислий аналіз варіантів підвищення швидкодії побудови систем ітеруючих функцій фрактального кодування зображень, їх

ефективності та можливості практичного застосування. Розглядаються теоретичні аспекти підвищення ефективності стиснення зображень фрактальним методом. Пропонуються варіанти суттєвого підвищення швидкості фрактального стиснення. Основні зусилля були спрямовані на вивчення, аналіз і подальший розвиток підходів, що використовуються для стиснення зображень та дозволяють зменшити кількість надлишкових даних.

Запропоновано варіант модифікованого методу, який дозволить суттєво підвищити швидкодію фрактального кодування. Описана послідовність кроків, необхідних для декодування зображення. Для перевірки методу розроблено його програмну реалізацію та експериментально доведено її ефективність на стандартних зображеннях.

Фрактальна компресія - це пошук самоподібних областей в зображенні і визначення для них параметрів афінних перетворень. У гіршому разі, якщо не застосовуватиметься оптимізуючий алгоритм, буде потрібно перебір і порівняння усіх можливих фрагментів зображення різного розміру. Навіть для невеликих зображень при урахуванні дискретності ми отримаємо астрономічне число варіантів, що перебираються. Різке звуження класів перетворень, наприклад, за рахунок масштабування тільки на певне число разів, не дозволить добитися прийнятного часу. Крім того, при цьому втрачається якість зображення. Переважна більшість досліджень в області фрактальної компресії зараз спрямовані на зменшення часу архівації, необхідного для отримання якісного зображення.

Для фрактального алгоритму компресії, як і для інших алгоритмів стиснення з втратами, дуже важливі механізми, за допомогою яких можна буде регулювати міру стиснення і втрат. До теперішнього часу розроблений досить великий набір таких методів. По-перше, можна обмежити кількість перетворень, свідомо забезпечивши міру стиснення не нижче фіксованої величини. По-друге, можна поставити вимогу, щоб за ситуації, коли різниця між оброблюваним фрагментом і найкращим його наближенням буде вищою певного граничного значення, цей фрагмент дробився обов'язково (для нього

заводиться декілька лінз). По-третє, можна заборонити дробити фрагменти розміром менше, наприклад, чотирьох точок. Змінюючи порогові значення і пріоритет цих умов, можна дуже гнучко управляти коефіцієнтом компресії зображення: від побітової відповідності, до будь-якої міри стиснення.

Процес кодування вимагає дуже великого об'єму обчислень. Для пошуку фрактальних малюнків в зображенні потрібні мільйони, навіть мільярди ітерацій. Залежно від розподільної здатності і змісту вхідних растрових даних, якості зображення, часу стиснення і розміру файлу процес стиснення одного зображення може зайняти від декількох секунд до декількох годин (і більше) навіть на дуже швидкодіючому комп'ютері.

Декодування фрактального зображення – процес набагато простіший, оскільки уся трудомістка робота була виконана при пошуку усіх фракталів під час кодування. В процесі декодування треба лише інтерпретувати фрактальні коди, перетворивши їх в растрове зображення.

Фрактальний метод стиснення зображень відноситься до групи методів стиснення з втратами інформації. Це означає що, у разі його застосування відновлене зображення буде відрізняється від початкового на величину шуму перетворення, хоча ця відмінність може бути дуже малою. Процес порівняння фракталів не передбачає пошуку точної їх відповідності. Замість цього шукається "найкраща" відповідність на підставі параметрів стиснення (часу кодування, якості та розміру вихідного зображення). Процесом кодування, проте, можна керувати, доводячи його до стану, в якому зображення візуально не має втрат. Тобто, користувач не повинен бачити погіршення зображення.

Ключові слова: фрактальне кодування, стиснення зображень, комп'ютерна графіка

ABSTRACT

Zubko R.A. Improving methods to increase the time efficiency of fractal image compression. - Manuscript

The dissertation on competition of a scientific degree of the candidate of technical sciences on a specialty 01.05.03 - "Mathematical and software of computers and systems". - Open International University of Human Development "Ukraine", Kyiv, 2021.

The relevance of the study is due to the high practical value and insufficient elaboration of the problem of application of fractal methods of image coding in information technology and computer graphics. Particular attention is paid to the mathematical model, the Iterated Function System (IFS), which is used in fractal image compression.

The fractal method of image compression is investigated. Among the various coding methods, it allows to obtain the highest compression ratios. The classical algorithm of coding-decoding of images by a fractal method is resulted. Its essence is to find self-similar areas of the image based on the compression parameters. The coding process by this method requires significant computational costs, respectively, its speed is low. It should be noted that image decoding does not require such large power and resources of the operating system. The low compression speed is due to the fact that a large number of domain domains need to be processed to obtain high quality original image. Therefore, a study was performed to find criteria that would allow for a given rank area to select the appropriate domain, which after affine transformations most accurately approximates the rank range.

A brief analysis of options for improving the speed of construction of systems of iterative functions of fractal image coding, their efficiency and feasibility of practical application. Theoretical aspects of increasing the efficiency of image compression by the fractal method are considered. Options for significantly increasing the rate of fractal compression are offered. The main

efforts were focused on the study, analysis and further development of approaches used for image compression and to reduce the amount of redundant data.

A variant of the modified method is proposed, which will significantly increase the speed of fractal coding. Describes the sequence of steps required to decode the image. To test the method, its software implementation was developed and its effectiveness on standard images was experimentally proved.

Fractal compression is the search for self-similar areas in the image and determining the parameters of affine transformations for them. In the worst case, if you do not use the optimization algorithm, you will need to search and compare all possible fragments of the image of different sizes. Even for small images, given the discreteness, we get an astronomical number of options that are moved. A sharp narrowing of the transformation classes, for example, by scaling only a certain number of times, will not achieve an acceptable time. In addition, the image quality is lost. The vast majority of research in the field of fractal compression is now aimed at reducing the archiving time required to obtain a quality image.

For the fractal compression algorithm, as for other lossy compression algorithms, the mechanisms by which the degree of compression and loss can be adjusted are very important. To date, a fairly large set of such methods has been developed. First, you can limit the number of transformations, deliberately providing a degree of compression not lower than a fixed value. Secondly, it is possible to require that in situations where the difference between the processed fragment and its best approximation is higher than a certain limit value, this fragment must be crushed (it gets several lenses). Third, it is possible to prohibit the fragmentation of fragments smaller than, for example, four points. By changing the thresholds and the priority of these conditions, you can very flexibly control the compression ratio of the image: from bitwise matching, to any degree of compression.

The coding process requires a very large amount of computation. It takes millions, even billions of iterations to find fractal images in an image. Depending on the resolution and content of the raster input, image quality, compression time,

and file size, the process of compressing a single image can take from a few seconds to several hours (or more) even on a very fast computer.

Decoding a fractal image is a much simpler process, as all the time-consuming work was done searching for all fractals during encoding. In the process of decoding, you only need to interpret fractal codes, turning them into a bitmap image.

Fractal method of image compression belongs to the group of methods of compression with information loss. This means that, if applied, the recovered image will differ from the original by the amount of conversion noise, although this difference may be very small. The process of comparing fractals does not involve finding their exact match. Instead, the "best" match is sought based on the compression parameters (encoding time, quality, and size of the original image). The encoding process, however, can be controlled by bringing it to a state in which the image is visually lossless. That is, the user should not see image degradation.

Keywords: fractal coding, image compression, computer graphics

Список публікацій здобувача

Статті у наукових виданнях, включених до переліку наукових фахових видань України, затвердженого ДАК МОН України

1. Зубко Р. А. Фрактали. Східно-Європейський журнал передових технологій. 2012. Т. 6, № 4(60). С. 13-14.

2. Зубко Р. А. Алгоритми стиснення зображень. Східно-Європейський журнал передових технологій. 2013. Т. 1, № 2(61). С. 40-44.

3. Зубко Р. А. Стиснення зображень фрактальним методом. Східно-Європейський журнал передових технологій. 2014. Т. 6, № 2(72). С. 23-28.

4. Зубко Р. А. Методи оптимізації фрактального стиснення зображень. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2015. №1 (17). С. 167-176.

5. Зубко Р. А. Покращення ефективності стиснення зображень фрактальним методом. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2017. №1 (18). С. 190-196.

6. Зубко Р. А. Варіанти підвищення швидкодії стиснення зображень фрактальним методом. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2018. №2 (21/2). С. 128-135.

7. Зубко Р. А. Дослідження можливості покращення часової ефективності фрактального стиснення зображень. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2019. № 1 (22). С. 256-266.

Статті в наукових виданнях Європейського Союзу

8. Roman Zubko. Improving the efficiency of fractal image compression by the criterion of detail of rank and domain blocks. «Danish Scientific Journal» (DSJ) Kobenhavn. Denmark. 2021. Vol. № 45 (2). Pp. 38-43. ISSN 3375-2389.

Матеріали і тези доповідей на конференціях

9. Зубко Р. А. Фрактальний метод стиснення зображень Матеріали VII Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 1-3 листопада 2012. С. 95-97.

10. Зубко Р. А. Алгоритм швидкого фрактального стиснення цифрових зображень. Матеріали VIII Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 7-9 листопада 2013. С. 72-73.

11. Зубко Р. А. Стиснення зображень фрактальним методом. Матеріали XII Всеукраїнської наукової конференції студентів і молодих вчених «Молодь: освіта, наука, духовність», Україна, Київ, 21-22 квітня 2015. С. 250-252.

12. Зубко Р. А. Методи оптимізації фрактального стиснення зображень. Матеріали IX Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 27-28 листопада 2015. С. 41-42.

13. Зубко Р. А. Покращення часової ефективності базового методу фрактального стиснення зображень. Матеріали XIII Всеукраїнської наукової конференції студентів і молодих вчених «Молодь: освіта, наука, духовність», Україна, Київ, 12-14 квітня 2016. С. 224-225.

14. Зубко Р. А. Табличний метод підвищення швидкодії фрактального стиснення зображень. Матеріали X Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 30-31 березня 2017. С. 86-88.

15. Зубко Р. А. Покращення ефективності стиснення зображень фрактальним методом з використанням табличного пошуку. Матеріали XI Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 23-24 березня 2018. С. 23-27.

16. Зубко Р. А. Дослідження можливості покращення часової ефективності фрактального стиснення зображень. Матеріали XVII Всеукраїнської наукової конференції студентів і молодих вчених «Молодь: освіта, наука, духовність», Україна, Київ, 27-28 травня 2020. Ч. 3. С. 401-403.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Можливість стиснення зображень зі значними втратами обумовлена особливостями сприйняття інформації органами чуття людини. Створення нових носіїв даних та поширення глобальних комп'ютерних мереж, викликали швидке збільшення об'ємів інформації. Пов'язано це з тим, що для зберігання зображень представлених в цифровій формі, необхідний досить великий об'єм пам'яті, а при передачі їх по каналах зв'язку витрачається значний час.

Частина даних, що транспортуються, припадає саме на статичні растрові зображення. Відповідно для економії пам’яті та більш ефективного використання ресурсів системи, створюють спеціальні методи кодування. Зображення - це особливий вид даних, який має надлишковість, що дає додаткові можливості для стиснення. Завдяки цьому такі методи кодування зображень можуть мати дуже високі коефіцієнти компресії.

Одним із перспективних методів кодування є стиснення зображень з використанням фрактальної графіки. Фрактальне стиснення - це математичний процес для кодування растрів, які містять реальне зображення, в сукупність математичних даних, що описують фрактальні властивості зображення. Цей вид кодування заснований на тому, що усі природні і більшість штучних об'єктів містять надмірну інформацію у вигляді подібних фрагментів зображення, що повторюються. Вони отримали назву фракталів.

Фрактальне кодування широко використовується для перетворення растрових зображень у фрактальні коди. Фрактальне декодування є зворотним процесом, в якому система фрактальних кодів перетворюється в растр.

Початок у розробці фрактального методу поклали дослідження Майкла Барнслі у 1988 році. Він відкрив клас теорем, що дозволили ефективно стискати зображення. Барнслі назвав метод «фрактальним перетворенням». При цьому кодується по суті не саме зображення, а алгоритм його побудови. Метод базується на певній особливості реальних зображень, яка полягає в

тому, що в них з невеликими варіаціями багаторазово повторюються окремі самоподібні фрагменти. Наприклад, листя в кроні дерева, вікна в будівлі, пішоходи на вулиці, луска на рибі, межі між темними і світлими ділянками зображення. І хоча ці фрагменти відрізняються між собою в деталях, проте, в них багато спільного. Це був практично перший алгоритм, застосований для математичного опису реального растрового зображення через його фрактальні властивості.

Основа базового методу фрактального стиснення - це виявлення подібних ділянок в зображенні. Математична модель, яка використовується при фрактальному стисненні зображень, називається системою ітерованих функцій (Iterated Function System - IFS). Вона представлена Майклом Барнслі і Аланом Слоуном на початку 80-х рр. минулого століття. Одна з можливих схем кодування зображень фрактальним методом запропонована Арно Жакеном. Цей підхід був вдосконалений Ювалом Фішером і рядом інших дослідників і став основою для більшості методів фрактального кодування.

На ступінь фрактального стиснення помітний вплив чинять вміст і розподільна здатність початкового растру. Зображення з високою розподільною здатністю та з великою кількістю фрактальних елементів краще стискаються, чим зображення з низьким вмістом таких елементів.

Задачами підвищення ефективності фрактального стиснення зображень займаються Д. Ватолін, С. Ільюшин, А. Шарабайко. Методи, які дозволяють зменшити обчислювальну складність фрактального кодування добре описані в публікаціях Ю. Фішера, С. Уестіда, В. П. Майданюка.

Але, незважаючи на те, що виконані в останні роки роботи дозволили значно зменшити час кодування, актуальним залишається проведення досліджень, які дозволять суттєво підвищити швидкіть стиснення зображень цим методом.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертацію виконано відповідно до плану науково-дослідної роботи кафедри інформаційних технологій та програмування Відкритого міжнародного університету розвитку людини «Україна» за темою «Дослідження алгоритмів стиснення даних», реєстраційний номер 0113U004530 (дата створення 2013 р.) та темою «Методи стиснення зображень» (2016-2020 р.).

Мета та завдання дослідження. Метою роботи є підвищення швидкості фрактального стиснення статичних растрових зображень за рахунок попереднього відбору доменних блоків для кожного рангового блоку.

Основними задачами дослідження є:

– огляд переваг і недоліків та аналіз використання базових методів фрактального кодування зображень;

– дослідження сучасного стану існуючих, на даний момент, підходів до підвищення часової ефективності фрактального стиснення зображень;

– розробка методу зменшення обчислювальної складності та скорочення часових витрат окремих етапів фрактального стиснення півтонових та кольорових зображень;

– програмна реалізація та розробка засобів для експериментального дослідження та аналізу часової ефективності модифікованого методу фрактального стиснення зображень.

Об’єкт дослідження – процес стиснення статичних растрових півтонових та кольорових зображень фрактальним методом.

Предмет дослідження – методи та засоби підвищення часової ефективності фрактального стиснення статичних растрових зображень.

Методи дослідження. У процесі дослідження застосовувалися: методи дискретної математики, теорія чисел та чисельних методів, теорія алгоритмів, теорія систем ітерованих функцій, основні поняття

математичного аналізу та теорії множин, методи цифрової обробки зображень, теорії фракталів.

Наукова новизна отриманих результатів:

1. Вперше запропоновано швидкодіючий метод фрактального стиснення статичних растрових кольорових зображень, особливість якого полягає у зменшенні простору пошуку доменних блоків за рахунок попереднього відбору найбільш близьких доменних блоків до заданого рангового блоку, в результаті чого швидкість стиснення зросла більш як у 25 разів, у порівнянні з базовим методом.

2. Подальшого розвитку отримав метод швидкісного фрактального стиснення півтонових зображень, особливістю якого є попередня обробка кожного доменного і рангового блоку оператором виділення контурів зображення і оцінка детальності кожного з цих блоків, що дозволило підвищити швидкість стиснення в середньому у 10-20 разів у порівнянні з повним перебором.

3. Розроблено нову модифікацію табличного методу побудови IFS-функцій, відмінністю якої є значне зменшення обчислювальної складності окремих етапів фрактального стиснення, що відповідно підвищує швидкість кодування.

4. Запропоновано нову схему оцінки детальності доменного і рангового блоків, яка відрізняється від відомої використанням модулів відстані від середнього значення, що дозволяє значно зменшити обчислювальні витрати при стисненні.

Результати, що виносяться на захист:

1. Розроблено новий метод зменшення обчислювальної складності та підвищення часової ефективності фрактального стиснення зображень, який точно ідентифікує контури, що дозволяє забезпечити менші розміри таблиць пошуку.

2. Експериментально показано, що при використанні авторського методу для фрактального стиснення істотно підвищується швидкість кодування-декодування зображень.

Практичне значення отриманих результатів полягає в тому, що на основі проведених теоретичних досліджень і отриманих наукових результатів розроблено програмні засоби для фрактального стиснення зображень, що підвищує ефективність використання комп’ютерних систем в цілому.

Результати дисертаційного дослідження впроваджені та пройшли апробацію у ПрАТ "Київський електровагоноремонтний завод" та Інституті розробки інформаційних систем (ТОВ «ІРІС») при підготовці цифрових зображень для збереження на носіях інформації.

Особистий внесок здобувача. Усі наукові результати, викладені у дисертаційній роботі, отримані автором особисто.

Достовірність отриманих результатів підтверджується експериментальними даними, які отримані в ході дослідження, а також результатами апробації модифікованого методу підвищення часової ефективності фрактального стиснення зображень.

Апробація матеріалів дисертації. Основні положення і результати дисертаційного дослідження доповідалися на VII - XI Всеукраїнських науково-практичних конференціх «Комп'ютерні технології: наука і освіта» (м. Київ, 2012-2018 рр.), XII - XIII, XVII Всеукраїнських наукових конференціях студентів і молодих вчених «Молодь: освіта, наука, духовність» (м. Київ, 2015-2016 рр., 2020 р.).

Публікації. Основні результати дисертаційного дослідження опубліковано у 16 наукових працях, зокрема: 7 статей у фахових виданнях України, 1 стаття у періодичному науковому виданні в країні Європейского Союзу, 8 публікацій у збірниках матеріалів тез доповідей на наукових конференціях.

Структура та обсяг дисертації. Дисертація складається з вступу, 4 розділів, висновку та додатку. Вона містить 149 сторінок машинописного тексту, включаючи 30 рисунків, 5 таблиць, список використаних джерел зі 101 найменуваня.

РОЗДІЛ 1. Математична модель стиснення зображень фрактальним методом

1.1 Основи кодування зображень

Найбільший об'єм інформації про навколишній світ людина отримує за допомогою зору. Зображення є найбільш універсальною формою представлення інформації. Вони використовуються при вирішенні різних задач в таких галузях як дослідження природних ресурсів, картографії, медицині, біології та ін. Не можливо представити собі життя без фотографії, кіно, телебачення, де зображення є основним джерелом інформації.

Слід відзначити, що все більше поширення набувають цифрові методи представлення відеоінформації оскільки вони забезпечують найбільшу достовірність при передачі, обробці та зберіганні.

Однак, застосування цифрових методів стикається з деякими труднощами, основна з яких це великі об'єми інформації, що підлягають збереженню, передачі та перетворенню. Так, часто необхідно зберігати на жорсткому диску окремі кадри або повний відеоряд. Також, все більшу популярність завойовують компактні флеш накопичувачі з відеозаписом. В даних випадках цифрова відеоінформація записується на носії з стисканням по певному стандарту.

Сучасні комп'ютери дозволяють з високою якістю вводити, виводити, обробляти зображення різного призначення. Важливий клас представляють зображення натурального походження, призначені для візуального спостереження (фотографії, телевізійні кадри, кадри кінофільмів та ін.). Незважаючи на постійний зріст ємкості запам'ятовуючих пристроїв, зростання об'ємів візуальної інформації, що використовується в комп'ютерних системах, значно його випереджає, що робить ще більш актуальнішим вирішення задачі стиснення (кодування) зображень. Метою кодування зображень являється зменшення числа біт, необхідних для

зберігання або передачі зображень. Кодування виконується тим ефективніше, чим менше об'єм цифрового сигнала, що описує зображення з заданою точністю. Об'єм цифрового сигнала характеризується кількістю біт на елемент (відлік) зображення. Вважається, що для високоякісного відтворення зображення, необхідно, щоб на 1 елемент зображення припадало не менше 8 біт [1]. За допомогою спеціальних методів кодування вдається зменшити цю величину в кілька разів і забезпечити прийнятну якість відтворення зображень з цифрового сигналу об'ємом до 1 біта на елемент зображення. Відповідно, при цьому зменшується необхідна ємкість запам'ятовуючих пристроїв для зберігання зображень і необхідна пропускна здатність каналу для передачі зображень.

Відомо, що ефективне кодування звичайно виконується в три етапи:

1. На початковому етапі знаходять представлення сигналу зображення, наприклад у вигляді деякого набору коефіцієнтів перетворення. Така операція, як правило, зворотня.

2. На другому етапі знижується точність представлення, але так, щоб виконувались задані вимоги до якості зображення. Така операція не зворотня.

3. На третьому етапі ліквідується статистична надлишковість, наприклад за допомогою кода Хаффмена. Ця операція зворотня.

При виконання пп. 1, 2 найбільш широке застосування знаходять методи кодування на основі двомірних ортогональних перетворень [2, 3]. Зокрема, стандарти JPEG та MPEG [4, 5] передбачають застосування дискретного косинусного перетворення (ДКП), яке потребує меншу кількість арифметичних операцій в порівнянні із звичайним перетворенням Фурьє. Однак, навіть в цьому випадку потрібні надзвичайно великі обчислювальні потужності [5]. Це свідчить про те, що необхідне детальне дослідження інших методів стиснення, які дозволяли б вирішувати задачі кодуванні зображень при менших обчислювальних і, відповідно, апаратурних витратах.

Очевидно, що досягнення великих коефіцієнтів стиснення можливе тільки при використанні якогось радикально-нового методу кодування. Слід

також зазначити, що при розробці ефективних алгоритмів стиснення зображень необхідно, в першу чергу, враховувати властивості самих зображень та особливості їх сприйняття отримувачем повідомлення, в даному випадку особливості сприйняття зображень зоровим аналізатором людини. Високих коефіцієнтів стиснення можливо очікувати, якщо спосіб кодування узгоджується із зоровою системою людини та імітує її функції [6].

Таким чином, тема кодування зображень є актуальною і потребує пошуку нових методів кодування.

Одним з нових методів кодування зображень є фрактальний метод. Але базовий алгоритм, що реалізує цей метод потребує великих обчислювальних потужностей та працює доволі повільно.

1.2 Характеристики расторових зображень

Растр - це матриця комірок (пікселів). Будь-який піксель (pixel - Picture Element) має свій колір. Сукупність пікселів різного кольору утворює зображення. Залежно від розташування пікселів в просторі розрізняють квадратний, прямокутний, гексагональний або інші типи растру. Для опису розташування пікселів використовують різноманітні системи координат. Загальним для усіх таких систем є те, що координати пікселів утворюють дискретний ряд значень (необов'язково цілі числа). Часто використовується система цілих координат - номерів пікселів з (0, 0) в лівому верхньому кутку. Таку систему ми використовуватимемо і надалі, бо вона зручна для розгляду алгоритмів графічного виводу.

Розмір растру зазвичай вимірюється кількістю пікселів по горизонталі і вертикалі. Роздільна здатність характеризує відстань між сусідніми пікселами - крок дискретної сітки растру. Роздільну здатність вимірюють кількістю пікселів на одиницю довжини. Найбільш популярна одиниця виміру - dpi (dots per inch) - кількість пікселів в одному дюймі довжини (2,54

см). Не слід ототожнювати крок з розмірами пікселів - розмір пікселів може дорівнювати кроку, а може бути як менше, так і більше кроку.

Можна сказати, що для комп’ютерної графіки найбільш зручний растр з однаковим кроком для обох осей, тобто dpi X= dpi Y. Це зручно для багатьох алгоритмів виведення графічних об'єктів.

Форма пікселів растру визначається особливостями пристрою графічного виводу. Наприклад, піксели можуть мати форму прямокутника або квадрата, які по розмірах дорівнюють кроку растру (дисплей на рідких кристалах); піксели можуть мати круглу форму і по розмірах можуть не дорівнювати кроку растру.

Кількість кольорів (глибина кольору) - важлива характеристика будь-якого зображення, не лише растрового. Відповідно до психофізіологічних досліджень, око людини здатне розрізняти 350 000 кольорів [1, 36-43].

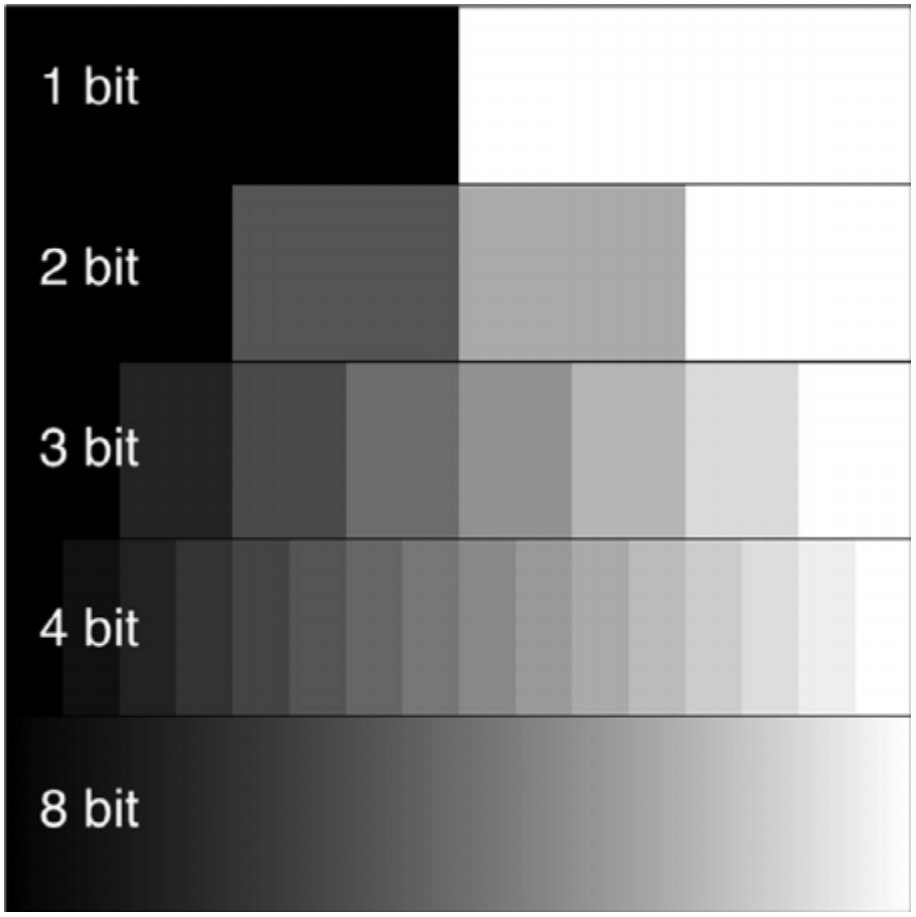


Рис. 1.1 – Градації сірого (1-8 бітів на піксель)

Класифікація зображень:

- Двоколірні (бінарні) - 1 біт на піксель. Серед двоколірних найчастіше зустрічаються чорно-білі зображення.

- Півтонові - градації сірого або іншого кольору. Наприклад, 256 градацій (8 біт на піксель) (рис. 1.1).

- Кольорові зображення (2 біта на піксел і більше). Глибина кольору 16 бітів на піксель (65536 кольорів) отримала назву High Color, 24 біта на піксел (16,7 млн. кольорів) - True Color. У комп'ютерних графічних системах використовують і велику глибину кольору - 32, 48 і більше бітів на піксел (рис. 1.2-1.3).

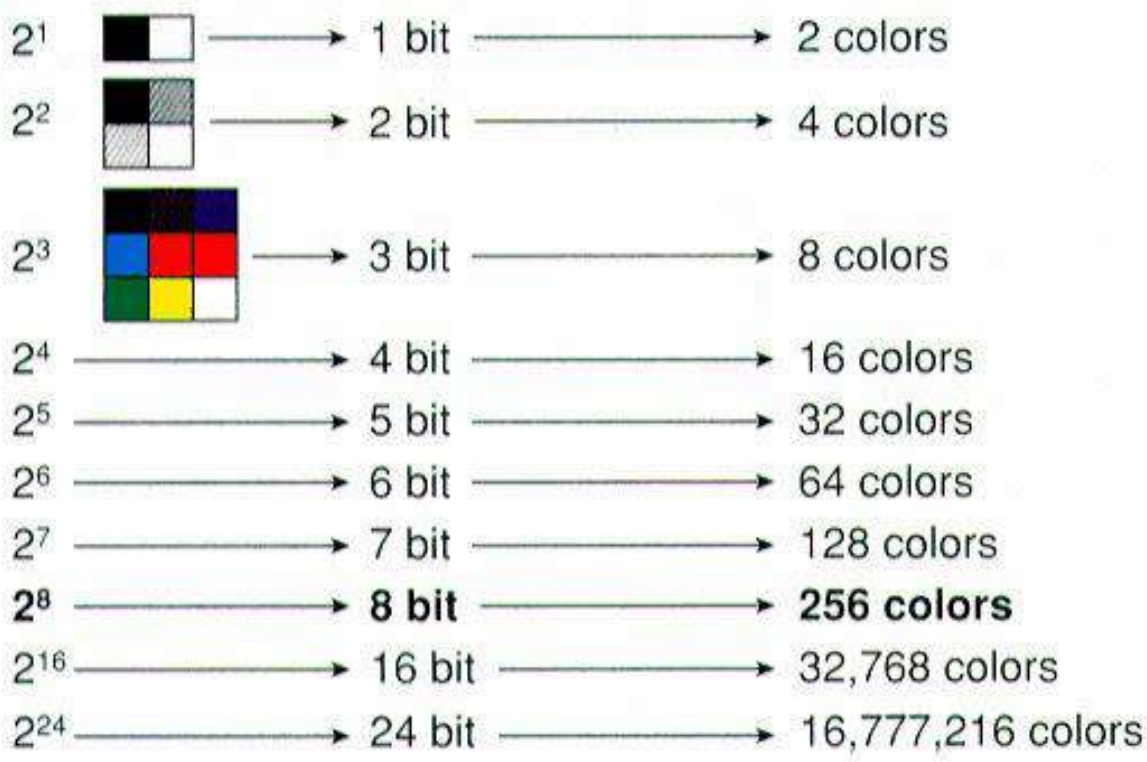


Рис. 1.2 – Класифікація зображень

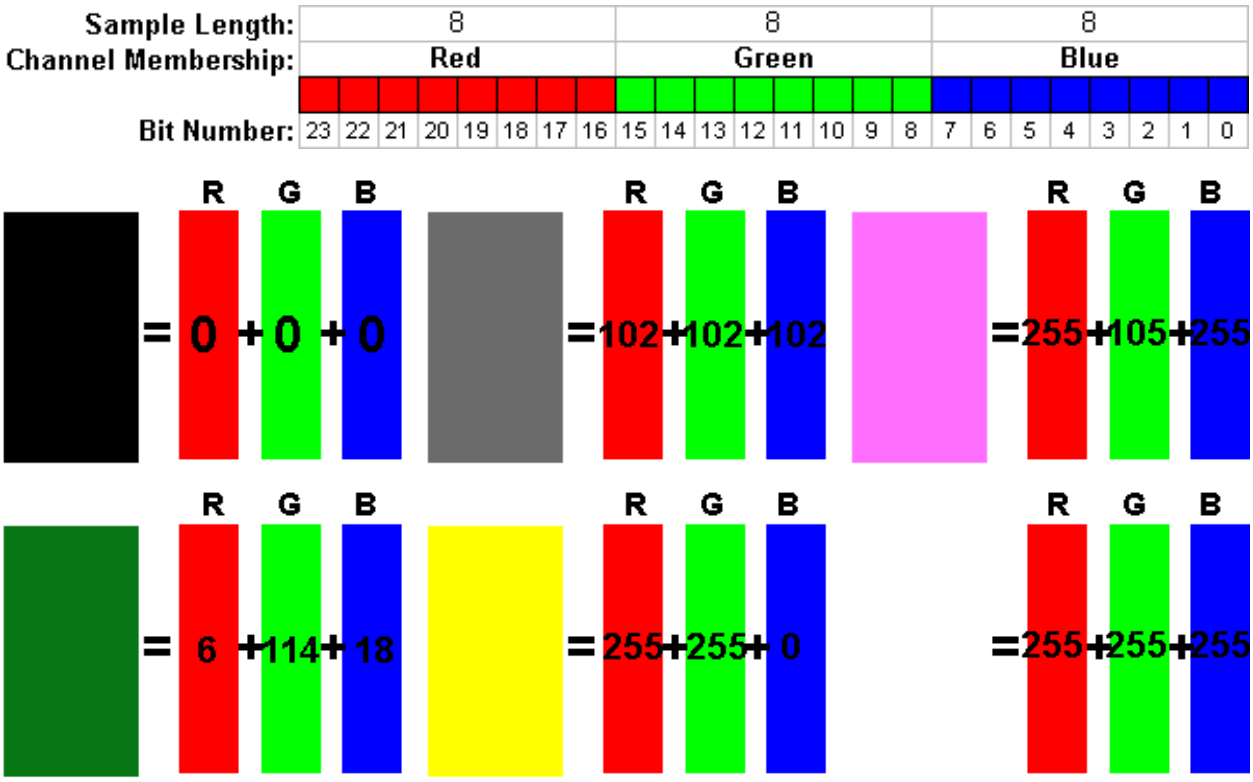


Рис. 1.3 – Модель кольору RGB

Як приклад розглянемо растровий малюнок (рис. 1.4). Кількість кольорів - 256 градацій сірого, роздільна здатність - приблизно 100 dpi.

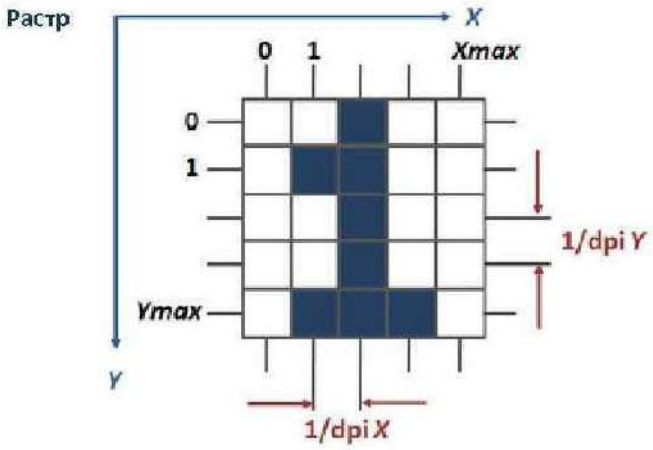


Рис. 1.4. Растр

1.2.1 Колірна модель RGB [1, 36-43]

Ця модель використовується для опису кольорів, які можуть бути отримані за допомогою пристроїв, що основані на принципі випромінювання. У якості основних кольорів вибрано червоний (Red), зелений (Green) та синій (Blue). Інші кольори та відтінки можуть бути отримані змішуванням певної кількості кожного з основних кольорів (рис. 1.5).

Рис. 1.5 – Базові кольори палітри RGB

Направивши на білий екран світло трьох джерел (червоний, зелений і синій), отримують таке зображення (рис. 1.6).

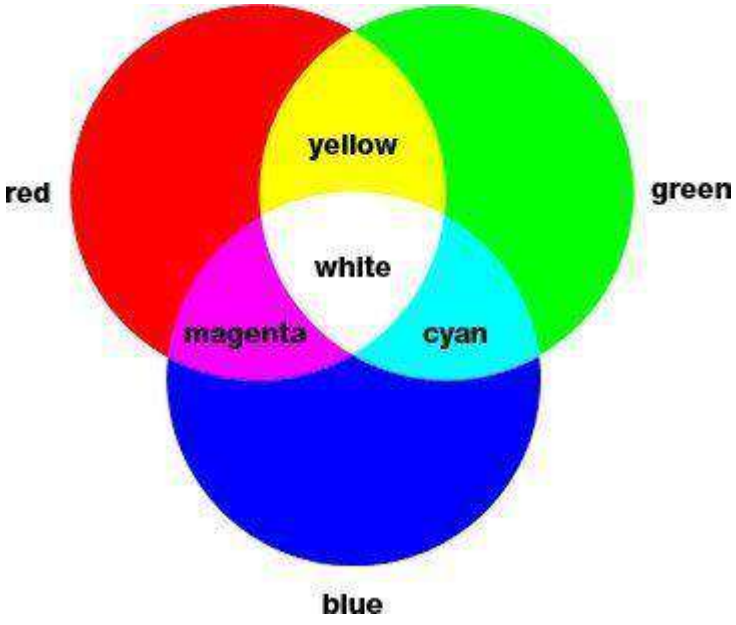


Рис. 1.6 – Основні кольори RGB та їх змішування

На екрані світло від джерел давало кольорові кола. У місцях перетинання кіл спостерігалося змішування кольорів. Жовтий колір давало змішування червоного й зеленого, блакитний — суміш зеленого Й синього, пурпурний — синього й червоного, а білий колір утворювався змішанням усіх трьох основних кольорів. Джеймс Максвелл (1831-1879) виготовив перший колориметр, за допомогою якого людина могла порівнювати монохроматичний колір і колір змішування в заданій пропорції компонентів RGB. Регулюючи яскравість кожного з компонентів, що змішуються, можна домогтися вирівнювання кольорів суміші й монохроматичного випромінювання. Це описується в такий спосіб:

$$Ц = rR + gG + bB,$$

де r, g і b - кількість відповідних основних кольорів.

Ця модель використовується для опису кольорів, які можуть бути отримані за допомогою пристроїв, які засновані на принципі випромінювання. Як основні кольори вибраний червоний (Red), зелений (Green) і синій (Blue). Інші кольори та відтінки можуть бути отримані змішуванням визначеної кількості основних кольорів.

Співвідношення коефіцієнтів r, g і b Максвелл наочно показав за допомогою трикутника, згодом названого його ім'ям [1, 36-43]. Трикутник Максвелла є рівнобічним, у його вершинах розташовуються основні кольори — R, G I B (рис. 1.7). Із заданої точки проводяться лінії, перпендикулярні сторонам трикутника. Довжина кожної лінії й показує відповідну величину коефіцієнта r, g чи b. Однакові значення r = g = b мають місце в центрі трикутника і відповідають білому кольору. Слід також зазначити, що деякі кольори відображаються точками зовні трикутника RGB — це означає від'ємне значення відповідного колірного коефіцієнта. Сума коефіцієнтів дорівнює висоті трикутника, а при висоті, що дорівнює одиниці, r + g + b = 1.

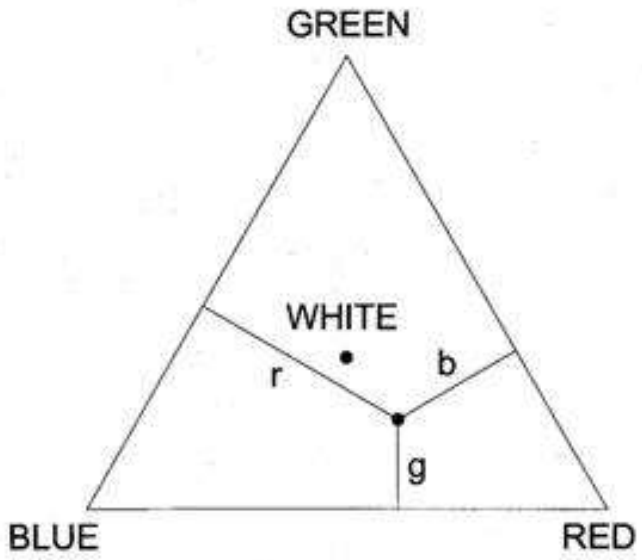


Рис. 1.7 – Трикутник Максвелла

Як основні кольори, Максвелл використовував випромінювання з такими довжинами хвиль: 630, 528 і 457 нм.

До теперішнього часу система RGB є офіційним стандартом. Рішенням Міжнародної Комісії з Освітлення — МКО (CIE — Commision International de VEclairage) у 1931 році були стандартизовані основні кольори, які було рекомендовано використовувати в якості R, G і B. Це монохроматичні кольори світлового випромінювання з довжинами хвиль відповідно:

R — 700 нм, G — 546.1 нм, B — 435.8 нм.

Червоний колір виходить за допомогою лампи розжарювання з фільтром. Для одержання чистих зелених і синіх кольорів використовується ртутна лампа. Також стандартизоване значення світлового потоку для кожного основного кольору [1, 36-43].

Ще одним важливим параметром для системи RGB є колір, одержаний після змішування трьох компонентів у рівних кількостях. Це білий колір. Виявляється, для того, щоб змішуванням компонентів R, G і B одержати білий колір, яскравості відповідних джерел не повинні бути рівними, а знаходитися у пропорції

$$L_R : L_G : L_B = 1 : 4{,}5907 : 0{,}0601$$

Якщо розрахунки кольору робляться для джерел випромінювання з однаковою яскравістю, то зазначене співвідношення яскравостей можна врахувати відповідними масштабними коефіцієнтами [1, 36-43].

Тепер розглянемо інші аспекти. Колір, створюваний змішуванням трьох основних компонентів, можна представити вектором у тривимірній системі координат R, G і B, зображеній на рис. 1.8.

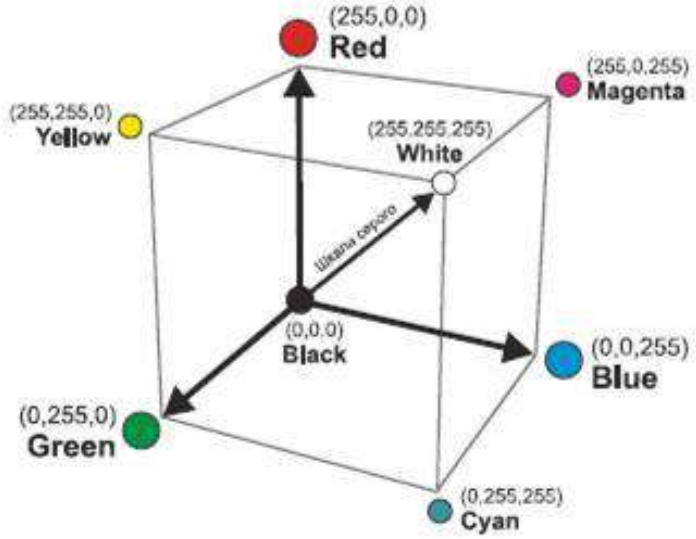


Рис. 1.8 – Тривимірні координати RGB

Чорному кольору відповідає центр координат — точка (0, 0, 0). Білий колір виражено максимальним значенням компонентів. Нехай це максимальне значення уздовж кожної осі дорівнює одиниці. Тоді білий колір — це вектор (1, 1, 1). Точки, що лежать на діагоналі куба від чорного до білого, мають однакові значення координат: Ri= Gi= Bi. Це градації сірого — їх можна вважати білим кольором різної яскравості. Узагалі говорячи, якщо усі компоненти вектора (r, g, b) помножити на однаковий коефіцієнт (k = 0 ... 1 ... 1), то колір (kr, kg, kb) зберігається, змінюється тільки яскравість. Тому для аналізу кольору важливе співвідношення компонентів. Якщо в колірному рівнянні

$$Ц = rR + gG + bB$$

розділити коефіцієнти r, g і b на їхню суму:

$$r' = \frac{r}{r+g+b},\ r' = \frac{r}{r+g+b},\ r' = \frac{r}{r+g+b},$$

то можна записати таке колірне рівняння

$$Ц = r' R + g' G + b' B.$$

Це рівняння репрезентує вектори кольору (r', g', b'), що лежать в одиничній площині r' + g' + b' = 1. Іншими словами, ми перейшли від куба до трикутника Максвелла.

У ході колориметричних експериментів були визначені коефіцієнти (r', g', b'), що відповідають чистим монохроматичним кольорам. Найпростіший колориметр —це призма з білого гіпсу, грані якої освітлюють джерелами світла. На ліву грань спрямоване джерело чистого монохроматичного випромінювання, а права грань освітлюється сумішшю трьох джерел RGB. Спостерігач бачить одночасно дві грані, що дозволяє фіксувати рівність кольорів.

Результати експериментів можна зобразити графічно (рис. 1.9).

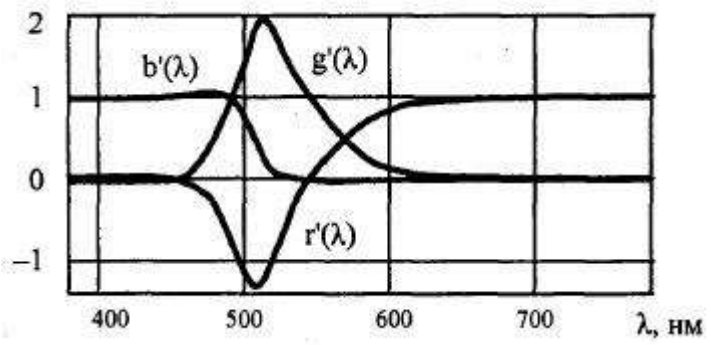


Рис. 1.9 - Триколірні коефіцієнти змішування RGB

Як бачимо, коефіцієнти r', g' і b' можуть бути і позитивними, і від'ємними. Що це означає? Те, що деякі монохроматичні кольори не можуть бути представлені сумою компонентів R, G і B. Але як відняти те, чого немає? Для вирівнювання кольору довелося додати до монохроматичного випромінювання один з компонентів R, G чи B. Наприклад, якщо монохроматичне випромінювання для деякого значення λ розбавлялося червоним, те це можна виразити так:

$$Ц(\lambda) + r'(\lambda) R = g'(\lambda) G + b'(\lambda) B$$

Як виявилося, жоден колір монохроматичного випромінювання (за винятком самих кольорів R, G і B) не може бути представлений тільки позитивними значеннями коефіцієнтів змішування. Це наочно можна зобразити за допомогою колірного графіка, побудованого на основі трикутника Максвелла (рис. 1.10).

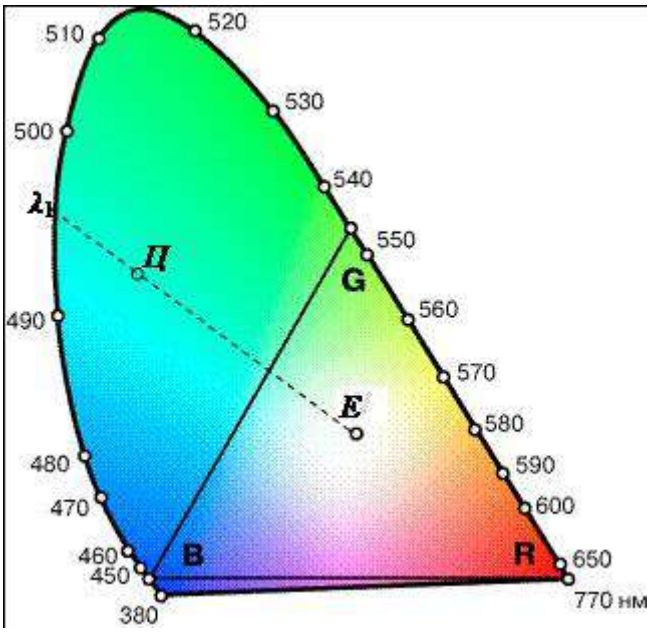


Рис. 1.10 - Колірний графік RGB

Верхня частина кривої лінії відповідає чистим монохроматичним кольорам, а нижня лінія — від 380 нм до 780 нм — представляє так називані пурпурні кольори (суміш синього й червоного), які не є монохроматичними. Точки, що лежать усередині контуру кривої, відповідають реальним кольорам, а поза цим контуром — нереальним кольорам. Точки усередині трикутника відповідають позитивним значенням коефіцієнтів r', g' та b' і представляють кольори, які можна одержати змішуванням компонентів RGB.

Таким чином, система RGB має неповне колірне охоплення — деякі насичені кольори не можуть бути представлені сумішшю зазначених трьох компонентів. У першу чергу, це кольори від зеленого до синього, включаючи усі відтінки блакитного — вони відповідають лівій частині кривої колірного графіка. Ще раз підкреслимо, що мова тут іде про насичені кольори, оскільки, наприклад, ненасичені блакитні кольори змішуванням компонент RGB одержати можна. Незважаючи на неповне охоплення, система RGB

широко використовується в даний час — у першу чергу, в кольорових телевізорах і дисплеях комп'ютерів. Відсутність деяких відтінків кольору не надто помітна.

Ще одним фактором, що сприяє популярності системи RGB, є її наочність — основні кольори знаходяться в трьох чітко помітних ділянках видимого спектра.

Крім того, однією з гіпотез, що пояснюють колірний зір людини, є трикомпонентна теорія, яка стверджує, що в зоровій системі людини є три типи світлочутливих елементів. Один тип елементів реагує на зелений, інший тип — на червоний, а третій тип — на синій колір. Така гіпотеза висловлювалася ще Ломоносовим [1, 36-43], її обґрунтуванням займалися багато вчених, починаючи з Т. Юнга. Утім, трикомпонентна теорія не є єдиною теорією колірного зору людини.

На рис. 1.11 наведено схему формування 24 - бітового кольору, що забезпечує можливість відтворення 256 × 256 × 256 = 16,7 млн. кольорів.

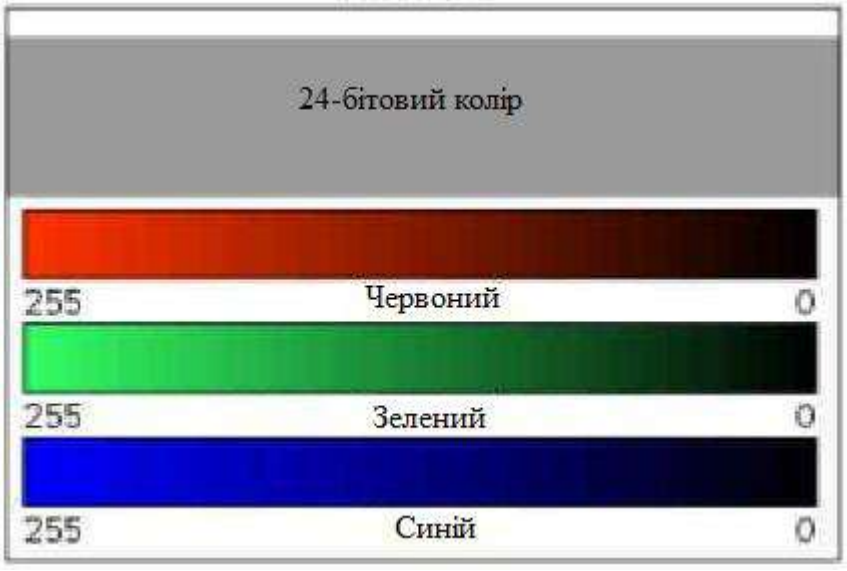


Рис. 1.11 - Кожен з трьох колірних компонентів RGB - тріади може приймати одне з 256 дискретних значень (від максимальної інтенсивності 255 до нульової, що відповідає чорному кольору)

На рис. 1.12 приведена ілюстрація отримання за допомогою адитивного синтезу п’ятнадцяти (з 16,7 млн.) кольорів. Як вже згадувалося раніше, у разі, коли усі три колірні компоненти мають максимальну

інтенсивність, результуючий колір здається білим. Якщо усі компоненти мають нульову інтенсивність, то результуючий колір - чистий чорний.

Номер кольору	R	G	B	Назва кольору	Колір
0	0	0	0	Чорний	
1	128	0	0	Темно-червоний	
2	0	128	0	Зелений	
3	128	128	0	Коричн.-зелений	
4	0	0	128	Темно-синій	
5	128	0	128	Темно-пурпурний	
6	0	128	128	Синьо-зелений	
7	128	128	128	Сірий 50%	
8	192	192	192	Сірий 25%	
9	255	0	0	Червоний	
10	0	255	0	Яскраво-зелений	
11	255	255	0	Жовтий	
12	0	0	255	Синій	
13	255	0	255	Фіолетовий	
14	0	255	255	Блакитний	
15	255	255	255	Білий	

Рис. 1.12 - Формування 15 з 16,7 млн. можливих кольорів шляхом варіації інтенсивностей кожної з трьох компонентів R, G і B колірної моделі rgb

1.3 Афінні перетворення

Задамо деяку двовимірну систему координат (x, y). Аффінне перетворення на площині описується формулами:

$$\begin{cases} X = Ax + By + C, \\ Y = Dx + Ey + F, \end{cases}$$

де A, B,.., F - константи. Значення (X, Y) можна розглядати як координати в новій системі координат.

Зворотне перетворення (X, Y) в (x, y) також є афінним:

$$\begin{cases} x = A'X + B'Y + C', \\ y = D'X + E'Y + F'. \end{cases}$$

Афінне перетворення зручно записувати в матричному виді. Константи A, B.... F утворюють матрицю перетворення, яка, будучи помноженою на матрицю-стовпець координат (x, y), дає матрицю-стовпець (X, Y). Проте, щоб врахувати константи C і F, необхідно перейти до так званих однорідних координат - додамо ще один рядок в матрицях координат:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Тепер розглянемо окремі випадки афінного перетворення.

Паралельний зсув координат (мал. 1. 13).

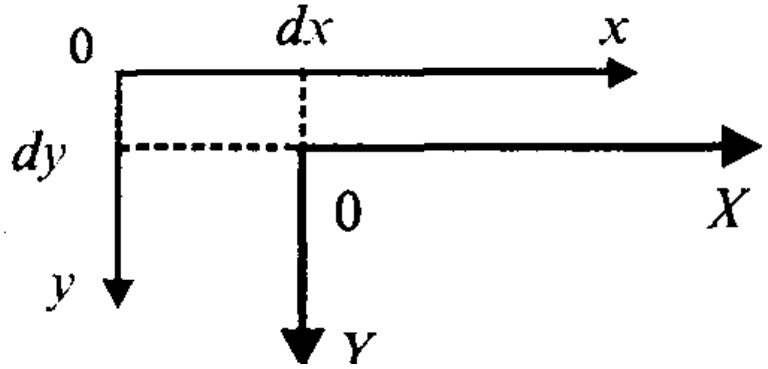


Рис. 1.13. Паралельний зсув координат

$$\begin{cases} X = x - dx, \\ Y = y - dy. \end{cases}$$

У матричній формі□

Зворотне перетворення: □

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}.$$

Розтягнення-стиснення осей координат (рис. 1. 14).

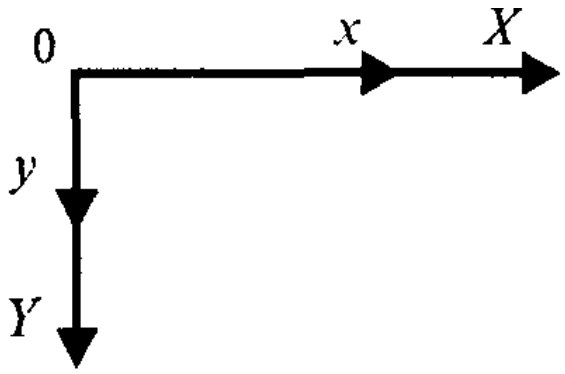


Рис. 1.14. Розтягнення-стиснення осей координат

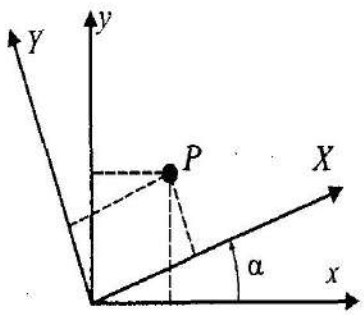
$$\begin{bmatrix} 1/k_x & 0 & 0 \\ 0 & 1/k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$\begin{cases} X = x/k_x, \\ Y = y/k_y, \end{cases}$$

Зворотне перетворення:

Коефіцієнти kx і ky можуть бути негативними. Наприклад, kx = - 1 відповідає дзеркальному віддзеркаленню відносно осі y.

Поворот (рис. 1. 15).



або

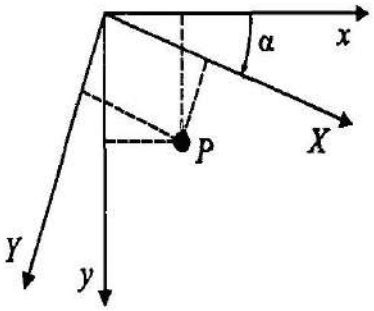


Рис. 1.15. Поворот

$$\begin{cases} X = x\cos\alpha + y\sin\alpha, \\ Y = -x\sin\alpha + y\cos\alpha, \end{cases} \qquad \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Зворотне перетворення відповідає повороту системи (X, Y) на кут (-α).

$$\begin{cases} x = X\cos\alpha - Y\sin\alpha, \\ y = X\sin\alpha + Y\cos\alpha, \end{cases}$$

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Властивості афінного перетворення. Будь-яке афінне перетворення може бути представлене як послідовність операцій з числа вказаних простих: зсув, розтягування/стиснення і поворот.

Зберігаються прямолінійність лінії, паралельність прямих, відношення довжин відрізків, що лежать на одній прямій, і співвідношення площ фігур.

Тривимірне афінне перетворення. Запишемо у вигляді формули:

$$\begin{aligned} X &= Ax + By + Cz + D, \\ Y &= Ex + Fy + Gz + H, \\ Z &= Kx + Ly + Mz + N, \end{aligned}$$

де A, B,..., N— константи.

Дамо також запис в матричній формі:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ K & L & M & N \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Для тривимірного простору будь-яке афінне перетворення також може бути представлене послідовністю простих операцій. Розглянемо їх.

1. Зсув осей координат відповідно на dx, dy, dz:

$$\begin{cases} X = x - dx, \\ Y = y - dy, \\ Z = z - dz, \end{cases}$$

$$\begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2. Розтягування/стиснення на kx, ky, kz.

$$\begin{cases} X = x / k_x, \\ Y = y / k_y, \\ Z = z / k_z, \end{cases} \qquad \begin{bmatrix} 1/k_x & 0 & 0 & 0 \\ 0 & 1/k_y & 0 & 0 \\ 0 & 0 & 1/k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

3. Повороти. Можна сказати, що в тривимірному просторі існує більше різновидів повороту, порівняно з двовимірним простором. Розглянемо декілька окремих випадків повороту.

Поворот навколо осі x на кут φ (мал. 1. 16).

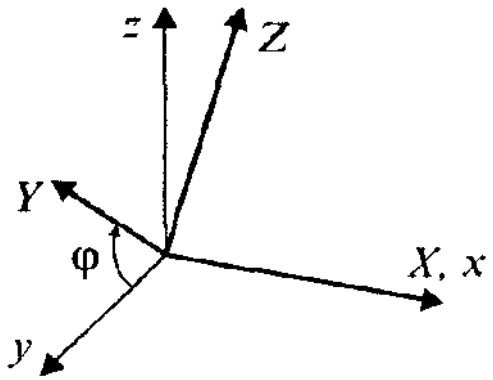
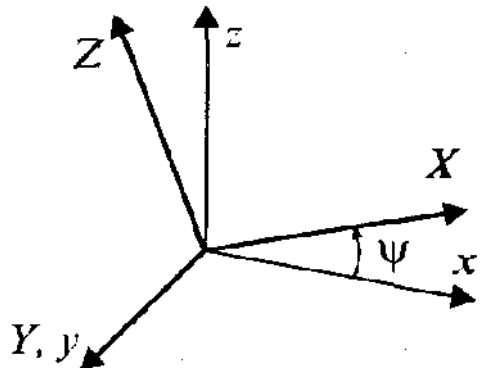


Рис. 1.16. Поворот навколо осі X

$$\begin{cases} X = x, \\ Y = y\cos\varphi + z\sin\varphi, \\ Z = -y\sin\varphi + z\cos\varphi, \end{cases}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$



1.17.1 Поворот навколо осей y и z

Поворот навколо осі y на кут ψ (рис. 1. 17.1).

$$\begin{cases} X = x\cos\psi + z\sin\psi, \\ Y = y, \\ Z = -x\sin\psi + z\cos\psi, \end{cases}$$

$$\begin{bmatrix} \cos\psi & 0 & \sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

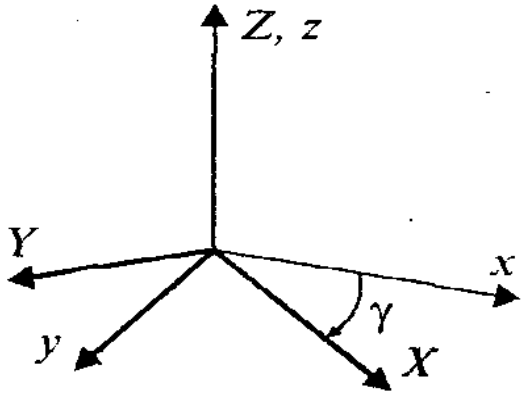


Рис. 1.17.2 Поворот навколо осей y и z

Поворот навколо осі z на кут γ (рис. 1. 17.2).

$$\begin{cases} X = x\cos\gamma + y\sin\gamma, \\ Y = -x\sin\gamma + y\cos\gamma, \\ Z = z, \end{cases}$$

$$\begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

1.4 Фрактальне представлення зображень

Поява фракталів в математичній літературі близько ста років тому була сприйнята дуже негативно, як це бувало і в історії розвитку багатьох інших математичних ідей [7]. Термін фрактал уперше ввів в 1975 році Бенуа Мандельброт. Слово фрактал утворюється від латинського дієслова frangere - ламати, і прикметника fractus - дробовий. Слід зазначити, що математичні ідеї сформувалися задовго до цього в XIX столітті в роботах Георга Кантора, Карла Вейєрштраса, Джузеппе Пеано, Гастона Жюліа, Пьера Фату та інших. Поняття фрактальної (дробової) розмірності з'явилося в 1919 році в роботі Фелікса Хаусдорфа. Проте, саме Бенуа Мандельброт об'єднав ці ідеї і поклав початок систематичному вивченню фракталів і їх застосувань [8].

В результаті зусиль Бенуа Мандельброта фрактальна геометрія стала прикладною наукою. Він і його учні відкрили багато нових фракталів, наприклад, фрактальний броунівський рух для моделювання лісового і гірського ландшафтів, флуктуації рівня річок і биття серця.

Відкриття фракталів призвело до революції не лише в геометрії, але і у фізиці, хімії, біології. Фрактали ініціювали інтенсивний розвиток теорії інформації. Фрактальні алгоритми знайшли застосування в інформаційних технологіях, наприклад для синтезу тривимірних комп'ютерних зображень природних ландшафтів.

Траєкторії часток броунівського руху, яким займалися Роберт Броун ще в 1828 році і Альберт Ейнштейн в 1905 році, є прикладом фрактальних кривих, хоча їх математичний опис був даний тільки в 1923 році Норбертом Вінером. У 1890 році Пеано сконструював свою знамениту криву - безперервне відображення, що переводить відрізок в квадрат і, отже, підвищує його розмірність з одиниці до двійки.

Фрактал, жодним чином не схожий на криву, який Мандельброт назвав пилом - це класична множина Кантора. Ця множина настільки розріджена, що вона не містить інтервалів, але, має стільки ж точок, скільки інтервалів. Мандельброт використовував такий "пил" для моделювання стаціонарного шуму в телефонії. Фрактальний пил того або іншого роду з'являється в багатьох ситуаціях. Фактично, він є універсальним фракталом в тому сенсі, що будь-який фрактал, - аттрактор системи ітерованих функцій - є або фрактальним пилом, або його проекцією на простір з нижчою розмірністю.

Фрактали досить часто зустрічаються в природі. Це контури берегової лінії, сніжинки, хмари, дерева, крони дерев. Різні деревовидні фрактали застосовувалися не лише для моделювання дерев-рослин, але і бронхіального дерева, роботи нирок, кровоносної системи і т.п. Цікаво відмітити припущення Леонардо да Вінчі про те, що усі гілки дерева на даній висоті, складені докупи, дорівнюють по товщині стволу (нижче їх рівня). Звідси слідує фрактальна модель для крони дерева у вигляді поверхні-фрактала [7].

Фрактал - геометрична фігура, що складена з декількох частин, кожна з яких подібна до усієї фігури у цілому. Коли ми дивимося на фрактал, то бачимо структуру, елементи якої залишаються однаковими незалежно від масштабу. Така структура є рекурсивною моделлю, кожна частина якої повторює розвиток усієї моделі в цілому. У найпростішому випадку кожна частина фрактала є зменшеною копією усього зображення. Як приклад можна привести зображення папороті Барнслі (рис. 1.18), побудованого за допомогою чотирьох перетворень [9].

Рис. 1.18 Папороть Барнслі

Сфера застосування фракталів в самих різних галузях фізики, хімії, біології, лінгвістики, музики, образотворчого мистецтва дуже велика і загальновідома. Успіх такого широкого використання фракталів обумовлений тим, що властивість власної подібності притаманна величезному числу структур як просторових (берегова лінія, поверхні зламу, різні агрегати, отримані за рахунок злиття), так і тимчасових (траєкторії руху). Мандельброт же знайшов єдину форму кількісного опису для цієї властивості, що нерідко дає можливість будувати прості моделі складних неврегульованих систем і вивчати їх за допомогою комп'ютерного експерименту. Такий підхід іноді дозволяє істотно покращити дослідження об'єктів, які іншим способом не піддаються розумінню і кількісному опису [10].

1.5 Методи стиснення зображень з втратами

Першими для архівації зображень стали застосовуватися звичні алгоритми, що використовуються в системах резервного копіювання, при створенні дистрибутивів. Ці алгоритми архівували інформацію без змін.

Проте основною тенденцією, останнім часом, стало використання нових класів зображень. Старі алгоритми перестали задовольняти вимогам, що пред'являються до архівації. Багато зображень практично не стискалися, хоча і мали явну надмірність. Це привело до створення нового типу алгоритмів - стискаючих з втратою інформації. Як правило, коефіцієнт архівації і, отже, величину втрат якості в них можна задавати. При цьому досягається компроміс між розміром і якістю зображень.

Одна з серйозних проблем машинної графіки полягає в тому, що досі не знайдений адекватний критерій оцінки втрат якості зображення. А втрачається вона постійно - при оцифровуванні, при переведенні в обмежену палітру кольорів, при переведенні в іншу систему представлення кольорів для друку, і, що для нас особливо важливе, при архівації з втратами.

Краще всього втрати якості зображень оцінювати візуально, за допомогою зору. Відмінною вважається архівація, при якій неможливо розрізнити первинне і відновлене зображення. Доброю - коли сказати, яке із зображень піддавалося архівації, можна тільки порівнюючи дві картинки, що знаходяться поруч. При подальшому збільшенні степені стискання, як правило, стають помітні побічні ефекти, характерні для цього алгоритму. На практиці, навіть при відмінному збереженні якості, в зображення можуть бути внесені регулярні специфічні зміни. Тому алгоритми архівації з втратами не рекомендується використовувати при стисненні зображень, які надалі передбачається друкувати з високою якістю, або обробляти програмами розпізнавання образів. Неприємні ефекти з такими зображеннями, можуть виникнути навіть при простому масштабуванні [11].

Алгоритм JPEG. Метод стиснення JPEG забезпечує високий коефіцієнт стиснення для малюнків фотографічної якості. Формат файлу JPEG, що використовує цей метод стиснення, розроблений об'єднаною групою експертів по фотографії (Joint Photographic Experts Group). Стиснення по методу JPEG сильно зменшує розмір файлу з растровим малюнком (можливий коефіцієнт стиснення100: 1). Високий коефіцієнт

стиснення досягається за рахунок стиснення з втратами, при якому в результуючому файлі втрачається частина початкової інформації. Метод JPEG використовує той факт, що людське око дуже чутливе до зміни яскравості, але зміни кольору він помічає гірше. Тому при стисненні цим методом запам'ятовується більше інформації про різницю між яскравостями відеопікселів і менше - про різницю між їх кольорами. Оскільки вірогідність помітити мінімальні відмінності в кольорі сусідніх пікселів мала, зображення після відновлення виглядає майже незмінним. Користувачеві надається можливість контролювати рівень втрат, вказуючи міру стиснення. Завдяки цьому, можна вибрати найбільш відповідний режим обробки кожного зображення: можливість завдання коефіцієнта стиснення дозволяє зробити вибір між якістю зображення і економією пам'яті. Якщо зображення, що зберігається, - фотографія, призначена для високохудожнього видання, то ні про які втрати не може бути і мови, оскільки малюнок має бути відтворений як можна точніше. Якщо ж зображення - фотографія, яка буде розміщена на вітальній листівці, то втрата частини початкової інформації не має великого значення. Експеримент допоможе визначити найбільш допустимий рівень втрат для кожного зображення.

JPEG - це алгоритм стиснення, заснований не на пошуку однакових елементів, як в RLE і LZW, а на різниці між пікселами. Кодування даних відбувається у декілька етапів. Спочатку графічні дані конвертуються в колірний простір типу LAB, потім відкидається половина або три чверті інформації про колір (залежно від реалізації алгоритму). Далі аналізуються блоки 8x8 пікселів. Для кожного блоку формується набір чисел. Перші декілька чисел представляють колір блоку в цілому, в той час, як наступні числа відбивають тонкі деталі. Спектр деталей базується на зоровому сприйнятті людини, тому великі деталі помітніші.

На наступному етапі, залежно від вибраного вами рівня якості, відкидається певна частина чисел, що представляють тонкі деталі. На останньому етапі використовується кодування методом Хафмана для

ефективнішого стиснення кінцевих даних. Відновлення даних відбувається в зворотному порядку.

Таким чином, чим вище рівень компресії, тим більше даних відкидається, тим нижче якість. Використовуючи JPEG можна отримати файл в 1-500 разів менше, ніж BMP. Формат апаратно незалежний, повністю підтримується на різних платформах. JPEG не підтримує індексовані палітри кольорів. Спочатку в специфікаціях формату не було і CMYK, Adobe додала підтримку кольороподілу.

Існують підформати JPEG. Baseline Optimized - файли дещо краще стискаються, але не читаються деякими програмами. JPEG Baseline Optimized розроблений спеціально для Інтернету, усі основні браузери його підтримують. Progressive JPEG так само розроблений спеціально для Інтернету, його файли менші за стандартні, але трохи більші Baseline Optimized. Головна особливість Progressive JPEG в підтримці аналога через стрічкового виводу.

Із сказаного можна зробити наступні висновки. JPEG 'ом краще стискати растрові картинки фотографічної якості, чим логотипи або схеми - в них більше півтонових переходів, серед однотонних заливок же з'являються небажані перешкоди. Краще стискаються і з меншими втратами великі зображення для web або з високою друкарською резолюцією (200-300 і більше dpi), чим з низькою (72-150 dpi), оскільки в кожному квадраті 8x8 пікселів переходи виходять м'якші, за рахунок того, що їх (квадратів) в таких файлах більше. Небажано зберігати з JPEG-стисненням будь-які зображення, де важливі усі нюанси перенесення (репродукції) кольорів, оскільки під час стиснення відбувається відкидання колірної інформації. У JPEG слід зберігати тільки кінцевий варіант роботи, тому що кожне нове зберігання призводить до усе новим втратам (відкиданню) даних і зниженню якості початкового зображення.

Досягти великих коефіцієнтів стиснення, використовуючи один метод (крім фрактального), практично неможливо. Тому ефективне кодування

зображень виконується із застосуванням декількох методів за декілька етапів. Ця концепція і покладена в основу стандартів, розроблених міжнародною організацією по стандартизації (International Organization for Standardization - ISO). Стандарт JPEG (Joint Photographic Expert Group) призначений для стиснення нерухомих зображень [12].

JPEG - досить потужний алгоритм. Практично він є стандартом де-факто для повноколірних зображень. Оперує алгоритм областями 8x8, на яких яскравість і колір змінюються порівняно плавно. Внаслідок цього, при розкладанні матриці такої області в подвійний ряд по косинусах, значущими виявляються тільки перші коефіцієнти. Таким чином, стиснення в JPEG здійснюється за рахунок плавності зміни кольорів в зображенні [11].

В цілому алгоритм базується на дискретному косинусоїдальному перетворенні - ДКП (Discrete-Cosine Transform - DCT), що застосовується до матриці зображення для отримання деякої нової матриці коефіцієнтів. Для отримання початкового зображення застосовується зворотне перетворення.

ДКП розкладає зображення по амплітудах деяких частот. Таким чином, при перетворенні ми отримуємо матрицю, в якій багато коефіцієнтів які близькі, або дорівнюють нулю. Крім того, людська система колірного сприйняття слабо розпізнає певні частоти. Тому можна апроксимувати деякі коефіцієнти грубіше без помітної втрати якості зображення.

Для цього використовується квантування коефіцієнтів (Quantization). У найпростішому випадку - це арифметичний побітовий зсув управо. При цьому перетворенні втрачається частина інформації, але можуть досягатися великі коефіцієнти стиснення [11].

Процес кодування за схемою JPEG ділиться на такі етапи (рис. 1.19):

1. Перетворення зображення в оптимальний колірний простір (тільки у випадку кодування кольорових зображень).

2. Субдискретизація компонент різницевих колірних сигналів шляхом усереднення груп пікселів (тільки у випадку кодування кольорових зображень).

3. Виконання ДКП для зменшення надлишковості даних зображення.

4. Квантування кожного блоку коефіцієнтів ДКП із застосуванням вагових функцій, оптимізованих з урахуванням сприйняття людиною.

5. Кодування результувальних коефіцієнтів із застосуванням статистичного кодування Хаффмана.

Необроблені дані зображення	
⇧	⇩
Перетворення в колірний простір	Перетворення в колірний простір
⇧	⇩
Зворотна субдискретизація	Субдискретизація
⇧	⇩
Зворотне ДКП	Пряме ДКП
⇧	⇩
Деквантування	Квантування
⇧	⇩
Декодування	Кодування
⇧	⇩
Дані стиснуті за методом JPEG	

Рис. 1.19 Послідовність операцій при стисненні зображень за методом JPEG.

При виконанні першого етапу виконується перетворення кольорового зображення з системи представлення RGB в іншу систему, наприклад YUV, де Y – сигнал яскравості, U і V – різницеві колірні сигнали.

Перетворення виконується від піксела до піксела за такими формулами:

$$Y = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B;$$
$$U = -0.15 \cdot R - 0.29 \cdot G + 0.44 \cdot B;$$
$$V = 0.62 \cdot R - 0.52 \cdot G - 0.1 \cdot B.$$

де *R, G, B* – колірні сигнали зображення (червоний, зелений і синій відповідно).

Представлення кольорового зображення в системі YUV дає можливість використати особливості зорового сприйняття зображень людиною – низьку чутливість до точності представлення кольорів. Це виконується за рахунок субдискретизації компонент різницевих колірних сигналів U і V шляхом усереднення груп пікселів (тільки у випадку кодування кольорових зображень). Наприклад, усереднення значень відліків в межах примикальних квадратів з розмірами сторін 2x2, зменшує розмір компонент U і V в чотири рази без помітного зменшення якості зображення. Враховуючи те, що на кожний піксель витрачається три байти (YUV), виграш значний.

Найбільш важким для реалізації є виконання ДКП. Однак, завдяки наявності швидких алгоритмів (ті ж самі, що для обчислення дискретного перетворення Фур'є - ДПФ), число арифметичних операцій може бути зменшено в десятки разів. Процес кодування із застосуванням ДКП пояснює рис. 1.20.

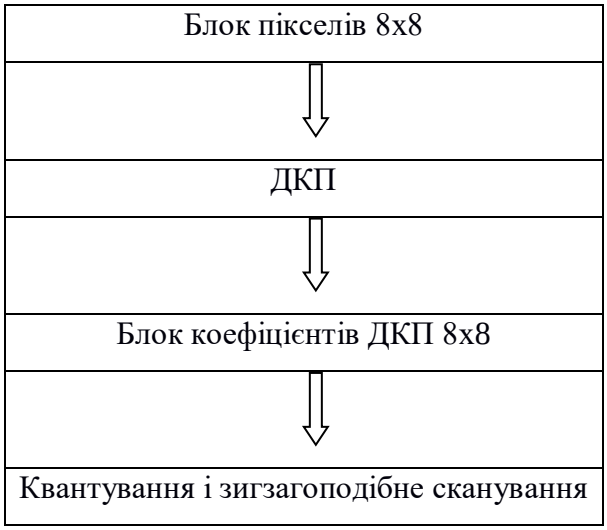


Рис. 1.20 Кодування з застосуванням ДКП.

Зображення розбивається на примикальні, один до одного, блоки розміром 8x8 (при кодуванні кольорових зображень кожна компонента обробляється незалежно) [12]. В межах кожного блоку виконується двовимірне ДКП у відповідності з виразом:

$$F(u,v) = \frac{1}{4} C(u)C(v) \sum_{I=0}^{7} \sum_{J=0}^{7} f(i,j) \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right),$$

де

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & x = 0 \\ 1 & x <> 0, \end{cases}$$

$$u, v = 0,1,2 \ldots 7.$$

При декодуванні обчислюється зворотне ДКП:

$$f(i,j) = \frac{1}{4} \sum_{U=0}^{7} \sum_{V=0}^{7} C(u)C(v)F(u,v) \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right),$$

$$i, j = 0,1,2 \ldots 7.$$

Квантування виконується за рахунок ділення кожного коефіцієнта ДКП на свій "коефіцієнт квантування" з округленням результату до цілого. Терми більшого порядку квантуються з більшим "коефіцієнтом квантування". Крім того, для даних яскравості і кольору застосовуються різні таблиці квантування, оскільки зір людини має різну чутливість до яскравості і кольору зображення.

На етапі статистичного кодування, крім алгоритму Хаффмана, специфікація JPEG допускає застосування інших методів з метою зменшення об'єму інформації.

Алгоритм JPEG 2000. Розроблений тією ж групою експертів в області фотографії, що і JPEG. Формування JPEG як міжнародного стандарту було закінчено в 1992 році. У 1997 стало ясно, що потрібний новий, гнучкіший і потужніший стандарт, який і був допрацьований до зими 2000 року.

Основні відмінності алгоритму в JPEG 2000 від алгоритму JPEG полягають в наступному [11]:

1. Краща якість зображення при значній величині стиснення.
2. Підтримка кодування окремих областей з кращою якістю. Відомо, що окремі області зображення критичні для сприйняття людиною (наприклад, очі на фотографії), тоді як якістю інших можна знехтувати (наприклад, задній план). При оптимізації власноруч, збільшення міри стиснення проводиться до тих пір, поки не буде втрачено якість в якійсь важливій частині зображення. Зараз з'являється можливість задати якість в критичних областях, стиснувши інші області сильніше, тобто ми отримуємо ще більшу остаточну міру стиснення при суб'єктивно рівній якості зображення.
3. Основний алгоритм стиснення замінений на Wavelet. Окрім вказаного підвищення міри стиснення це дозволило позбавитися від 8-піксельної, блоковості, що виникає при підвищенні міри стиснення. Крім того, плавний прояв зображення тепер закладений в стандарт.
4. Для підвищення міри стиснення в алгоритмі використовується арифметичне стиснення. Спочатку в стандарті JPEG також було закладено арифметичне стиснення, проте пізніше воно було замінене менш ефективним стисненням по Хаффману, оскільки арифметичне стиснення було захищене патентами. Зараз термін дії основного патенту збіг і з'явилася можливість поліпшити алгоритм.
5. Підтримка стиснення без втрат. Окрім звичного стиснення з втратами новий JPEG тепер підтримує і стиснення без втрат. Таким чином, стає можливим використання JPEG для стиснення медичних зображень, в

поліграфії, при збереженні тексту під розпізнавання OCR системами і так далі.

6. Підтримка стиснення однобітових (2-кольорових) зображень. Для збереження однобітових зображень (малюнки тушшю, відсканований текст і тому подібне) раніше усюди рекомендувався формат GIF, оскільки стиснення з використанням ДКП дуже неефективне до зображень з різкими переходами кольорів. У JPEG при стисненні 1-бітова картинка приводилася до 8-бітової, тобто збільшувалася в 8 разів. Після чого робилася спроба стиснення, нерідко менш ніж в 8 разів. Зараз можна рекомендувати JPEG 2000 як універсальний алгоритм.
7. На рівні формату підтримується прозорість. Плавно накладати фон при створенні WWW-сторінок тепер можна не лише в GIF, але і в JPEG 2000. Крім того, підтримується не лише 1 біт прозорості (піксель прозорий/непрозорий), а окремий канал, що дозволило задавати плавний перехід від непрозорого зображення до прозорого фону.

Крім того, на рівні формату підтримуються включення в зображення інформації про копірайт, підтримка стійкості до бітових помилок при передачі і широкомовленні, можна залучати для декомпресії або обробки зовнішні засоби (plug-ins), можна включати в зображення його опис, інформацію для пошуку і так далі.

Базова схема JPEG 2000 дуже схожа на базову схему JPEG. Відмінності полягають в наступному:

1. Замість дискретного косинусного перетворення (ДКП) використовується дискретне вейвлет-перетворення (ДВП).
2. Замість кодування по Хаффману використовується арифметичне стиснення.
3. У алгоритм, з самого початку, закладено управління якістю областей зображення.

4. Не використовується явно дискретизація компонент U і V після перетворення колірних просторів, оскільки при ДВП можна досягти того ж результату, але дещо краще.

Рекурсивний (хвильовий) алгоритм стиснення зображень. Англійська назва рекурсивного стиснення – Wavelet. Цей алгоритм орієнтований на стиснення кольорових і чорно-білих зображень з плавними переходами, ідеальний для картинок типу рентгенівських фотографій. Коефіцієнт стиснення варіюється в межах 5-100. При великих коефіцієнтах стиснення на різких границях, особливо діагональних, можливі спотворення [11].

Основна ідея алгоритму полягає в тому, що зберігаються різниці між середніми значеннями сусідніх блоків зображення, які звичайно приймають значення близькі до нуля.

Рекурсивне стиснення базується на пірамідальному S-перетворенні, яке може використовуватись для стиснення фотореалістичних зображень як майже без втрат, так і з втратами.

Стиснення виконується за два кроки: перший - S-перетворення початкового зображення; другий - перетворені дані кодуються одним з статистичних методів. Обидві операції зворотні, що дозволяє відновити початкове зображення. Однак для отримання великих коефіцієнтів стиснення необхідно зниження точності представлення компонент зображення, отриманих в результаті виконання S-перетворення, але так щоб спотворення не були візуально помітні.

Структура кодування приведена на рис. 3. Вхідні дані обробляються фільтрами S-перетворення, які формують компоненти зображення. Адаптивний квантувач призначений для квантування значень відліків компонент з урахуванням особливостей зорового сприйняття. Процес виконання квантування пов'язаний з втратами інформації, однак ці втрати повинні бути непомітними. Тобто візуально якість відновленого зображення не повинна відрізнятися від початкового.

Власне стиснення може бути виконано на основі одного з методів статистичного кодування (наприклад, кодування Хаффмана, LZW-кодування, арифметичного кодування) [12].

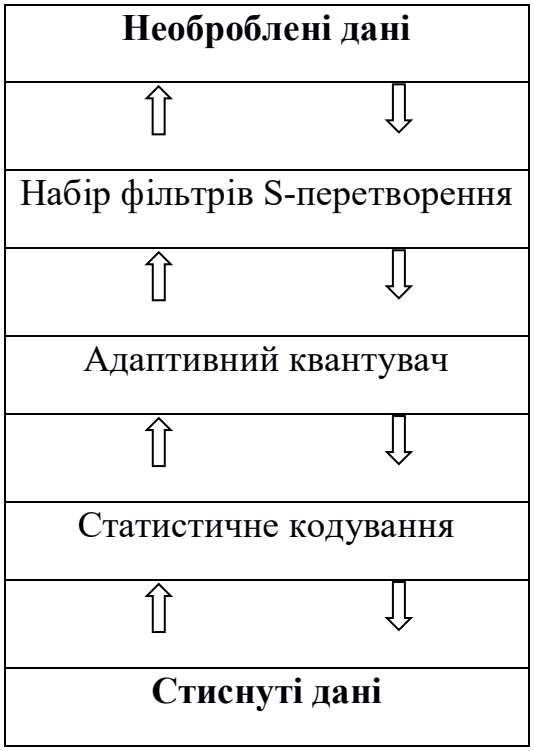


Рис. 1.21 Схема хвильового кодування.

До переваг цього алгоритму можна віднести те, що він дуже легко дозволяє реалізувати можливість поступового "проявлення" зображення при передачі його по мережі. Крім того, оскільки на початку зображення ми фактично зберігаємо його зменшену копію, що спрощує його показ по заголовку.

На відміну від JPEG і фрактального алгоритмів цей метод не оперує блоками, наприклад 8x8 пікселів. Точніше ми оперуємо блоками 2x2, 4x4, 8x8 і так далі. Проте за рахунок того, що коефіцієнти для цих блоків ми зберігаємо незалежно, є можливість досить легко уникнути дроблення зображення на "мозаїчні" квадрати [11].

1.6. Фрактальний метод стиснення зображень

Продуктивність комп'ютерів росте не щодня, а щогодини. Сьогодні вони вже настільки потужні, що спроможні, наприклад, відтворювати відеоінформацію в реальному часі. Проте канали зв'язку розвиваються значно повільніше. Велика частина користувачів Internet підключається до мережі по телефонних каналах, що, як відомо, не відрізняються високою пропускною спроможністю. Саме тому, ідея перегляду повноекранного відео по Internet поки що так і залишилася ідеєю через низькі коефіцієнти стиснення зображень. Того, що досягнуто в цьому напрямку зараз явно не достатньо. До мети можна було б наблизитися, якби коефіцієнт стиснення був біля декількох сотень, чого в даний час не спостерігається.

З відомих методів кодування зображень найбільші коефіцієнти стиснення забезпечують методи, до яких відноситься і метод фрактального стиснення.

Народження фрактальної геометрії часто пов'язують з ім'ям відомого бельгійського вченого Мандельброта. У 1977 році вийшла в світ його книга «The Fractal Geometry of Nature» [1]. Мандельброт показав, що традиційна геометрія, із її ідеально рівними і гладкими площинами, не може описати такі об'єкти, як дерева, хмари, гори. З іншої сторони їх можна задати об'єктами нескінченно складної природи - фракталами. Існує множина методів генерації фракталів. Математики, із їхньою невтомною пристрастю до узагальнень, стали шукати щось загальне між цими методами. Першим знайшов це «щось» Джон Хатчинсон, якому ми зобов'язані створенням нової теорії - теорії ітерованих функцій. Через деякий час Майкл Бансли, дослідник компанії Georgia Tech, написав книгу «Fractals Everywhere» [2]. У ній він описав клас систем ітерованих функцій, що задають зображення. Незабаром математики достатньо добре засвоїли методи побудови фрактальних зображень і перейшли до питання створення системи ітерованих функцій по

заданому зображенню. Розв'язок цієї задачі дозволить описати зображення як набір функцій, а значить досягти значної економії пам'яті.

Однак процес кодування зображень цим методом вимагає значних обчислювальних затрат, тому швидкодія методу невисока. Хоча потрібно відмітити, що процес декодування зображень, стиснених цим методом не потребує таких високих потужностей та ресурсів операційної системи.

Високі коефіцієнти стиснення разом з високою швидкістю декодування роблять цей метод привабливим у використанні. Так, випущена у 1997 році фірмою MICROSOFT енциклопедія ENCARTA вміщує на одному компакт-диску велику кількість текстової інформації ілюстрованої близько 7000-ми фотографій закодованих саме цим методом.

Низька швидкодія стиснення пов’язана з тим, що для отримання високої якості вихідного зображення необхідно обробляти велику кількість доменних областей, причому сам процес обробки одного домену також включає в себе потужні обчислення (рис. 1.21). Тому, актуальним є проведення досліджень по пошуку критеріїв які дозволяли б для даного рангового блоку відібрати близькі за відстанню доменні блоки.

Рис. 1.21 Самоподібні фрагменти зображення.

Фрактальна архівація заснована на тому, що за допомогою коефіцієнтів системи ітерованих функцій - СІФ (Iterated Function System - IFS) зображення представляється в компактнішій формі. Перш ніж розглядати процес архівації, розберемо, як СІФ будує зображення [13].

СІФ - це набір тривимірних афінних перетворень, що переводять одне зображення в інше. Перетворенню піддаються точки в тривимірному просторі (x координата, y координата, яскравість). Цей процес продемонстрував Майкл Барнслі у своїй книзі "Фрактальне стиснення зображень". У ній введено поняття Фотокопіювальної Машини, яка складається з екрану, на якому зображена початкова картинка, і системи лінз, що проектують зображення на інший екран. Кожна лінза проектує частину початкового зображення. Розставляючи лінзи і міняючи їх характеристики, можна управляти отримуваним зображенням. На лінзи накладається вимога - вони повинні зменшувати в розмірах проектовану частину зображення. Крім того, вони можуть міняти яскравість фрагмента і проектують не круги, а області з довільною межею.

Фрактальне кодування відрізняється від інших методів стиснення з втратами. Метод JPEG забезпечує ущільнення, відкидаючи ті компоненти зображення, які не важливі для сприйняття людським оком. Отримані в результаті дані надалі обробляються за допомогою методу стиснення без втрат. Для досягнення високої міри стиснення необхідно відкидати більше даних, що призводить до погіршення якості зображення і появи дискретних (блокових) ділянок.

Фрактальні зображення не будуються на базі растру, а кодування не порівнюється з візуальними характеристиками людського ока. Навпаки, растрові дані відкидаються, якщо необхідно створити найкращий фрактальний малюнок. Високий ступінь стиснення досягається шляхом виконання великого об'єму перетворень і обчислень, що може погіршити якість зображення, проте завдяки фрактальним компонентам спотворення не такі помітні.

Більшість інших методів стиснення з втратами за своїми характеристиками симетричні. Це означає, що вони засновані на використанні конкретної послідовності операцій, які при розпаковуванні виконуються в зворотному порядку. На стиснення і розпаковування потрібно приблизно однаковий час. Фрактальне стиснення – процес асиметричний. Стиснення триває набагато довше, ніж розпаковування. Ця характеристика дає можливість ефективно використовувати метод для зображень, які безперервно розпаковуються, але ніколи не стискаються. Тому фрактальний метод доцільно застосовувати в базах даних зображень. Так, випущена у 1997 році фірмою Microsoft енциклопедія Encarta вміщує на одному компакт-диску велику кількість текстової інформації та ілюстрована близько сімома тисячами фотографій, які закодовані саме цим методом.

Фрактальний метод кодування зображень забезпечує коефіцієнти стиснення, що становлять від п'ятдесяти до п'ятисот разів залежно від типу зображення і допустимого рівня шуму перетворення. Такі коефіцієнти стиснення разом з високою швидкістю декодування роблять цей метод привабливим у використанні. Сам процес, що використовується деякими фрактальними пакетами, включаючи Fractal Transform Майкла Барнслі, вважається патентованим [15].

Проблемам оптимізації фрактального стиснення цифрових зображень присвячена велика кількість досліджень зарубіжних учених [16–22]. Слід відмітити роботу С. В. Винокурова [20], у якій розглянуто алгоритм фрактального стиснення зображень з використанням просторово-чутливого хешування. Проте досі мало уваги приділяється методам оптимізації фрактального стиснення, які б мали однаково високі характеристики ступеня стиснення і якості відновленого зображення. Особливо актуальним на сьогоднішній день залишається питання побудови швидкодіючих алгоритмів фрактальної компресії.

1.7. Системи ітеруючих функцій

Серед різноманітних методів кодування фрактальний метод дозволяє отримувати найбільші коефіцієнти стиснення. З фізичної точки зору фрактальне кодування ґрунтується на твердженні, що зображення містить афінну надлишковість. Математична модель, яка використовується при фрактальному стисненні зображень, називається системами ітеруючих функцій (Iterated Function Systems – IFS) [12]. Системи ітеруючих функцій містять набір стискаючих перетворень ω_i, які можливо задати так:

$$W(S) = \bigcup_{i=1}^{n} \omega_i(S) \tag{1}$$

де S – зображення,

ω_i – набір стискаючих перетворень.

Відповідно до теореми Банаха [23], існує певний клас відображень, які називаються стискаючими і для них справедливо таке твердження: якщо до якогось зображення f_0 почати багаторазово застосовувати відображення W таким чином, що:

$$f_1 = W(f_1), \quad f_i = W(f_{i-1}) \tag{2}$$

то в межі при i, що прямує до нескінченності, отримаємо таке саме зображення незалежно від того, яке зображення прийнято за f_0:

$$f = \lim_{i \to \infty} f_i \tag{3}$$

Зображення f називається нерухомою точкою перетворення W або атрактором.

В якості перетворень w_i використовуються афінні відображення:

$$w_i \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & S_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ O_i \end{bmatrix} \tag{4}$$

де a_i, b_i, c_i, d_i – афінні коефіцієнти деформації, стиснення, обертання,

dx, dy – коефіцієнти переміщення,

x, y – координати точки, що перетворюється,

z – її інтенсивність.

Параметр S_i керує контрастністю, а O_i – яскравістю зображення. Знаючи коефіцієнти цих перетворень, можна відновити початкове зображення [12].

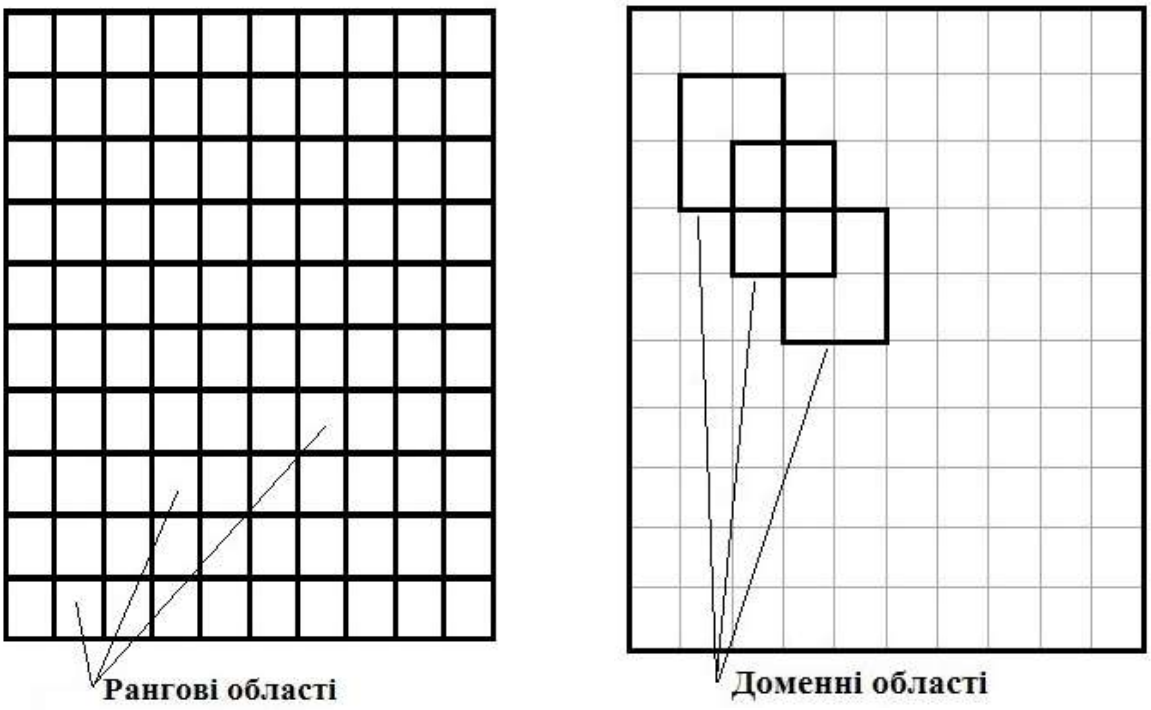
Алгоритм фрактального кодування зображень можна описати так. Процес стиснення починається з того, що беруться два ідентичні екземпляри зображення - А і Б, один з них розділяється на блоки, що не перекриваються (рангові області), а на другому задається набір доменів, які можуть взаємно перекриватися, як це показано на рис. 1.22. Домени повинні включати характерні фрагменти, які надалі використовуються для побудови декодованого зображення. Після цього починається кодування зображення шляхом підбору для кожної рангової області найбільш відповідного домену, за допомогою якого розподіл яскравості в ранговій області може бути апроксимований розподілом яскравості в домені. Для того, щоб отримати найкращу апроксимацію, домени піддаються афінним перетворенням, в результаті яких відбувається не лише їх геометрична деформація, але і зміни контрасту та яскравості. Якщо розподілом яскравості в перетвореному домені не вдається досягти задовільної апроксимації розподілу яскравості в ранговій області, рангова область ділиться на чотири частини і процес повторюється. Якість необхідної апроксимації задається у вигляді допустимого значення середнього квадрата помилки апроксимації (середнього квадрата невідповідності). Номери доменів, використаних при кодуванні кожної рангової області, а також коефіцієнти афінних перетворень стискаються шляхом кодування ентропії і записуються у файл. Файл стисненого зображення містить заголовок з інформацією про розташування рангових областей і доменів, а також таблицю ефективно упакованих афінних коефіцієнтів для кожної рангової області.

Одна з можливих схем кодування зображень фрактальним методом, запропонована А. Жакеном [24, 25], містить такі етапи:

– Зображення розділяється на примикаючі одна до одної області розміром N x N (рангові області).

– Задається набір доменних областей. Доменні області можуть перекриватись, вони не повинні обов’язково закривати всю поверхню зображення. Розміри доменних областей звичайно вибирають 2N x 2N.

Для кожної рангової області підбирається доменна область, яка після афінних перетворень найбільш точно апроксимує рангову область. На практиці застосовується вісім варіантів відображення одного квадрата в інший з використанням афінних перетворень, які приведені на рис. 1.23. Це повороти зображення на кути 90,180, 270 (-90) градусів відносно його центра і перетворення симетрії відносно ортогональних осей, які проходять через центр фрагменту перпендикулярно його сторонам.



а *б*

Рис. 1.22 Розподіл на зображенні: *а* – рангових областей; *б* – доменних областей

Точність апроксимації F визначається за допомогою середньоквадратичного критерію:

$$F = \sum_{i,j} (Sd_{ij} + O_{ij} - r_{ij})^2 \tag{5}$$

де d_{ij} – значення, отримані в результаті усереднення по фрагментам з розмірами 2×2 елементів доменної області, що приводить її розмір до розміру рангової області,

r_{ij} – значення елементів рангової області,

O_{ij} – зміщення, може бути як константою так і описуватись поліномами першого, другого або третього порядків.

Прирівнявши до нуля часткові похідні від виразу (5) по S і O:

$$\frac{dF}{dS} = 0, \ \frac{dF}{dO} = 0, \tag{6}$$

можна знайти значення S і O, при яких досягається мінімум виразу (5):

$$O = \frac{1}{n^2}\left(\sum_{i,j}^{n} r_{ij} - S\sum_{i,j}^{n} d_{ij}\right) \tag{7}$$

$$S = \frac{n^2\sum_{i,j}^{n} r_{ij}d_{ij} - \sum_{i,j}^{n} r_{ij}\sum_{i,j}^{n} d_{ij}}{n^2\sum_{i=j}^{n} {d_{ij}}^2 - (\sum_{i,j}^{n} d_{ij})^2} \tag{8}$$

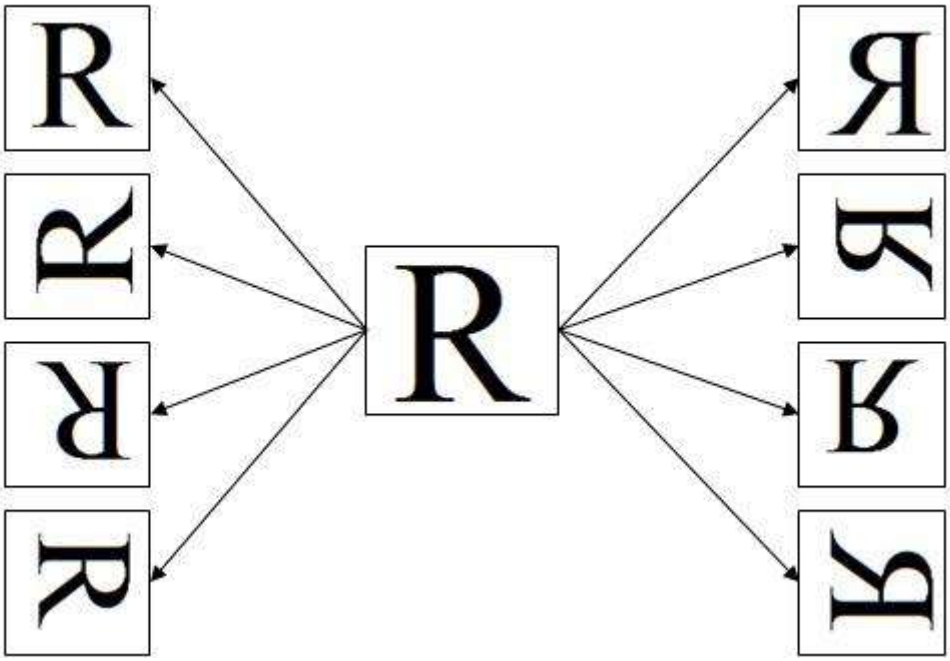


Рис. 1.23 - Можливі варіанти відображення одного квадрата в інший з використанням афінних перетворень.

Домені блоки звичайно вибираються з кроком n/2 при n=4. У вихідний файл записуються такі параметри:

– координати доменної області з найменшим значенням F_{min};
– значення для O і S, отримані згідно формул (7, 8);
– номер афінного перетворення.

Алгоритм декодування полягає в тому, що беруться два екземпляри одного і того ж зображення А і Б, розподіл яскравості в яких неважливий. На цих зображеннях виділяються області, межі яких співпадають з межами рангових областей і доменів, а потім, використовуючи відомі значення афінних коефіцієнтів, по доменах, виділених на зображенні Б, знаходяться розподіли яскравості в рангових областях зображення А. Після цього зображення А і Б міняються місцями і операція повторюється. Можна показати, що при багатократному повторенні цієї операції розподіл яскравості в зображеннях А і Б наближатиметься до розподілу яскравості в початковому зображенні. Звернемо увагу на те, що алгоритми стиснення і

декомпресії асиметричні. Слід також відмітити, що процес стиснення вимагає набагато більше часу, чим процес декомпресії.

Декодування стисненого зображення носить ітераційний характер і складається з таких етапів [26]:

– Створюються два зображення однакового розміру А і Б. Розмір цих зображень не обов'язково рівний розміру початкового зображення, початковий рисунок областей А і В будь-який.

– Зображення Б розбивається на рангові області так як на першій стадії процесу стиснення. Для кожної рангової області зображення Б виконується афінне перетворення відповідної доменної області зображення А і результат поміщається в Б.

– Виконуються операції ідентичні попередньому пункту, тільки зображення А і Б міняються місцями.

– Багатократно повторюються другий і третій кроки до тих пір поки зображення А і Б не стануть нерозрізненими.

Основним недоліком фрактального методу є низька швидкість кодування, яка пов'язана з тим, що для отримання високої якості зображення для кожного рангового блоку необхідно виконати перебір всіх доменних блоків, і для кожного доменного блоку необхідно виконати не менше восьми афінних перетворень.

В цифровій обробці сигналів швидкодія методу оцінюється кількістю арифметичних операцій, необхідних для виконання перетворення. Оцінимо, наприклад, число операцій множення при кодуванні зображень фрактальним методом (вважаючи, що S=1), оскільки для їх виконання витрачається найбільше часу. Нехай MxN розміри зображення і $M=N=n^к$, де n - розмір сторін рангового блоку. Нехай крок вибору доменних блоків n/2. Для порівняння даного рангового блоку з доменним блоком необхідно виконати $8n^2$ операцій множення. Тоді загальна кількість операцій множення для пошуку відповідного доменного блоку така:

$M = 8n^2$ * *(кількість доменних блоків),*

Кількість доменних блоків при кроці вибору n/2 складе:

$$\text{Кількість доменних блоків}_{n/2} = (\frac{n^k - 2n}{n/2} + 1) * (\frac{n^k - 2n}{n/2} + 1),$$

Після підстановки кінцева формула має такий вигляд:

$$M = 8(4n^{k+1}(n^{k-1} - 3) + 9n^2),$$

Для порівняння, виконання ДКП для блоку такого ж розміру, навіть без використання швидких алгоритмів, вимагає лише n^4 операцій множення [26].

1.8 Аналіз методів побудови системи IFS - функцій фрактального стиснення зображень

Найпростіший і найповільніший спосіб фрактального кодування є перевірка кожного доменного блоку і виконання обчислень згідно формул (5, 7, 8). Такий спосіб називається повним пошуком або повним перебором. При кодуванні зображень природного походження можна підвищити швидкодію кодування, прийнявши S=1, оскільки враховуючи статистику зображень завжди знайдеться доменний блок, який апроксимує заданий ранговий блок з необхідною точністю. Тоді з виразів (5, 7) одержимо:

$$F = \sum_{i,j} (d_{ij} + O_{ij} - r_{ij})^2 \qquad (9)$$

$$O = \frac{1}{n^2}\left(\sum_{i,j}^{n} r_{ij} - \sum_{i,j}^{n} d_{ij}\right) \qquad (10)$$

Контрастність декодованого зображення може бути відновлена після декодування іншими методами. Таке спрощення дозволяє знизити кількість арифметичних операцій на 60% і відповідно підвищити швидкість кодування.

Однак, такий спосіб підвищення швидкодії не є коректним з тієї точки зору, що вже в самому алгоритмі передбачена візуально недостовірна передача деяких графічних зображень штучного походження, тому необхідні додаткові заходи для забезпечення хоча би візуальної достовірності.

1.9 Методи підвищення швидкості фрактального стиснення зображень

1.9.1 Підвищення швидкості фрактального стиснення шляхом зміни алгоритму порівняння рангу з доменами [27]

Для збільшення швидкості пропонується в якості критерію оптимальності використовувати коефіцієнт кореляції Пірсона:

$$r' = \rho(R, D)$$

Чим краще реальна залежність R від D апроксимується лінійною $T(D) = sD + oI$, тим ближче по модулю до 1 буде їх коефіцієнт кореляції. Де $|s| < 1$ - коефіцієнт контрасту; $o \in [-255, 255]$ - коефіцієнт яскравості; I - одинична матриця. Використання цього коефіцієнта дозволяє відразу оцінити оптимальність поточного домена для даного рангу, без розрахунку коефіцієнта перетворення контрасту і яскравості. Таким чином, коефіцієнти перетворення контрасту і яскравості для кожного рангу розраховуватимуться тільки один раз, а не для кожного порівняння ранг-домен. Формулу для коефіцієнта перетворення контрасту перепишемо через коефіцієнт кореляції:

$$\mathrm{s} = \mathrm{r}' \frac{\sigma \mathrm{R}}{\sigma \mathrm{D}}, \qquad (1)$$

де σ - середньоквадратичне відхилення.

Це відхилення яскравості пікселів рангів і доменів можна підрахувати заздалегідь, а не обчислювати при кожному порівнянні, що дозволить збільшити швидкість обробки на комп'ютері. Для коефіцієнта перетворення контрасту повинна виконуватися умова $|s| < 1$. Коефіцієнт кореляції, що є

першим співмножником в (1), не може бути більше одиниці по абсолютному значенню. Таким чином, якщо другий співмножник $\sigma R/\sigma D<1$, то $|s|<1$ для будь-яких R і D. Щоб забезпечити виконання цієї умови, ми повинні розглядати тільки ті домени, які задовольняють нерівності:

$$\sigma D>\sigma R \tag{2}$$

Практично умова (2) означає, що контрастність домена має бути вищою за контрастність рангу [27].

Пропонується підрахувати середнє значення яскравості пікселів для кожного рангу, а не коефіцієнт перетворення яскравості o для кожного домена, що допоможе збільшити швидкість стиснення:

$$o'=\overline{R} \tag{3}$$

Очевидно, що $o'\in[0,255]$ для будь-кого R на відміну від $o\in[-255,255]$. Це дозволяє заощадити 1 біт на знаку і, отже, пришвидшити стиснення. Формулу можна переписати у вигляді:

$$o=\overline{R}-s\overline{D} \tag{4}$$

Підставляючи (4) в вираз $T(D)=sD+oI$, отримаємо:

$$T(D)=sD+oI=s(D-\overline{D}I)+\overline{R}I \tag{5}$$

Відновлення рангової області відбувається аналогічно, тільки замість $R = sD + oI$, враховуючи (5) і (3), буде використовуватись формула:

$$R = s\left(D - \overline{D}I\right) + o'I \qquad (6)$$

1.9.2 Пошук доменних блоків, для яких точність апроксимації F не перевищує заданого значення

Передбачається перемінний розмір доменних і рангових блоків [26, 28, 29]. При постійному розмірі доменних і рангових блоків n = const пошук продовжується до тих пір, поки не буде знайдений доменний блок, який забезпечує умові $F < E_C$ (рис. 1.24).

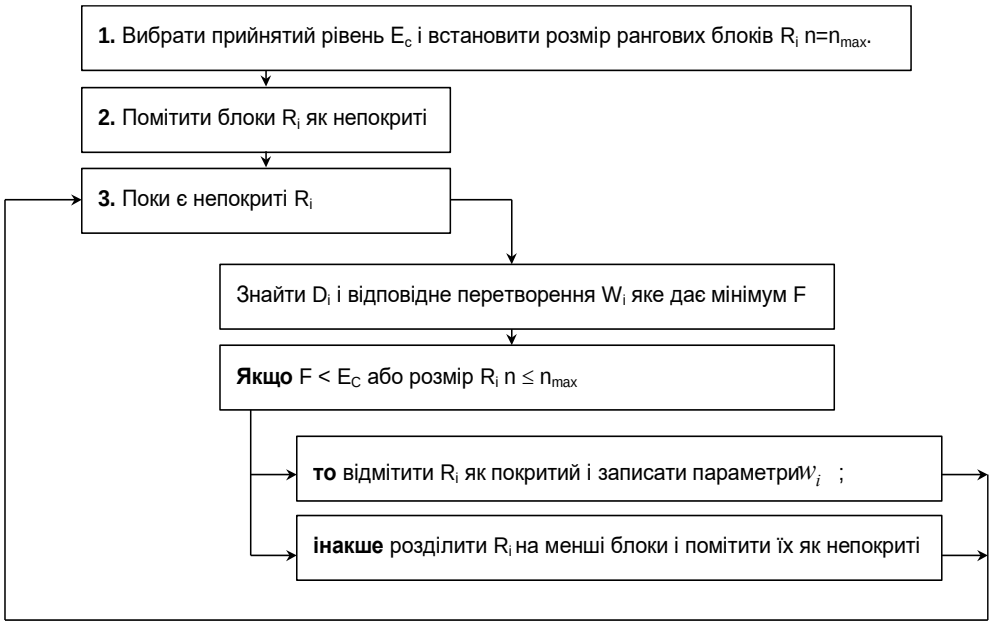


Рис. 1.24 Пошук доменних блоків, для яких задана точність апроксимації

1.9.3 **Локальний пошук** (Local Search)

Передбачає пошук доменного блоку тільки на ділянках зображення сусідніх з даним ранговим блоком [2, 11]. Також збільшити швидкість кодування можливо збільшивши крок вибору доменних блоків. Це зменшує простір пошуку. Однак, дуже великий крок веде до зниження якості зображення. Тому спочатку пошук виконується з великим кроком, і коли знаходиться блок з мінімальним значенням $F_{\min}$, то пошук продовжується навколо цього блоку з меншим кроком (рис. 1.25). Такий спосіб носить назву локальний субпошук (Local Sub Search – LSS).

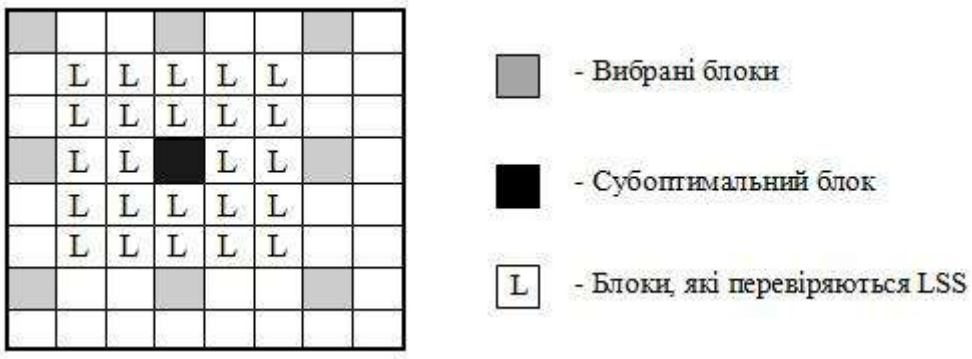


Рис. 1.25 Локальний субпошук

1.9.4 Ізометричне передбачення

Кількість афінних перетворень зменшується за рахунок ізометричного передбачення [30, 31]. Кожний доменний (ранговий) блок розділяється на чотири субблоки, обчислюється середнє значення яскравості в межах кожного субблоку і виконується сортування цих субблоків в порядку зростання. Порівнюючи сортований і несортований списки, обчислюється лексикографічний порядок перестановок:

$$Perm\# = inv(x_1)\cdot 3! + inv(x_2)\cdot 2! + inv(x_3)\cdot 1! + inv(x_4)\cdot 0!, \qquad (7)$$

де $inv(x_i)$ - кількість перестановок, яка відноситься до i-ої позиції (рис. 1.26).

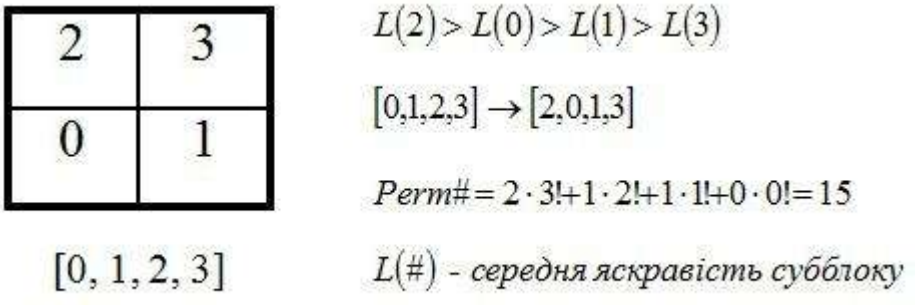


Рис. 1.26 Приклад обчислення лексикографічного порядку

Тепер кожний блок має свій лексикографічний порядок, що дає можливість визначати найкраще покриття рангового блоку.

1.9.5 Класифікація доменних і рангових блоків

Ранговий блок порівнюється з доменними блоками того ж самого класу [32-34]. Серед методів з класифікацією необхідно відзначити класифікацію, запропоновану Арно Жакуіном, яка ґрунтується на топології блоків і передбачає блоки трьох типів:

- блоки без контурів (shade blocks). Shade-блоки можуть бути апроксимовані за допомогою поліномів третього або другого порядку. Пошук для них не потрібен;
- блоки, інваріантні до орієнтації (текстурні блоки). Пошук блоків виконується без перевірки ізометрії (афінні перетворення не виконуються);
- контурні блоки (edge blocks). Тільки для таких блоків виконується повний перебір.

1.10 Висновки до першого розділу

У першому розділі **«Математична модель стиснення зображень фрактальним методом»** був проведений огляд переваг і недоліків та аналіз використання базового методу фрактального кодування зображень. Описана математична модель даного методу. Досліджено перспективні алгоритми кодування зображень з втратами, а також можливості покращення стиснення добре відомих методів та їх подальша практична реалізація.

Методи кодування зображень з втратами мають значно кращі показники стиснення у порівнянні з іншими існуючими методами. В багатьох ситуаціях невеликі втрати даних допустимі та компенсуються зростанням ступеня стиснення. Найбільш перспективним для подальшого розвитку, вдосконалення та практичного використання в комп’ютерних технологіях, на наш погляд, є фрактальний алгоритм. Він має декілька унікальних особливостей та переваг: відновлення зображення відбувається набагато швидше, відсутні спотворення на границі різких переходів кольорів, можливість використання даного методу для стиснення зображень, які готують для якісного друку, а також масштабування зображень.

Розділ 2. Удосконалення табличного методу підвищення часової ефективності фрактального стиснення зображень

2.1 Табличний метод пошуку

Доменний блок можливо представити як вектор з n^2 координатами $\vec{D}(d_1,d_2,...,d_{n^2})$, де: d_i - значення яскравості елементів блоку, n - розмір сторін рангового блоку.

Після виконання афінних перетворень утворюються ще 7 векторів, координати яких є перестановками координат початкового вектору. Якщо для представлення елементу d_i витрачається 8 біт, то розрядність числа $d_1d_2...d_n^2$ буде рівною 2^{8n^2}. Формуються таблиці, представлені на рис. 2.1.

Як видно з рисунку, для значення координати рангового блоку «k» вибираються доменні блоки з номерами m та m1, для яких необхідно обчислити тільки значення S та O згідно виразів (7, 8). Зрозуміло, що розміри таблиці 1 на рис. 2.1 не дозволяють поки що реалізувати такий підхід на практиці без попередньої обробки доменних блоків для скорочення розрядності представлення елементів цих блоків [26, 33, 34].

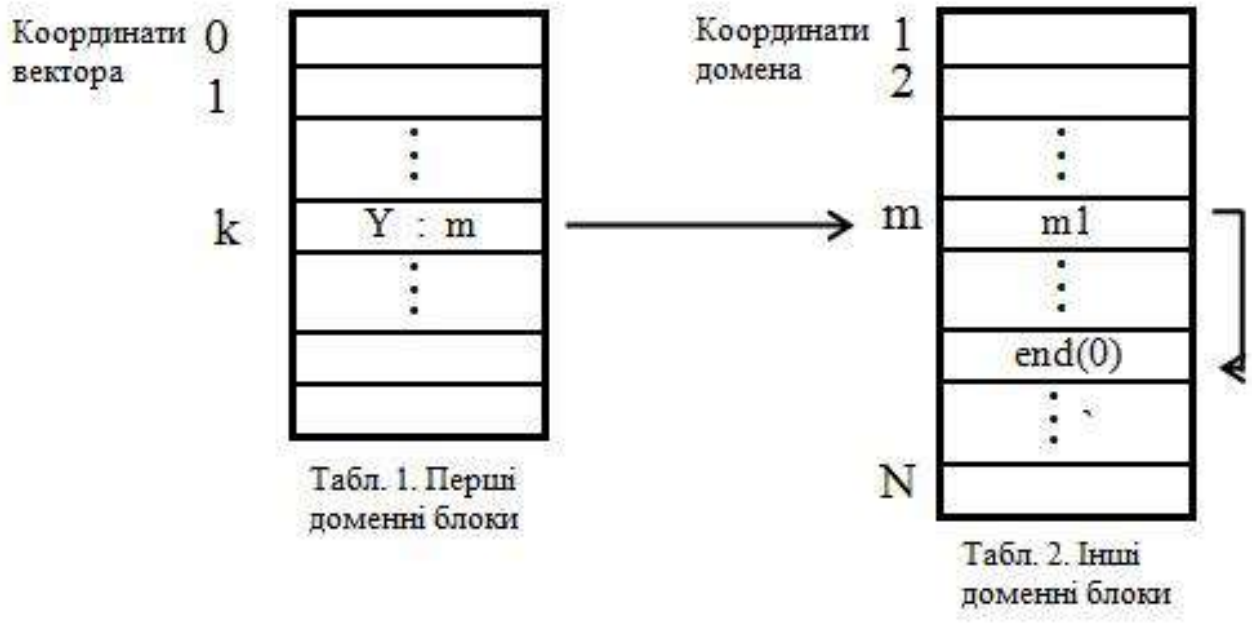


Рис. 2.1 Табличний метод пошуку

де: Y - номер афінного перетворення, m - координати домену в площині зображення, N - кількість доменних блоків.

2.2. Оцінка детальності блоку

Наш підхід побудови IFS - функцій фрактального стиснення ґрунтується на попередній обробці кожного доменного і рангового блоку одним з операторів виділення контурів зображення і оцінці детальності кожного з цих блоків. Найбільші переваги має використання методу підвищення часової ефективності, що розробляється в даній роботі, оскільки він точно ідентифікує контури в порівнянні з фільтром Лапласа та градієнтними операторами і забезпечує менші розміри таблиць пошуку:

$$СЗ_D = \frac{1}{4n^2}\left(\sum_{i=0}^{2n-1}\sum_{j=0}^{2n-1} f_D(i,j)\right)$$

$$СЗ_R = \frac{1}{n^2}\left(\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} f_R(i,j)\right)$$

$$G_D(i,j) = f_D(i,j) - СЗ_D;\ i,j = 0 \div 2n\text{-}1$$

$$G_R(i,j) = f_R(i,j) - СЗ_R;\ i,j = 0 \div n\text{-}1 \qquad (11)$$

де:$СЗ_D$ – середнє значення елементів даного доменного блоку,

$СЗ_R$ – середнє значення елементів даного рангового блоку,

$f_D(i,j)$ – елемент зображення доменного блоку,

$f_R(i,j)$ – елемент зображення рангового блоку,

$G_D(i,j)$ - контурна компонента доменних блоків,

$G_R(i,j)$ - контурна компонента рангових блоків.

f_R (0, 0)	f_R (0, 1)	f_R (0, 2)	f_R (0, 3)
f_R (1, 0)	f_R (1, 1)	f_R (1, 2)	f_R (1, 3)
f_R (2, 0)	f_R (2, 1)	f_R (2, 2)	f_R (2, 3)
f_R (3, 0)	f_R (3, 1)	f_R (3, 2)	f_R (3, 3)

а

f_D(0, 0)	f_D(0, 1)	f_D(0, 2)	f_D(0, 3)	f_D(0, 4)	f_D(0, 5)	f_D(0, 6)	f_D(0, 7)
f_D(1, 0)	f_D(1, 1)	f_D(1, 2)	f_D(1, 3)	f_D(1, 4)	f_D(1, 5)	f_D(1, 6)	f_D(1, 7)
f_D(2, 0)	f_D(2, 1)	f_D(2, 2)	f_D(2, 3)	f_D(2, 4)	f_D(2, 5)	f_D(2, 6)	f_D(2, 7)
f_D(3, 0)	f_D(3, 1)	f_D(3, 2)	f_D(3, 3)	f_D(3, 4)	f_D(3, 5)	f_D(3, 6)	f_D(3, 7)
f_D(4, 0)	f_D(4, 1)	f_D(4, 2)	f_D(4, 3)	f_D(4, 4)	f_D(4, 5)	f_D(4, 6)	f_D(4, 7)
f_D(5, 0)	f_D(5, 1)	f_D(5, 2)	f_D(5, 3)	f_D(5, 4)	f_D(5, 5)	f_D(5, 6)	f_D(5, 7)
f_D(6, 0)	f_D(6, 1)	f_D(6, 2)	f_D(6, 3)	f_D(6, 4)	f_D(6, 5)	f_D(6, 6)	f_D(6, 7)
f_D(7, 0)	f_D(7, 1)	f_D(7, 2)	f_D(7, 3)	f_D(7, 4)	f_D(7, 5)	f_D(7, 6)	f_D(7, 7)

б

Рис. 2.2 Елементи зображення в ранговому (а) та доменному (б) блоках

Обробка блоків згідно виразу (11) виключає з порівняння постійну складову зображення і тому дозволяє попередньо відібрати близькі до даного рангового блоку доменні блоки. Під час першого проходу оцінюється детальність доменних і рангових блоків. Такою оцінкою може бути сума модулів значень відліків контурної компоненти:

$$S_{k_D} = \frac{1}{4n^2}\left(\sum_{i=0}^{2n-1}\sum_{j=0}^{2n-1}|G_D(i,j)|\right) \tag{12}$$

$$S_{k_R} = \frac{1}{n^2}\left(\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}|G_R(i,j)|\right) \tag{13}$$

де: k_R - номер рангового блоку, k_D - номер доменного блоку, S_{k_D}, S_{k_R} - детальність відповідно доменного та рангового блоку.

Суми S_{k_R} заносяться в масив в порядку слідування рангових блоків, а для доменних блоків формуються таблиці, приведені на рис. 2.3

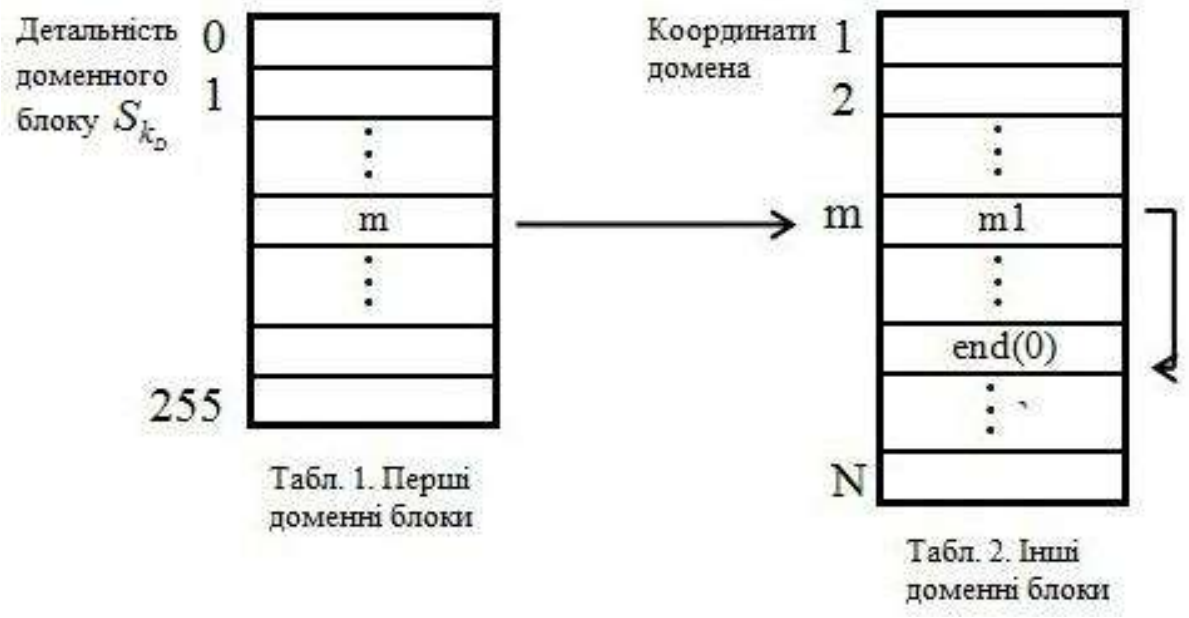


Рис. 2.3 Пошук доменного блоку по критерію детальності

де: m - координати домену в площині зображення, N - кількість доменних блоків.

Під час другого проходу для кожного значення детальності рангового блоку S_{k_R} з таблиць 1, 2 (рис. 2.3) відбираються відповідні доменні блоки, для яких виконується обробка, характерна для фрактального стиснення та обчислення згідно виразів (5, 7, 8). Оскільки вибраних блоків значно менше загальної кількості доменних блоків, то слід очікувати значного виграшу в швидкодії [44-61].

Розглянемо детально етапи розробки методу побудови IFS-функцій, який дозволить скоротити часові витрати при стисненні зображення. Візьмемо за основу класичне фрактальне стиснення зображень та спробуємо застосувати для попередньої оцінки доменних блоків модифікований метод. Початковий доменний блок зображення буде мати вигляд (рис. 2.4):

Source (I,J)	Source (I,J+1)	Source (I,J+2)	Source (I,J+3)	Source (I,J+4)	Source (I,J+5)	Source (I,J+6)	Source (I,J+7)
Source (I+1,J)	Source (I+1,J+1)	Source (I+1,J+2)	Source (I+1,J+3)	Source (I+1,J+4)	Source (I+1,J+5)	Source (I+1,J+6)	Source (I+1,J+7)
Source (I+2,J)	Source (I+2,J+1)	Source (I+2,J+2)	Source (I+2,J+3)	Source (I+2,J+4)	Source (I+2,J+5)	Source (I+2,J+6)	Source (I+2,J+7)
Source (I+3,J)	Source (I+3,J+1)	Source (I+3,J+2)	Source (I+3,J+3)	Source (I+3,J+4)	Source (I+3,J+5)	Source (I+3,J+6)	Source (I+3,J+7)
Source (I+4,J)	Source (I+4,J+1)	Source (I+4,J+2)	Source (I+4,J+3)	Source (I+4,J+4)	Source (I+4,J+5)	Source (I+4,J+6)	Source (I+4,J+7)
Source (I+5,J)	Source (I+5,J+1)	Source (I+5,J+2)	Source (I+5,J+3)	Source (I+5,J+4)	Source (I+5,J+5)	Source (I+5,J+6)	Source (I+5,J+7)
Source (I+6,J)	Source (I+6,J+1)	Source (I+6,J+2)	Source (I+6,J+3)	Source (I+6,J+4)	Source (I+6,J+5)	Source (I+6,J+6)	Source (I+6,J+7)
Source (I+7,J)	Source (I+7,J+1)	Source (I+7,J+2)	Source (I+7,J+3)	Source (I+7,J+4)	Source (I+7,J+5)	Source (I+7,J+6)	Source (I+7,J+7)

Рис. 2.4 Початковий доменний блок зображення

$$SZD = \frac{1}{4n^2}\left[\sum_{i,j=0}^{2n-1} Source(i,\ j)\right]$$

$$MasOpt(I,J) = Source\ (I,J) - SZD;\ I,J = 0 \div 2n\text{-}1 \qquad (14)$$

де: *Source(I,J)* - масив, який містить інтенсивність пікселів початкового зображення,

MasOpt(I,J) - масив, який має таку ж розмірність, як і масив *Source(I,J)*, його індекси відповідають індексам точок початкового зображення, а значення комірок розраховуються за формулою (14),

SZD - середнє значення елементів доменного блоку.

Застосування модифікованого методу буде складатись з таких етапів:

1. Розбиття початкового зображення на доменні блоки.

2. Застосування до кожного доменного блоку модифікованого методу підвищення часової ефективності.

3. Збереження отриманих значень в окремому масиві.

4. Створення двох масивів. Один буде мати розмірність, яка відповідає максимально допустимій інтенсивності точки зображення (наприклад від 0 до 255), а інший – як кількість доменних блоків для даного зображення.

5. Підрахунок для кожного доменного блоку значення S_k за формулою (12) та занесення отриманих значень в масив 1.

6. Створення масивів 1 та 2 за принципом FAT. Індекси першого масиву будуть відповідати підрахованим S_k. Тобто проводиться підрахунок S_k для конкретного доменного блоку та записується номер цього доменного блоку в таблицю 1 в комірку, індекс якої відповідає отриманому S_k. Значення цієї комірки одночасно буде індексом масиву 2, в який записується перелік доменних блоків з однаковими S_k. Якщо при розрахунках знайдено доменний блок з параметром S_k, а комірка в масиві 1 під таким індексом вже зайнята, то необхідно прочитати значення цієї комірки та записати в масив 2 номер знайденого доменного блоку в комірку з індексом, який відповідає значенню комірки масиву 1.

Таким чином буде отримано перелік доменних блоків з однаковими параметрами S_k. Фактично ми отримаємо доменні блоки з однаковими контурами, а значить можливо буде значно полегшити пошук найбільш близького доменного блоку для кожного рангового.

Наступним кроком програми буде класичне фрактальне стиснення. Але тепер не потрібно перебирати всі доменні блоки для кожного рангового. Достатньо підрахувати параметр S_k для рангового блоку і вибрати з масивів 1 та 2 доменні блоки, які найбільш близькі для даного рангового [62-82].

Граф-схему процесу відбору доменних блоків за допомогою модифікованого методу покращення часової ефективності подано на рис. 2.5. В граф-схемі використовуються наступні змінні та позначення:

I, J, K, L - Змінні типу “Ціле”, використовуються в якості змінних циклів, Q, Z - висота та ширина зображення, N - висота та ширина доменного блоку, Source - масив, що містить інтенсивність точок вхідного зображення та має відповідну розмірність.

MasOpt - масив, що має розмірність як і Source та містить значення кожної точки вхідного зображення після застосування методу підвищення часової ефективності.

SZD - середнє значення елементів доменного блоку.

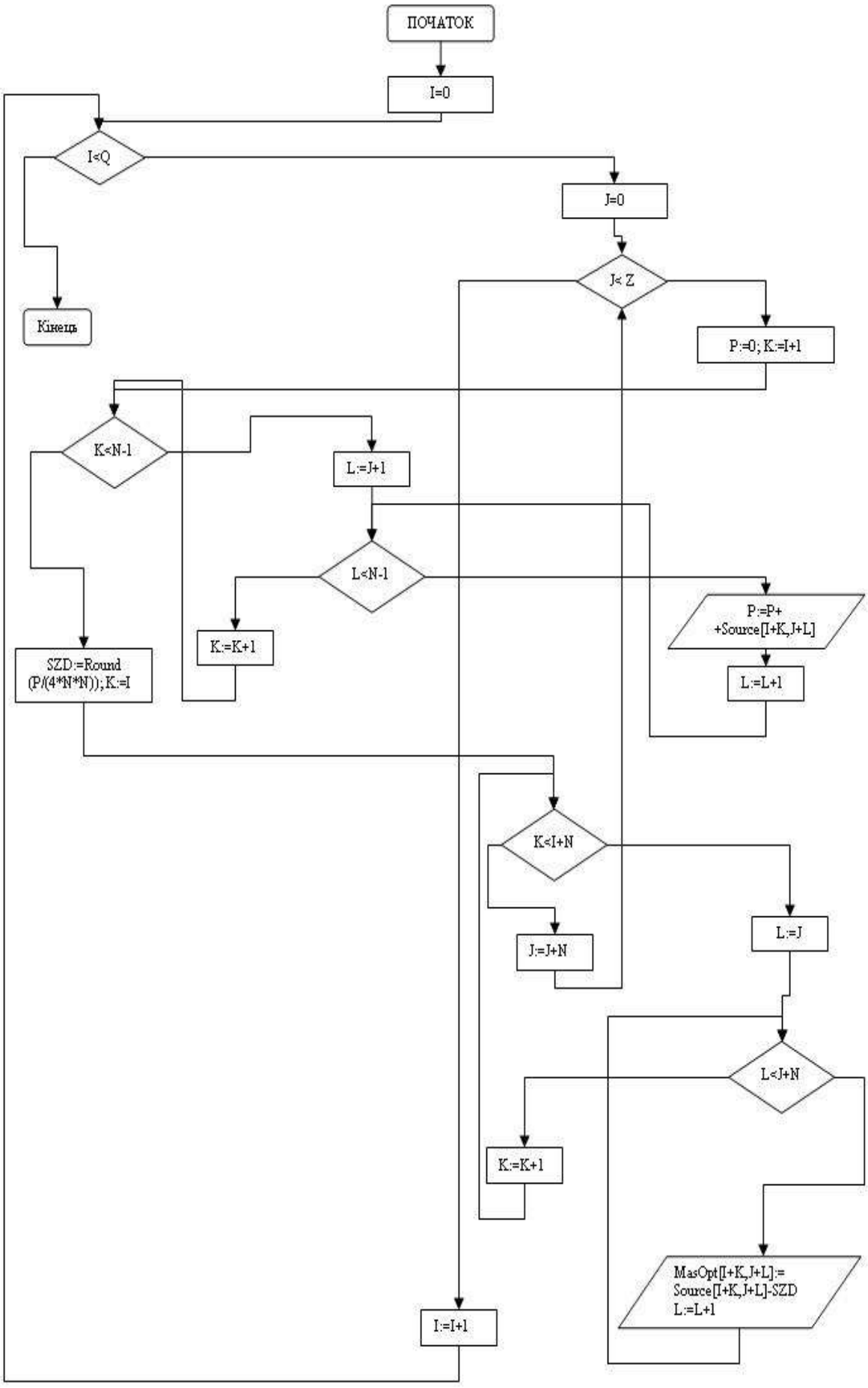


Рис. 2.5 - Блок-схема обробки зображення модифікованим методом

2.3 Висновки до другого розділу

У другому розділі **«Удосконалення табличного методу підвищення часової ефективності фрактального стиснення зображень»** подано опис побудови системи IFS–функцій, висвітлено суть модифікованого методу зменшення обчислювальної складності процесу стиснення зображень.

В результаті аналізу аспектів та особливостей застосування фрактального методу стиснення зображень, можемо зробити висновки, що він здатний забезпечити найкраще співвідношення ступеня стиснення і якості відновленого зображення і має хороші перспективи для подальшого розвитку. Стало зрозуміло, що в процесі перетворення звичайних растрових зображень у фрактальні дані відразу ж реалізуються кілька суттєвих переваг. Наприклад, розмір фізичних даних, які використовуються для запису фрактальних кодів, значно менший розміру початкових растрових даних. Саме ця відмінність фрактальної технології, що називається фрактальним стисненням, викликала найбільший інтерес у сфері формування і відтворення комп'ютерних зображень.

Основним недоліком фрактального методу є низька швидкість стиснення. Вона пов’язана з тим, що для отримання високої якості зображення, для кожного рангового блоку необхідно виконати перебір всіх доменних блоків. Для кожного доменного блоку необхідно виконати не менше восьми афінних перетворень. Ця проблема розв'язана тільки частково. Внаслідок відмічених недоліків, цей метод застосовувався на практиці порівняно рідко.

Пропонуються варіанти зменшення обчислювальної складності та підвищення швидкодії стиснення зображень з використанням фрактальної графіки. Це дозволить зменшити час стиснення зображень без суттєвого погіршення якості.

Розділ 3. Розробка модуля швидкого фрактального пошуку

3.1. Реалізація дерева швидкого пошуку

Застосувавши модифікований метод підвищення часової ефективності для вхідного зображення ми отримаємо перелік доменних блоків з близькими контурами. Це і є завданням фрактального пошуку – знайти найбільш близькі доменні блоки до заданого рангового. Але перед нами стає задача впорядкувати ці доменні блоки та забезпечити швидкий пошук необхідного нам блоку [82-90].

Ця задача реалізується наступним чином:

1. Резервуються 2 масиви. Перший буде мати розрядність яка відповідає найбільшій інтенсивності, яку може мати точка зображення (наприклад 0..255). Другий масив буде мати розрядність, яка відповідає взятій кількості доменних блоків для даного зображення (наприклад 0..64000).

2. Візьмемо за основу масив, який містить всі пікселі зображення, оброблені модифікованим методом. Підрахуємо для першого доменного блоку значення S_k за формулою:

$$S_{k_D} = \frac{1}{4n^2}\left(\sum_{i=0}^{2n-1}\sum_{j=0}^{2n-1}|\, G_D(i,j)\,|\right) \qquad (15)$$

де: k_D - номер доменного блоку, S_{k_D} - детальність доменного блоку, $G_D(i,j)$ - значення, яке приймає точка зображення після застосування модифікованого методу за формулою (14).

3. Занесення номеру доменного блоку в перший масив (далі масив А) у комірку, індекс якої відповідає отриманому значенню S_{k_D}.

4. Вибираються доменні блоки таким чином до тих пір, доки в масиві А не зустрінеться вже зайнята комірка. Це означає, що знайдений доменний блок дуже близький, з точки зору фрактального стиснення, до домену, номер якого зберігається в цій комірці.

5. Якщо зайнята комірка зустрілась, необхідно проаналізувати її вміст (номер домену) та звернутись до другого масиву (надалі масив Б) до комірки з індексом, який дорівнює вмісту знайденої комірки масиву А (рис. 5).

6. Якщо зустрілась вже зайнята комірка в масиві Б, то необхідно аналогічним чином проаналізувати її вміст та звернутись до комірки цього ж масиву (масиву Б) за індексом, який дорівнює вмісту попередньо знайденої комірки і записати туди номер обробленого нами доменного блоку. Тобто в масиві А індекс – це значення знайденого параметру S_{k_D} для конкретного доменного блоку, а значення – це номер знайденого доменного блоку та одночасно показник на комірку масиву Б, де зберігається номер близького до нього наступного доменного блоку. Індекси масиву Б відповідно є показниками на попередній номер близького за параметром S_{k_D} доменного блоку, а значення – чергового доменного блоку для даного значення S_{k_D} та одночасно показниками на наступну комірку, де зберігається наступний близький доменний блок.

7. В якості кінцевого показника можна ввести будь-яке від’ємне число, або 0.

8. Таким чином буде отримана послідовність доменних блоків з близькими, з точки зору фрактального стиснення параметрами. Кроки 1-7 виконуються до тих пір, доки не будуть перебрані усі блоки.

Після того, як побудовані масиви А та Б можна приступити безпосередньо до аналізу рангових блоків. При цьому, як буде показано нижче, за розробленим методом пошук для кожного рангового блоку близького за контуром доменного блоку значно прискорюється. Алгоритм пошуку наступний:

Береться перший ранговий блок та підраховується для нього параметр S_{k_R} - детальність рангового блоку за формулою:

$$S_{k_R} = \frac{1}{n^2}\left(\sum_{i=0}^{n}\sum_{j=0}^{n}|G_R(i,j)|\right) \quad (16)$$

де: k_R - номер рангового блоку,

S_{k_R} - детальність рангового блоку.

Наступним кроком запам’ятовується знайдене значення та звертаємось до масиву A в комірку з індексом, що дорівнює знайденому значенню S_{k_R} і в окремий масив виписується її вміст. Це буде перший доменний блок, що за контуром дуже схожий до даного рангового.

Далі іде звернення до масиву B в комірку, з індексом, що дорівнює номеру знайденого доменного блоку. Вмістом цієї комірки буде номер наступного доменного блоку, що входить до нашої послідовності.

Така операція повторюється до тих пір, доки вмістом комірки не буде деяке фіксоване значення, прийняте за показник кінця послідовності.

Отримана послідовність і є переліком найбільш близьких доменних блоків до даного рангового за контурними характеристиками, яскравістю та контрастністю. Далі виконується класичне фрактальне стиснення. Відмінність лише в тому, що не перебираються всі доменні блоки для даного рангового, а іде лише звернення до знайденої послідовності і проводяться розрахунки вже для конкретних доменних блоків [91-101].

Блок-схема модуля швидкого пошуку представлена на рис. 3.1, де:

I, J, K, L - Змінні типу “Ціле”, використовуються в якості змінних циклів.

Q, Z - Висота та ширина зображення.

N - Висота та ширина рангового блоку.

S - Значення середньої інтенсивності доменного блоку, підраховане за формулою (15).

ABS(F) - Функція, яка залишає абсолютне значення (значення по модулю) виразу F.

MasOpt - Масив, що має розмірність відповідно довжині та ширині початкового зображення та містить значення кожної точки вхідного зображення після застосування алгоритму оптимізації.

ArrayA - Масив А, описаний в попередньому розділі. Містить в якості індексів середні значення інтенсивності доменного блоку, а в якості вмісту комірок – номери перших знайдених доменних блоків з відповідною інтенсивністю.

ArrayB - Масив В, описаний в попередньому розділі. Містить в якості індексів номери “попередніх” близьких доменних блоків, а в якості вмісту комірок – номери наступних “близьких” доменних блоків.

Find - логічна змінна.

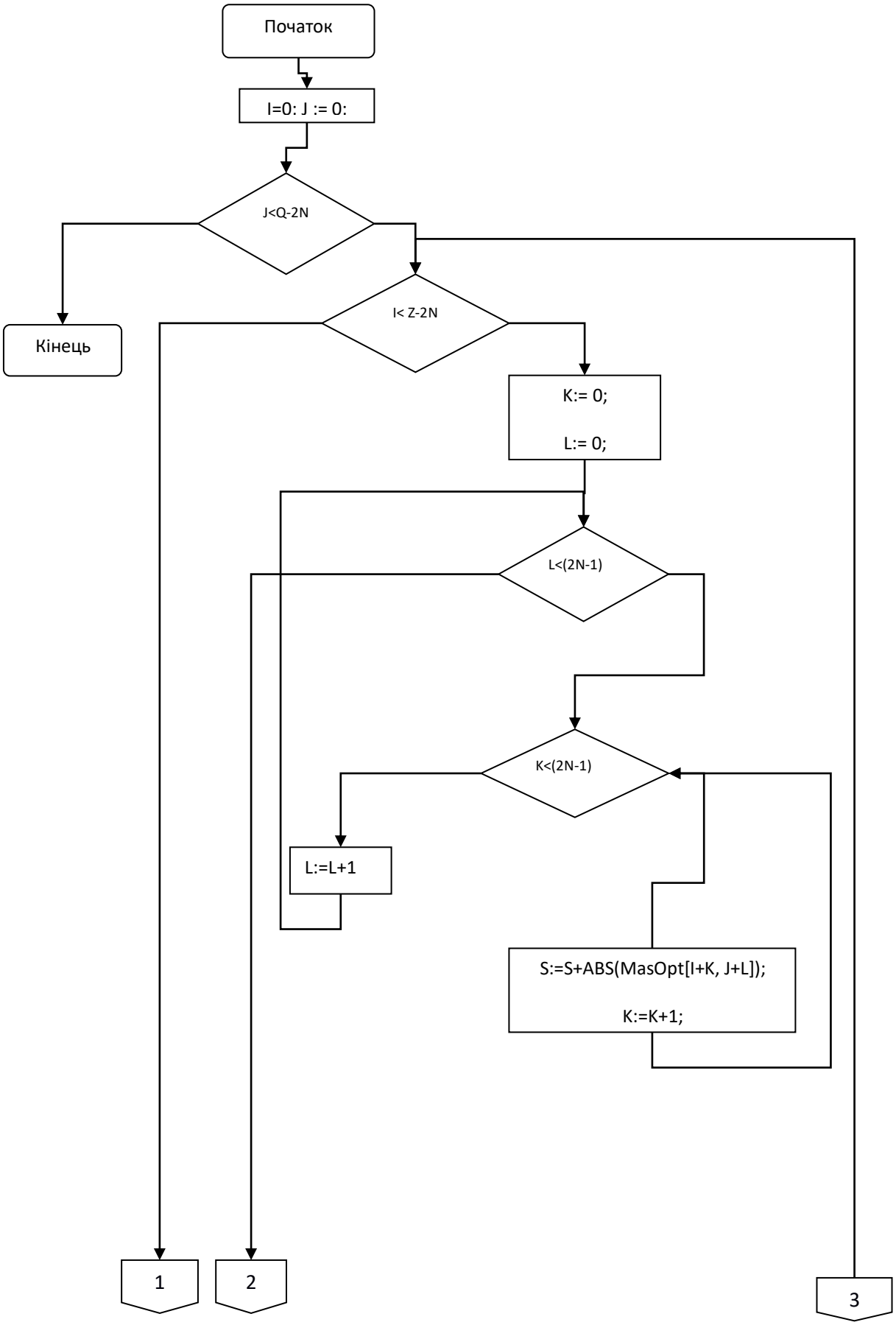


Рис. 3.1 - Блок-схема алгоритму побудови дерева і швидкого пошуку

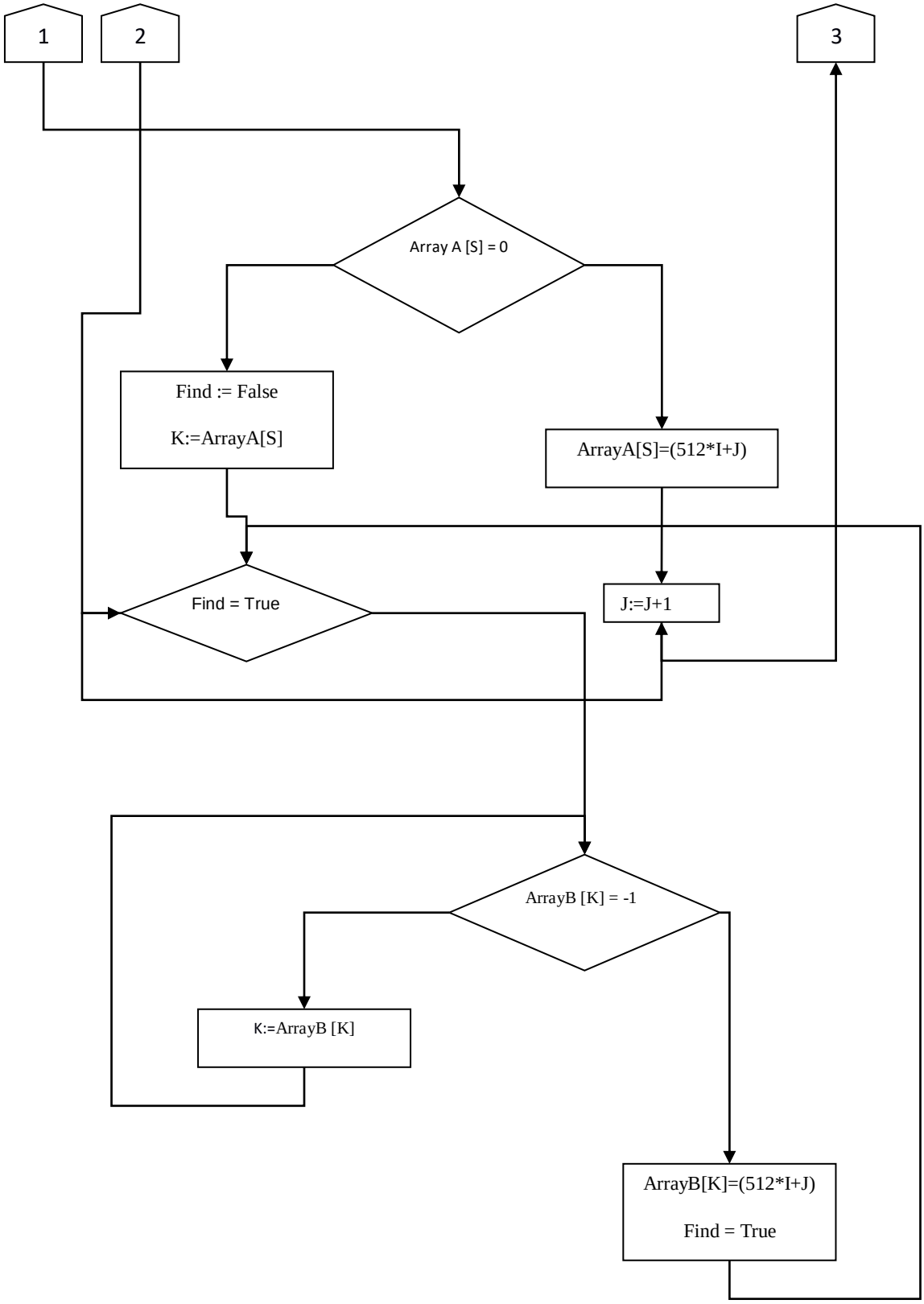


Рис. 3.1, аркуш 2

3.2 Висновки до третього розділу

В третьому розділі **«Розробка модуля швидкого фрактального пошуку»** описана методика пошуку найбільш близьких доменних блоків до заданого рангового та забезпечення умови швидкого фрактального пошуку.

Розглянуті теоретичні і практичні питання розробки і побудови ефективних методів фрактального стиснення зображень. Основна увага приділяється поліпшенню часової ефективності базового методу фрактального стиснення і розширенню горизонтів його практичного застосування. Наведені блок-схеми реалізації процесу обробки зображення модифікованим методом, створення дерева для вибору близьких, з точки зору фрактального стиснення доменних блоків, швидкого їх пошуку, безпосередньо кодування зображення та представлення його у власному форматі FCF (Fractal Coding File) та в стандартному форматі BMP (Bitmap Picture). Ефективність фрактального кодування, з точки зору співвідношення якості зображення і швидкості стиснення, порівнянна з кращими сучасними методами компресії. Це робить фрактальне стиснення найбільш придатним для застосування в різних додатках, де стиснення не вимагається виконувати в реальному часі.

Розділ 4. Розробка та апробація модуля фрактального кодування-декодування зображень

4.1 Розробка модуля стиснення зображень фрактальним методом

Безпосередньо фрактальне стиснення зображень - це фактично останній етап кодування зображення. Він найбільш повільний, порівняно з двома попередніми етапами (обробкою модифікованим методом покращення часової ефективності та побудовою списку доменних блоків), а тому заслуговує на особливу увагу, так як це дослідження спрямоване на покращення побудови системи IFS-функцій при кодуванні зображень фрактальним методом.

Процес кодування зображення фрактальним методом розіб'ємо на декілька етапів:

– зображення розбивається на області, що примикають одна до одної, розміром N x N (рангові області).

– підбирається для кожної рангової області доменна область (з попередньо вибраних та побудованих шляхом використання дерева пошуку, на відміну від класичного фрактального стиснення, коли для кожного рангового блоку перебираються всі доменні блоки), яка після афінних перетворень найбільш точно апроксимує рангову область. На практиці застосовується вісім варіантів відображення одного квадрата в інший з використанням афінних перетворень, які приведені на рис. 2. Це повороти зображення на кути 90,180, 270 (-90) градусів відносно його центра і перетворення симетрії відносно ортогональних осей, які проходять через центр фрагмента перпендикулярно його сторонам.

– точність апроксимації визначається за допомогою середньоквадратичного критерію:

$$F=\sum_{i,j}(Sd_{ij}+O_{ij}-r_{ij})^2 \qquad (17)$$

де :d_{ij} - значення, отримані в результаті усереднення по фрагментам з розмірами 2×2 елементів доменної області, що приводить її розмір до розміру рангової області;

r_{ij} - значення елементів рангової області. Зміщення O_{ij} може бути як константою так і описуватись поліномами першого, другого, третього порядків.

У вихідний файл записуються такі параметри:

- координати доменної області з найменшим значенням F_{min};
- значення для O і S, отримані згідно формул (7, 8);
- номер афінного перетворення.

Опишемо метод фрактального стиснення зображень мовою блок-схем (рис. 4.1), де: I, J, K, L - змінні типу «Ціле», використовуються в якості змінних циклів. Q, Z - висота та ширина зображення. N - висота та ширина рангового блоку. S - значення середньої інтенсивності рангового блоку. ABS(F) - функція, яка залишає абсолютне значення (значення по модулю) виразу F. MasOpt – масив, що має розмірність відповідно довжині та ширині початкового зображення та містить значення кожної точки вхідного зображення після застосування модифікованого методу підвищення часової ефективності.

В блок-схемі також використовується процедура пошуку множини доменних блоків, найбільш близьких до даного рангового з масивів A та B і запис параметрів знайденого доменного блоку в файл на диску.

Розробка модуля декодування зображень, стиснених фрактальним методом є завершальним етапом в розробці програми фрактального стиснення зображень.

Процес фрактального перетворення асиметричний. Тобто, процес декодування не є простою інверсією процедур стиснення і вимагає значно менше часу для його виконання, що дає можливість уже тепер використовувати цей метод для зберігання зображень на компакт-дисках та інших носіях.

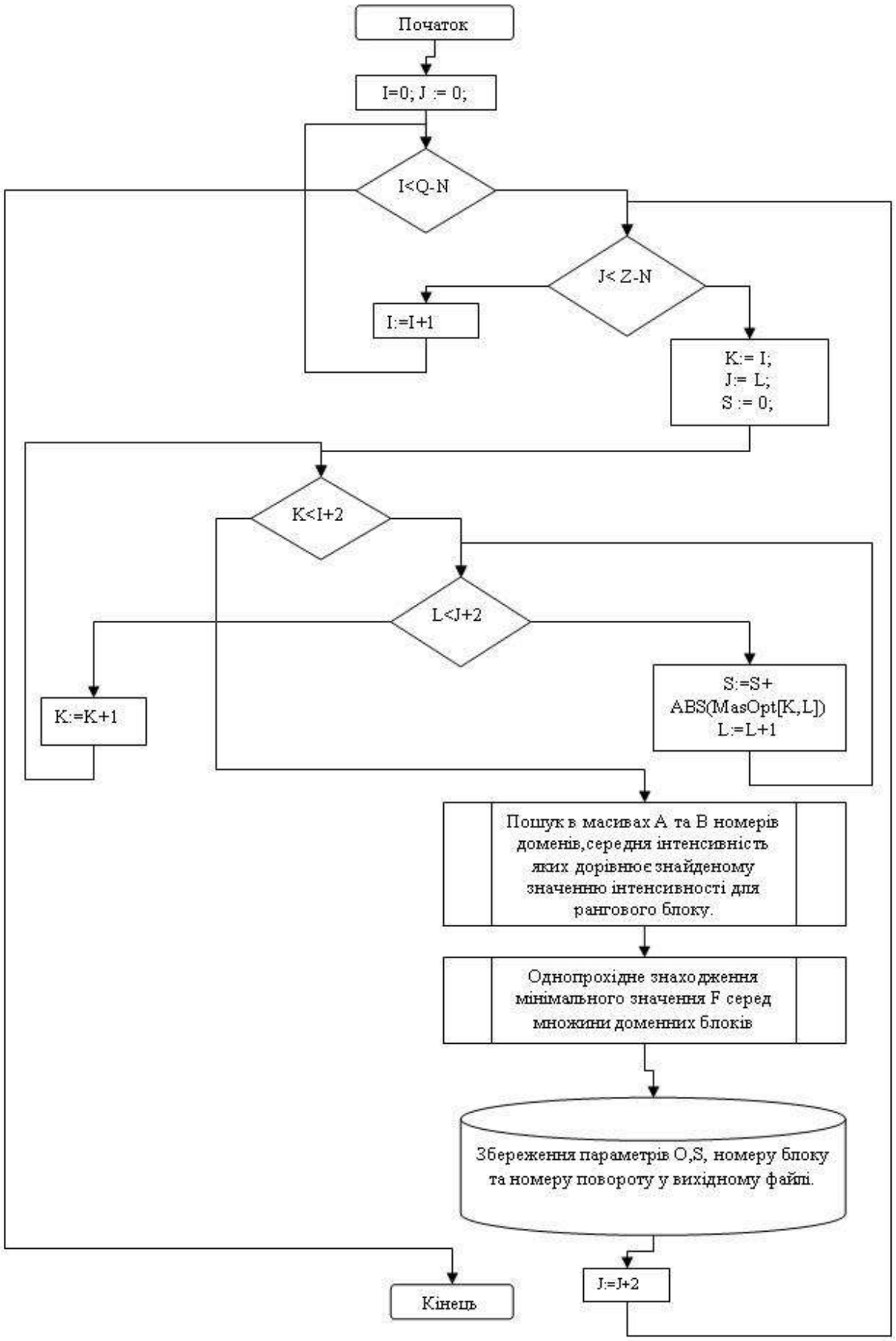


Рис. 4.1 Граф-схема методу фрактального стиснення зображень

4.2 Розробка модуля декопресії зображень фрактальним методом

Декодування стисненого зображення носить ітераційний характер і складається з таких етапів:

1. Створюються два зображення однакового розміру А і Б. Розмір цих зображень не обов'язково рівний розміру початкового зображення, початковий рисунок областей А і В будь-який.

2. Зображення Б розбивається на доменні області у відповідності до процесу стиснення. Зображення А розбиваються на рангові області також у відповідності до процесу стиснення.

3. Кожен ранг зображення А отримується апроксимуючи відповідний домен зображення Б.

4. Міняються зображення А і Б місцями.

5. Повторюються пункти 2-4 доти, поки зображення А і Б не будуть відрізнятися.

Відновлення стисненого зображення носить ітераційний характер, і закінчується після 15-20 ітерацій, тому що подальші ітерації покращення зображення не дають.

На кожній ітерації з файлу формату IFS читається поблочно інформація для кожного рангу та проводяться, згідно з нею, необхідні перетворення. Блок інформації для рангової області містить 5 байт, які в свою чергу включають в себе:

– перший байт містить середнє значення для вибраного рангу;

– наступне слово містить координати апроксимуючого доменного блоку у вигляді:

$$Position = DomainYPos * ImageXSize + DomainXPos.$$

– наступне слово містить інформацію про різницю між доменним та ранговим блоком (приведену до додатного значення шляхом збільшення

на 255, як модуль максимальної від’ємної різниці між точкою домену та рангу) та інформацію про афінне перетворення.

Граф-схема алгоритму програмного модуля декодування зображень приведена на рисунку 3.6, де:

I, J - змінні типу “Ціле”, використовуються в якості змінних циклів.

ImageA - масив, необхідний для відновлення зображення. Розмірність – будь-яка.

ImageB - масив, необхідний для відновлення зображення. Розмірність – будь-яка.

ItaretionCount - лічильник ітерацій.

RangeCount - лічильник рангових блоків.

Read(F,Info) - Процедура, яка виконує читання з файлу, стисненого фрактальним методом, блоки, які містять інформацію про кожний ранговий блок.

Transform(A,BInfo) - Процедура, яка виконує апроксимацію масиву ImageB на основі ImageA у відповідності з даними запису Info.

TMPImage - Тимчасовий масив для зберігання інформації, що міститься або в масиві ImageA або в ImageB.

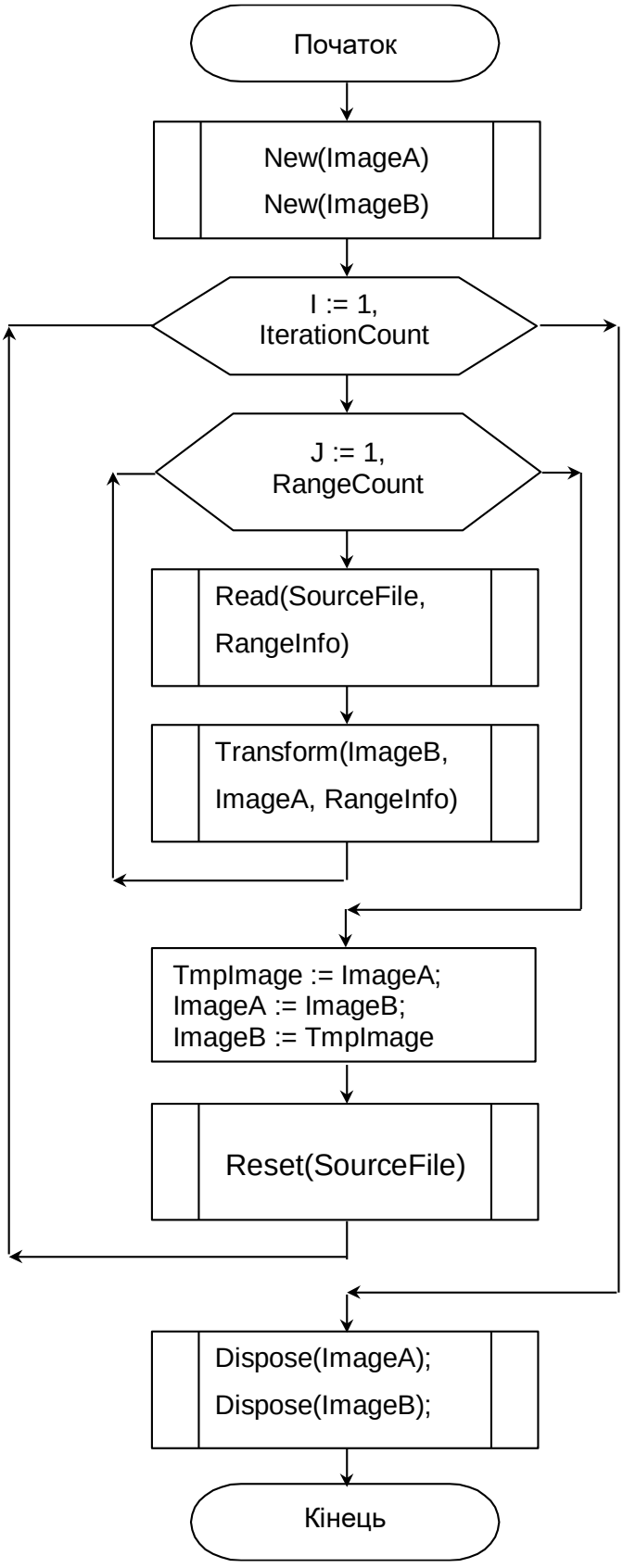


Рисунок 4.2 - Блок-схема модуля декодування зображення, стисненого фрактальним методом

Покрокове відновлення стисненого півтонового зображення модифікованим фрактальним методом показано на рис. 4.3.

на 3-й ітерації	на 10-й ітерації	на 15-й ітерації
		

Рис. 4.3 - Процес декодування півтонового зображення

Покрокове відновлення кольорового зображення модифікованим фрактальним методом показано на рис. 4.4

на 3-й ітерації	на 4-й ітерації	на 5-й ітерації
		
на 6-й ітерації	**на 7-й ітерації**	**на 8-й ітерації**
		

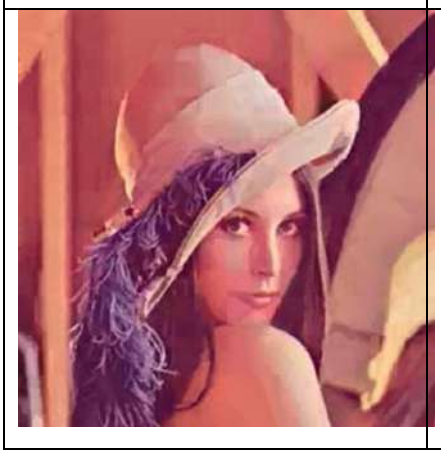
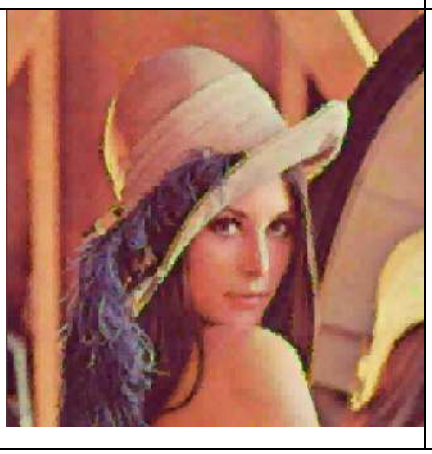
на 10-й ітерації	на 12-й ітерації	на 15-й ітерації
		

Рис. 4.4 – Процес декодування кольорового зображення

4.3 Експериментальні дослідження модифікованого методу фрактального стиснення зображень

Проведемо апробацію методу підвищення часової ефективності фрактального стиснення зображень на прикладі. Для оцінки якості відновленого зображення будемо використовувати метрики PSNR (peak signal-to-noise ratio) і SSIM (structure similarity), які найбільш часто використовується для вимірювання рівня спотворень при стисненні зображень. Величина PSNR - пікове відношення сигналу до шуму, позначає співвідношення між максимумом можливого значення сигналу і потужністю шуму, що спотворює значення сигналу. Дана метрика зазвичай вимірюється в децибелах. Більшому значенню PSNR відповідає краща якість зображення на тлі шуму. SSIM - індекс структурної подібності, є одним з методів вимірювання схожості між двома зображеннями. Тобто повне зіставлення та вимірювання якості на основі не стисненого вихідного зображення.

Таблиця 1 - Півтонове зображення Lenna 512x512 - 8 біт на піксель (див. рис. 4.5)

Метод фр. стиснення	Час стиснення, с	Час декодування, с	Розмір вихідного файлу, кб	Розмір файлу після стиснення, кб	Коефіцієнт стиснення	PSNR, дБ	SSIM
Базовий	7,482	0,094	257	48	5,35	28,247	0,939
Квадродерево	3,851	0,078	257	48	5,35	27,358	0,877
Табличний модифікований	0,374	0,078	257	48	5,35	27,384	0,892

При застосуванні табличного модифікованого методу для кодування півтонових зображень з глибиною кольору 8 біт на піксель швидкість стиснення зросла у 7,482 / 0,374 = 20,01 рази в порівнянні з базовим методом, 3,851 / 0,374 = 10,30 разів в порівнянні з методом квадродерева.

Рис. 4.5 – Відновлення півтонового зображення стисненого фрактальним методом

Зобразимо графічно процес стиснення та декомпресії зображення:

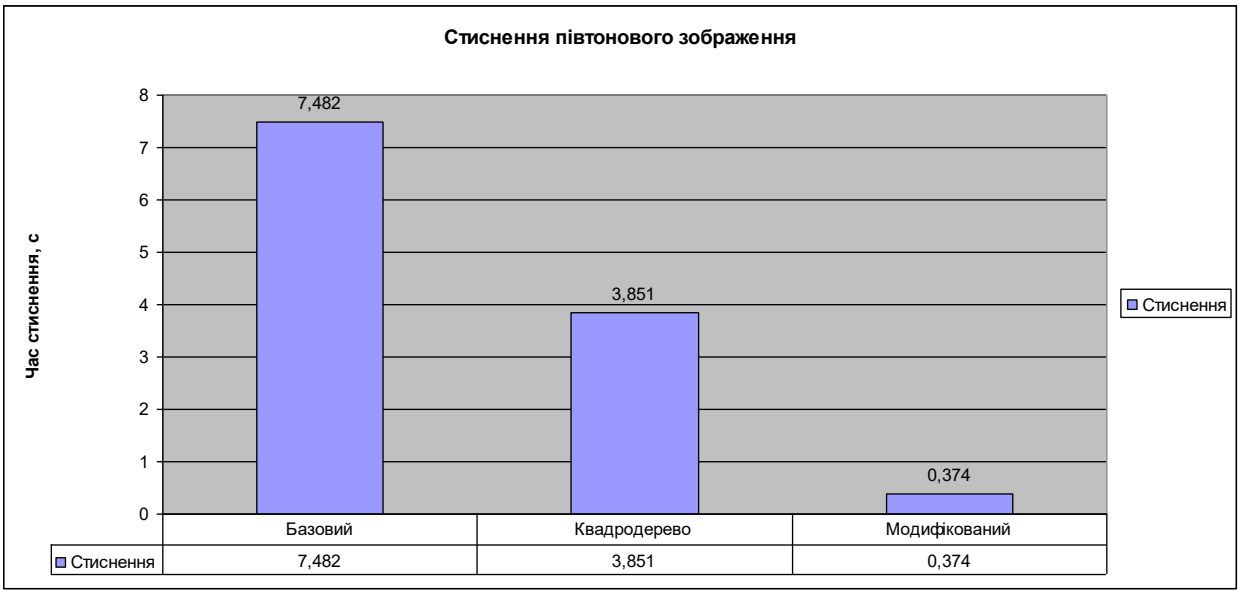


Рис. 4.6 – Стиснення півтонового зображення

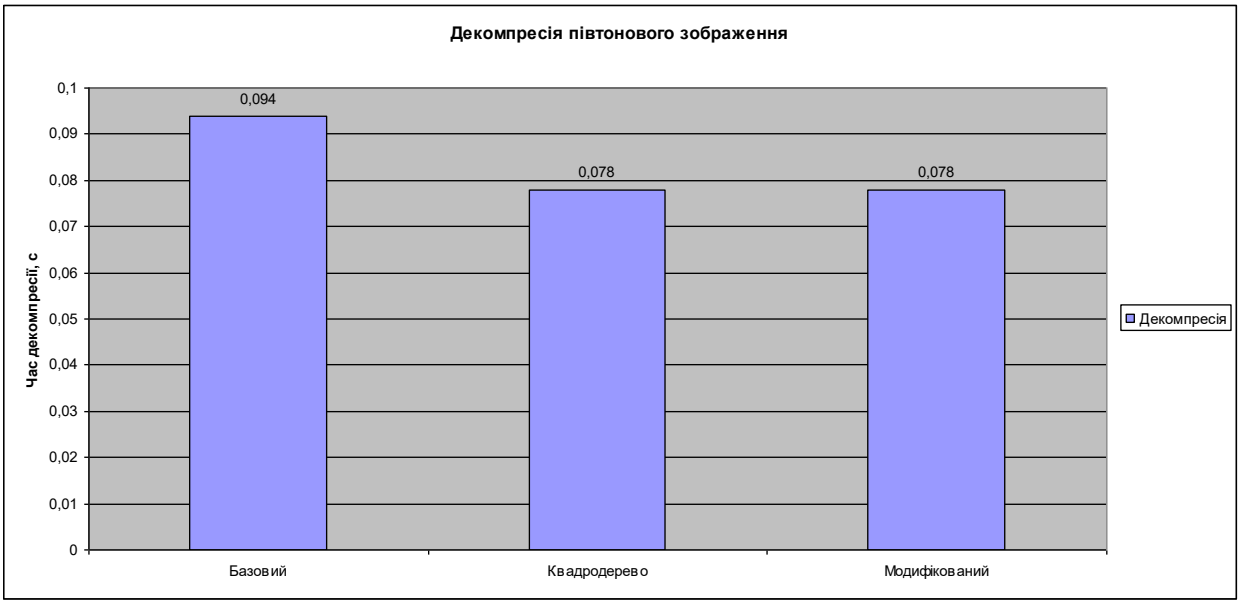


Рис. 4.7 – Декомпресія півтонового зображення

Зобразимо графічно відношення Час стиснення – PSNR:

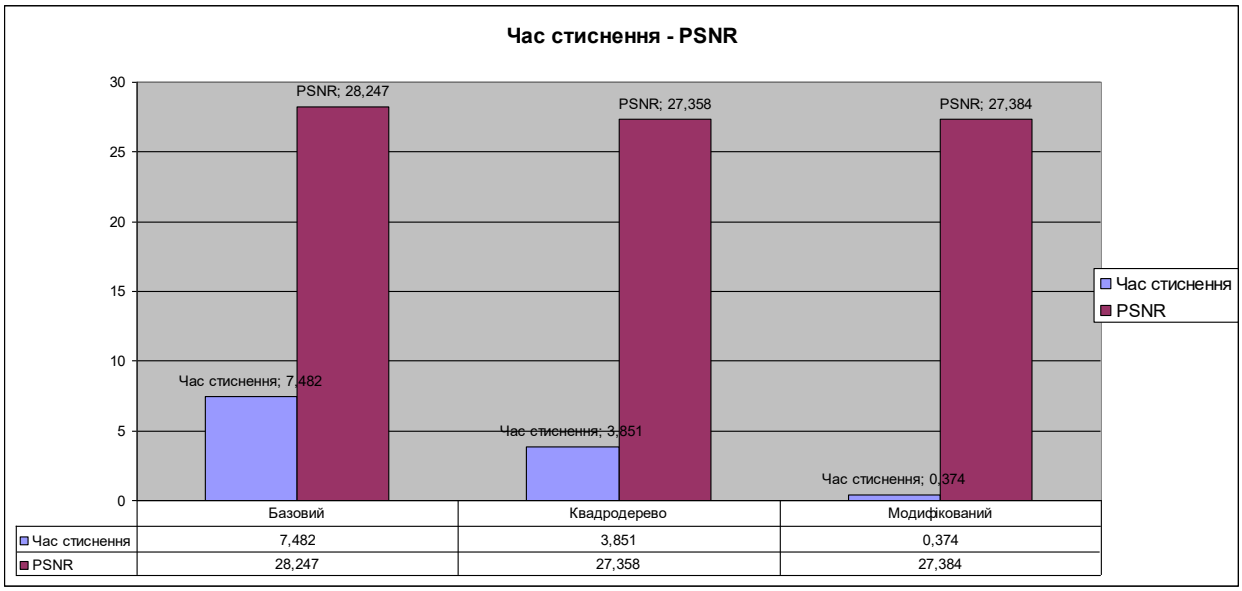


Рис. 4.8 – Час стиснення – PSNR

Зобразимо графічно відношення Час стиснення – SSIM:

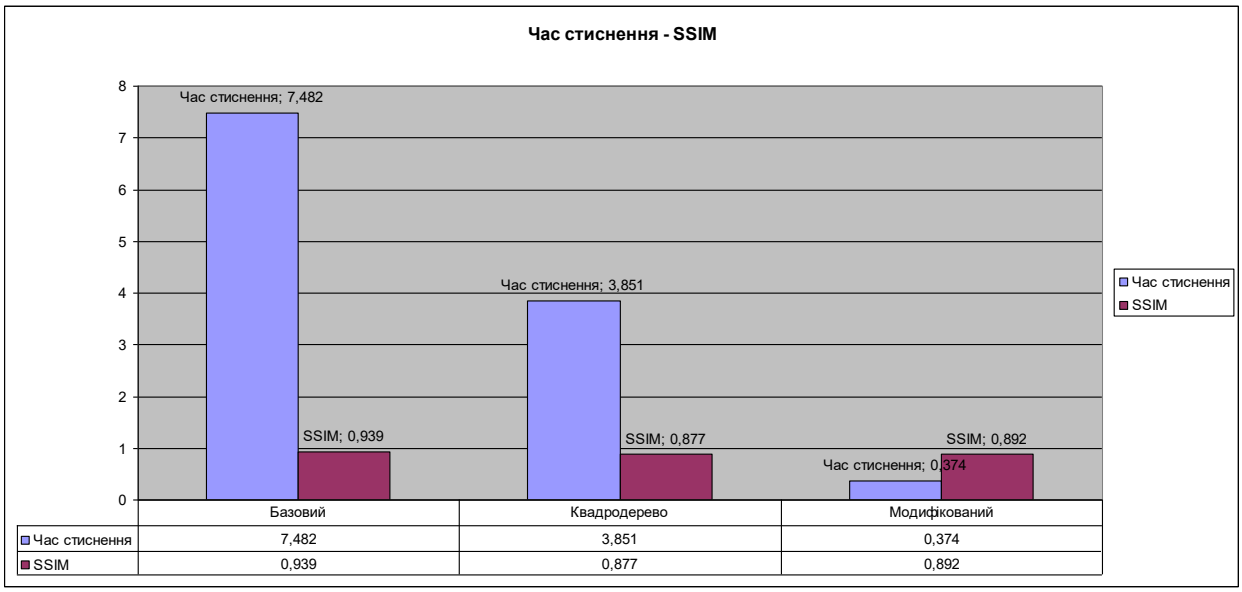


Рис. 4.9 – Час стиснення - SSIM

Таблиця 2 – Час стиснення півтонового зображення модифікованим методом у відповідності з кроком домену та розміром рангового блоку

Крок домену	Розмір рангового блоку		
	4x4	8x8	16x16
	Час стиснення, с		
1	4,020	2,251	1,783
2	0,984	0,564	0,440
3	0,374	0,241	0,196
4	0,252	0,139	0,114
5	0,155	0,086	0,073
6	0,111	0,062	0,049
7	0,083	0,045	0,038
8	0,062	0,037	0,030
9	0,052	0,027	0,021
10	0,038	0,023	0,018
11	0,029	0,022	0,015
12	0,030	0,016	0,013
13	0,026	0,015	0,011
14	0,023	0,016	0,010
15	0,016	0,009	0,011
16	0,015	0,009	0,010
17	0,014	0,008	0,007
18	0,015	0,007	0,006
19	0,013	0,007	0,005
20	0,011	0,006	0,006

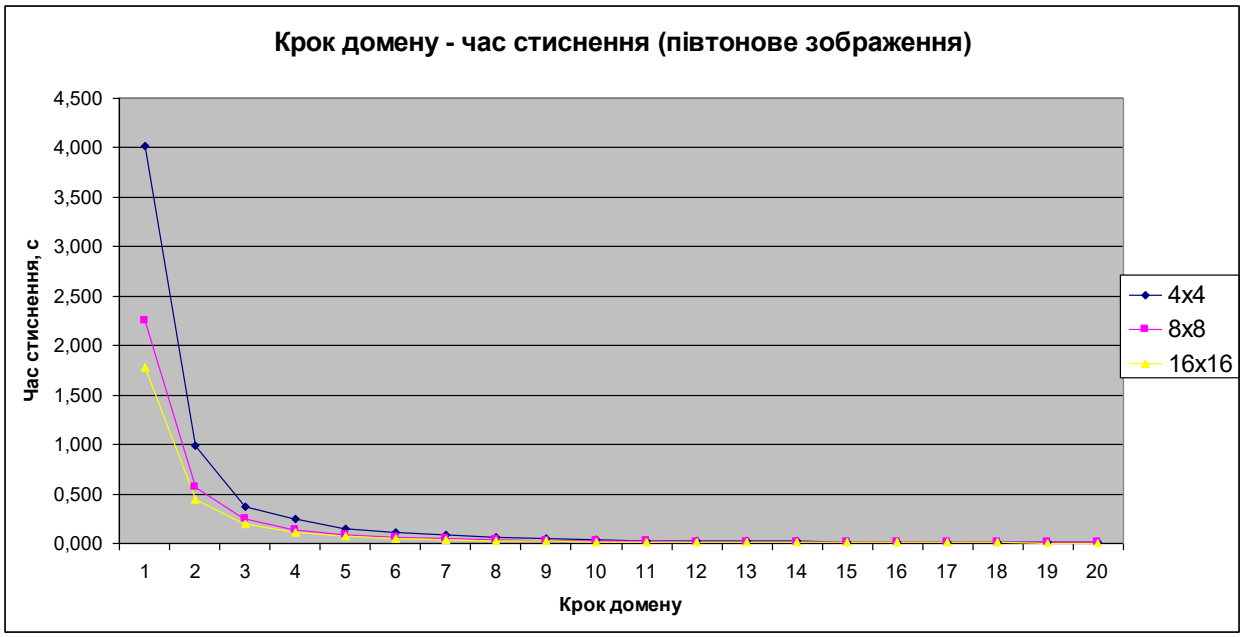


Рис. 4.10 – Залежність часу стиснення від кроку домену для різних розмірів рангового блоку півтонового зображення (4x4, 8x8, 16x16)

Крок домену 1	Крок домену 2	Крок домену 3
		
Крок домену 4	Крок домену 6	Крок домену 8
		
Крок домену 10	Крок домену 15	Крок домену 20
		

Рис. 4.11 – Стиснення півтонового зображення з розміром рангового блоку 4x4

Крок домену 1	Крок домену 2	Крок домену 3
		
Крок домену 4	Крок домену 6	Крок домену 8
		
Крок домену 10	Крок домену 15	Крок домену 20
		

Рис. 4.12 – Стиснення півтонового зображення з розміром рангового блоку 8x8

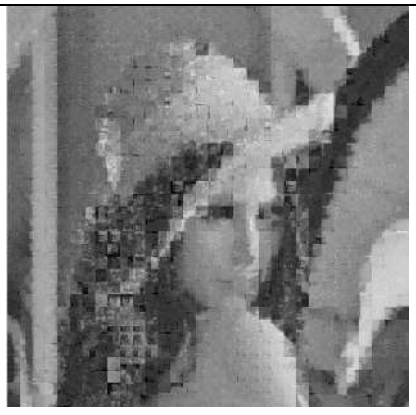
Крок домену 1	Крок домену 2	Крок домену 3
		
Крок домену 4	Крок домену 6	Крок домену 8
		
Крок домену 10	Крок домену 15	Крок домену 20
		

Рис. 4.13 – Стиснення півтонового зображення з розміром рангового блоку 16x16

Порівняємо півтонові зображення з однаковим кроком домену, але з різними розмірами рангових блоків.

4x4	8x8	16x16
Крок домену 1		
		
Крок домену 3		
		
Крок домену 6		
		
Крок домену 8		
		

Рис. 4.14 – Стиснення півтонового зображення з однаковим кроком домену, але з різними розмірами рангових блоків.

Таблиця 3 - Кольорове зображення Lenna 512x512 - 24 біт на піксель (див. рис. 4.15)

Метод фр. стиснення	Час стиснення, с	Час декодування, с	Розмір вихідного файлу, кб	Розмір файлу після стиснення, кб	Коефіцієнт стиснення	PSNR, дБ	SSIM
Базовий	30,175	0,044	768	45	17,07	27,223	0,755
Квадродерево	13,542	0,078	768	48	16	27,557	0,894
Табличний модифікований	1,203	0,047	768	48	16	26,435	0,712

При застосуванні модифікованого методу для кодування кольорових зображень з глибиною кольору 24 біт на піксель швидкість стиснення зросла у

30,175 / 1,203 = 25,08 разів в порівнянні з базовим методом,

13,542 / 1,203 = 11,26 разів в порівнянні з методом квадродерева.

Початкове зображення	Відновлене зображення базовим методом
	
Відновлене зображення методом квадродерева	Відновлене зображення модифікованим методом
	

Рис. 4.15 – Відновлення кольорового зображення стисненого фрактальним методом

Зобразимо графічно процес стиснення та декомпресії кольорового зображення:

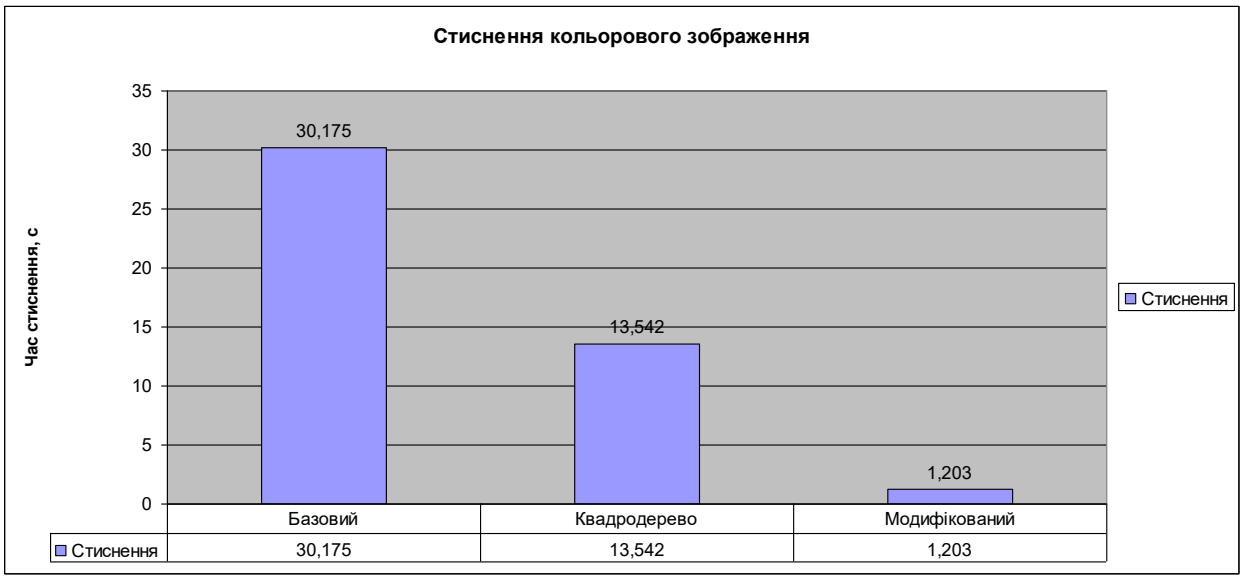


Рис. 4.16 – Час тиснення кольорового зображення

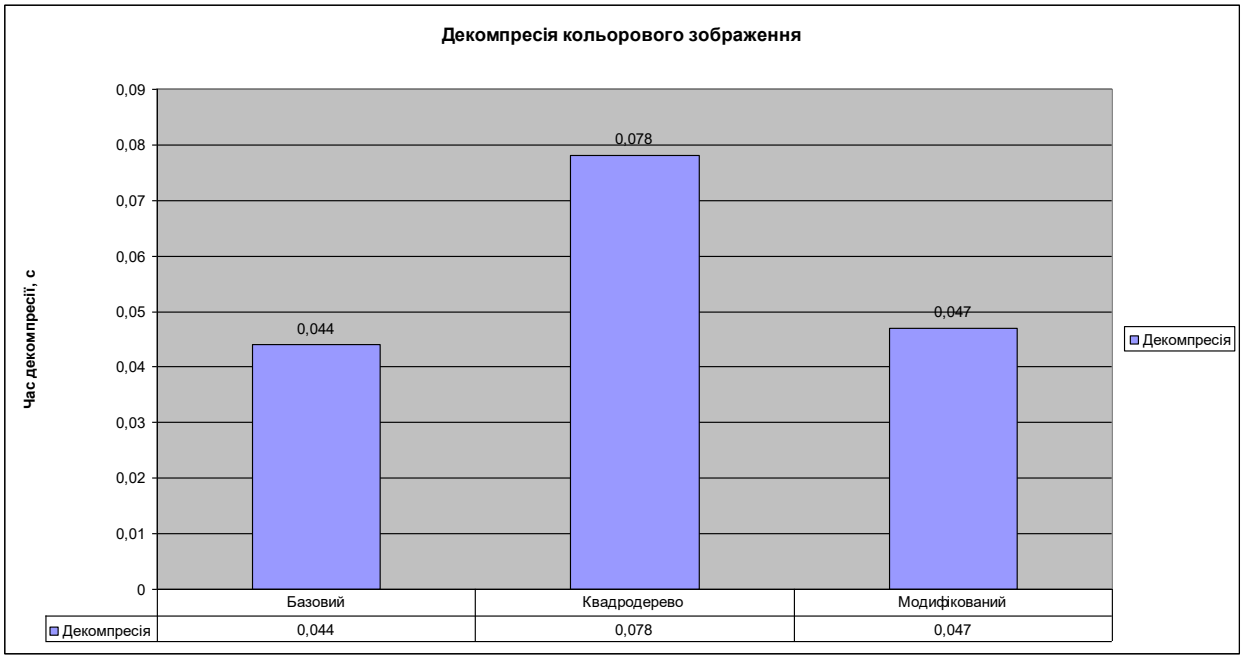


Рис. 4.17 –Час декомпресії кольорового зображення

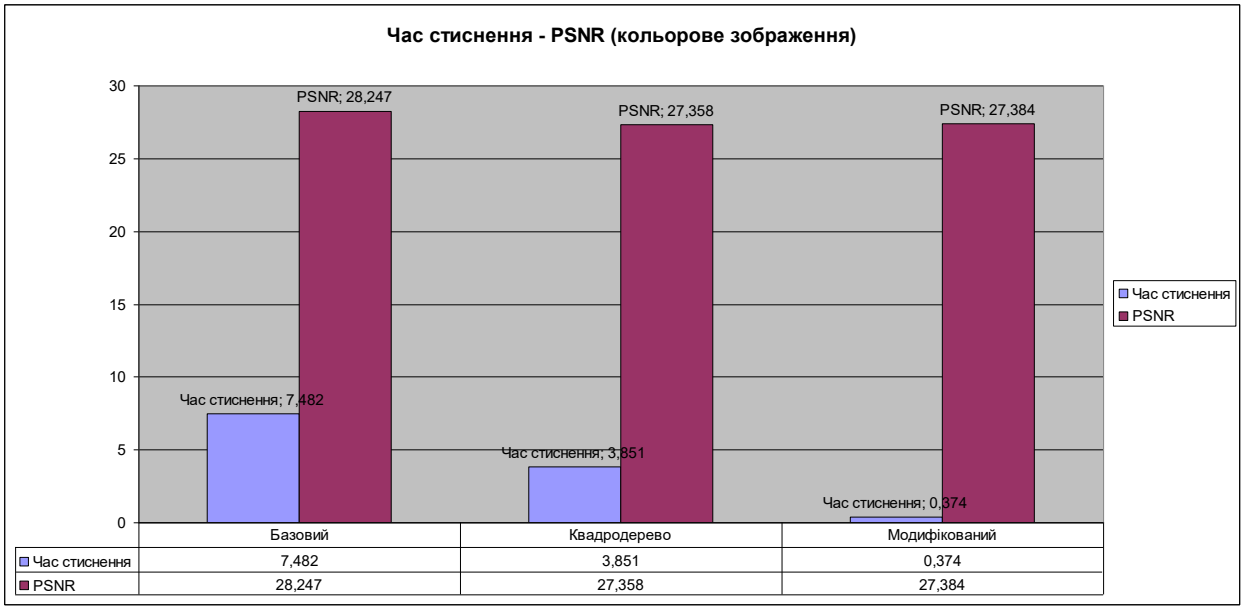


Рис. 4.18 – Час стиснення - PSNR (кольорове зображення)

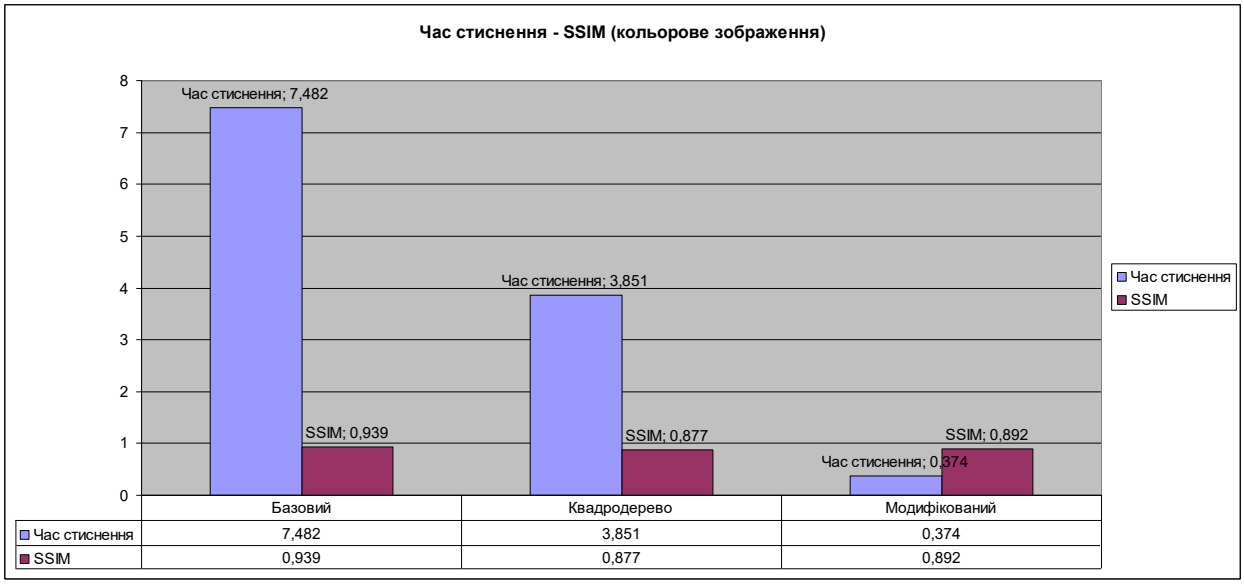


Рис. 4.19 – Час стиснення - SSIM (кольорове зображення)

Таблиця 4 – Час стиснення кольорового зображення модифікованим методом у відповідності з кроком домену та розміром рангового блоку

Крок домену	Розмір рангового блоку		
	4x4	8x8	16x16
	Час стиснення, с		
1	17,740	12,157	10,531
2	4,368	2,846	2,589
3	1,941	1,336	1,133
4	1,203	0,715	0,649
5	0,688	0,448	0,410
6	0,488	0,319	0,291
7	0,355	0,230	0,214
8	0,262	0,183	0,166
9	0,212	0,141	0,124
10	0,173	0,111	0,105
11	0,139	0,094	0,083
12	0,123	0,080	0,073
13	0,103	0,071	0,061
14	0,093	0,058	0,056
15	0,075	0,050	0,048
16	0,065	0,045	0,044
17	0,053	0,038	0,037
18	0,055	0,035	0,034
19	0,048	0,034	0,028
20	0,042	0,028	0,028

Рис. 4.20 – Залежність часу стиснення від кроку домену для різних розмірів рангового блоку кольорового зображення (4x4, 8x8, 16x16)

Крок домену 1	Крок домену 2	Крок домену 3
		
Крок домену 4	Крок домену 6	Крок домену 8
		
Крок домену 10	Крок домену 15	Крок домену 20
		

Рис. 4.21 – Стиснення кольорового зображення з розміром рангового блоку 4x4

Крок домену 1	Крок домену 2	Крок домену 3
		
Крок домену 4	Крок домену 6	Крок домену 8
		
Крок домену 10	Крок домену 15	Крок домену 20
		

Рис. 4.22 – Стиснення кольорового зображення з розміром рангового блоку 8x8

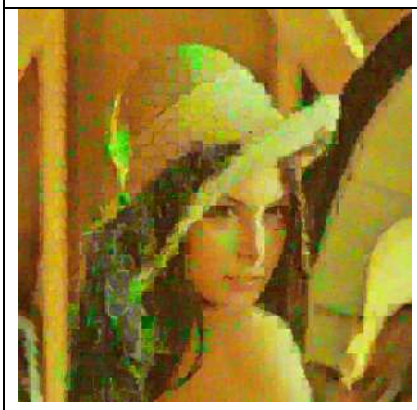
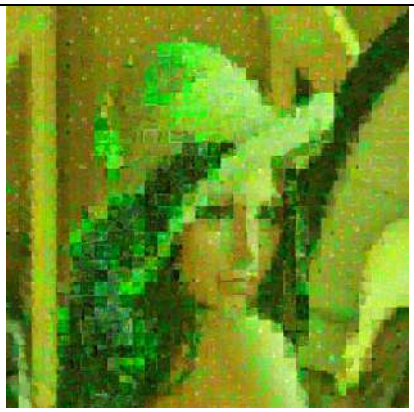
Крок домену 1	Крок домену 2	Крок домену 3
		
Крок домену 4	Крок домену 6	Крок домену 8
		
Крок домену 10	Крок домену 15	Крок домену 20
		

Рис. 4.23 – Стиснення кольорового зображення з розміром рангового блоку 16x16

Таблиця 5 – Час стиснення півтонового зображення базовим та модифікованим методами у відповідності з кроком домену та розміром рангового блоку

Крок домену	Розмір рангового блоку					
	4x4 базовий	4x4 модифікований	8x8 базовий	8x8 модифікований	16x16 базовий	16x16 модифікований
	Час стиснення, с					
1	9,275	4,020	8,686	2,251	8,491	1,783
2	2,316	0,984	2,183	0,564	2,122	0,440
3	1,017	0,374	0,958	0,241	0,933	0,196
4	0,597	0,252	0,540	0,139	0,537	0,114
5	0,380	0,155	0,348	0,086	0,339	0,073
6	0,260	0,111	0,237	0,062	0,229	0,049
7	0,188	0,083	0,173	0,045	0,187	0,038
8	0,149	0,062	0,137	0,037	0,134	0,030
9	0,115	0,052	0,105	0,027	0,103	0,021
10	0,092	0,038	0,085	0,023	0,081	0,018
11	0,078	0,029	0,072	0,022	0,069	0,015
12	0,065	0,030	0,059	0,016	0,058	0,013
13	0,054	0,026	0,049	0,015	0,048	0,011
14	0,048	0,023	0,044	0,016	0,043	0,010
15	0,042	0,016	0,038	0,009	0,037	0,011
16	0,039	0,015	0,035	0,009	0,034	0,010
17	0,035	0,014	0,030	0,008	0,030	0,007
18	0,029	0,015	0,027	0,007	0,026	0,006
19	0,026	0,013	0,023	0,007	0,022	0,005
20	0,021	0,011	0,019	0,006	0,019	0,006

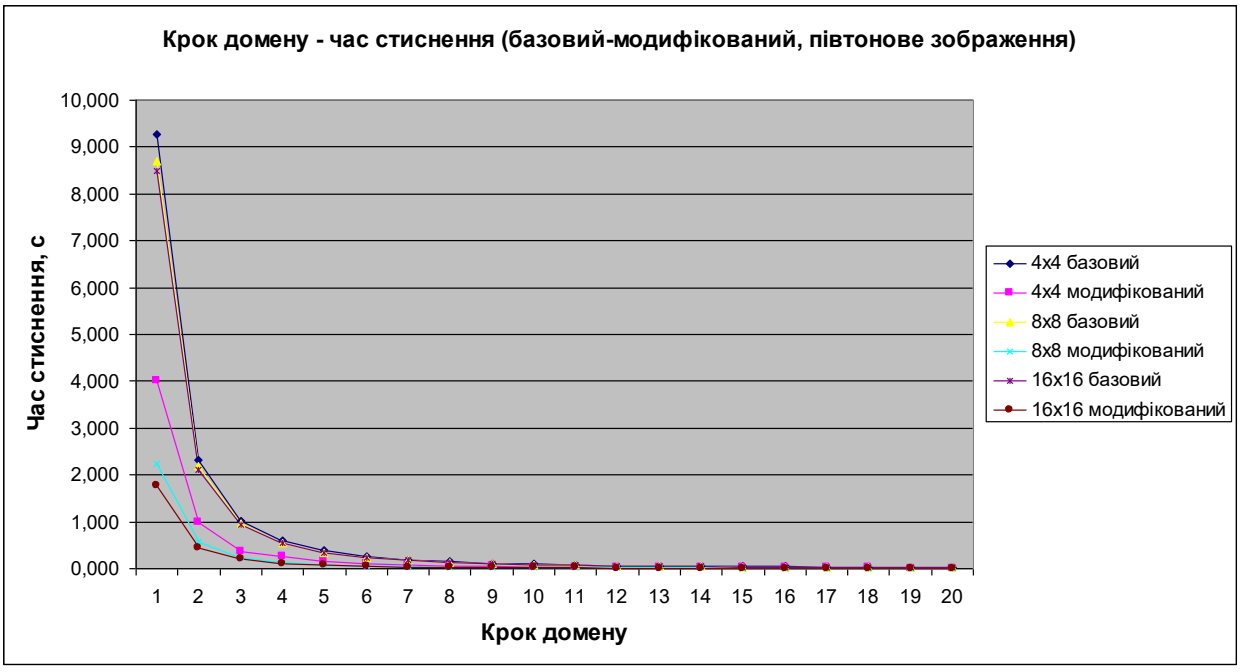


Рис. 4.24 – Порівняння методів стиснення для півтонового зображення

4.4 Висновки до четвертого розділу

Четвертий розділ **«Розробка та апробація модуля фрактального кодування-декодування зображень»** описує процес програмної реалізації та апробації методу підвищення часової ефективності фрактального стиснення зображень. Наводяться метрики для оцінювання якості відновленого зображення.

Виконано програмну реалізацію для виконання стиснення зображень фрактальним методом, яке відповідає вимогам сучасного інтерфейсу. Наведені часові характеристики процесу стиснення зображення фрактальним методом та проведені аналогії з існуючими програмними еквівалентами. Програмний варіант, який розроблено у середовищі об’єктно-орієнтованого візуального програмування, дозволяє кодувати зображення, первинний формат яких містить кольорову палітру і має глибину кольору 24 біт на піксель. Також є можливість кодувати зображення, первинний формат яких містить палітру, яка складається з 256 відтінків сірого, тобто має глибину кольору 8 біт на піксель.

Продемонстровано часові характеристики стиснення зображень, які значно випереджають відомі еквівалентні програмні зразки. Впроваджено принципово новий метод вибору доменних блоків зображення, їх сортування та швидкий пошук.

Як вхідні дані для кодування програма використовує зображення представлені у форматі BMP (Bitmap Picture), а після кодування створює новий файл, у якому зображення представляється у власному форматі FCF (Fractal Coding File). При бажанні користувач може конвертувати файл, стиснений фрактальним методом, з формату FCF у формат BMP.

ВИСНОВКИ

У дисертаційній роботі розв’язано важливу науково-прикладну задачу, яка полягає в удосконаленні методів підвищення часової ефективності фрактального стиснення зображень. Продемонстрована актуальність підвищення часової ефективності фрактального стиснення, як перспективного підходу до стиснення зображень. Проведено аналітичний огляд інших робіт за даною тематикою, у ході якого виявлені можливості покращення часових характеристик фрактальних методів стиснення. Розглянута математична модель, система ітерованих функцій, яка використовується при фрактальному стисненні зображень.

Експериментально показано, що авторський метод здатний забезпечити краще співвідношення ступеня стиснення і якості відновленого зображення. В процесі перетворення цим методом звичайних растрових зображень у фрактальні дані, відразу ж реалізуються кілька суттєвих переваг. Наприклад, можливість масштабувати фрактальне зображення без появи артефактів і втрати деталей, як це характерно для растрових зображень. Цей процес не залежить від розподільної здатності початкового зображення і масштаб обмежується тільки об'ємом вільної пам'яті комп'ютера. Ще одна перевага полягає в тому, що розмір фізичних даних, які використовуються для запису фрактальних кодів, значно менший розміру початкових растрових даних.

Процес фрактального перетворення асиметричний. Тобто відтворення зображення відбувається набагато швидше ніж стиснення. Процес ітераційного декодування не є простою інверсією процедур стиснення і вимагає значно менше часу для його виконання, що дає можливість використовувати цей метод для компактного зберігання зображень на різних носіях інформації.

В роботі отримані наступні результати:

1. Проведено аналіз способів скорочення обчислювальної складності та мінімізації часових витрат на фрактальну компресію зображень.

2. Створено новий швидкодіючий модифікований метод фрактальної компресії, що базується на оцінці детальності доменних і рангових блоків.

3. Експериментальне дослідження модифікованого методу на півтонових зображеннях показало, що його застосування для обробки таких зображень, дозволяє підвищити швидкість стиснення в 10-20 разів.

4. Проведено експериментальну перевірку модифікованого методу на стандартних кольорових зображеннях, яка підтвердила справедливість теоретичних результатів і висновків, а також показала значний виграш (більш як в 25 разів) по швидкості стиснення у порівнянні з базовим методом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бенуа, Б. Мандельброт. Фрактальная геометрия природы / Б. Бенуа. – М.: Институт компьютерных исследований, 2002. – 656 с.
2. Барнсли М., Ансон Л. Фрактальное сжатие изображений // Мир ПК. 1992. - N 10. - С. 52-58.
3. Птачек М. Цифровое телевидение. Теория и техника: пер. с чешск. - М.: Радио и связь, 1990. 528 с.
4. Кроновер Р. Фракталы и хаос в динамических системах : учеб.пособие Р. Кроновер – М. : Техносфера, 2006. – 488 с.
5. Цифровое кодирование телевизионных изображений / Под ред. И.И. Цуккермана. - М.: Радио и связь, 1981. -240 с.
6. Харатишвили Н.Г. Цифровое кодирование с предсказанием непрерывных сигналов. - М.: Радио и связь, 1986. - С.3-25.
7. Кроновер, Р.М. Фракталы и хаос в динамических системах. Основы теории. М.: Постмаркет, 2000. – 352 с.
8. Бейбалаев В.Д. Математические модели неравновесных процессов в средах с фрактальной структурой : автореф. дис. канд. физ.-мат. наук. – Махачкала, 2009. – 18 с.
9. Мокрый В.Ю. Учебный модуль «Методы, алгоритмы и технологии сжатия данных»: [Електронний ресурс] - Режим доступу: https://sites.google.com/site/szatieinformacii/. - Назва з екрану.
10. Дубовиков М.М. Индекс вариации и его приложение к анализу фрактальных структур // М.М. Дубовиков, Н.В. Старченко. Научный альманах «Гордон», изд-во «Поматур», М. 2005.
11. Ватолин Д., Ратушняк А., Смирнов М, Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. — М.: ДИАЛОГ-МИФИ, 2003. -384 с.

12. Майданюк В.П. Методи і засоби комп’ютерних інформаційних технологій. Кодування зображень. Навчальний посібник. - Вінниця: ВДТУ, 2001. - 65 с.
13. Ватолин. Д.С. Фрактальное сжатие изображений. // ComputerWorld-Россия. 1996. - № 6 (23). - с.21-28.
14. Артюшенко В.М., Шелухин О.И., Афонин М.Ю. Цифровое сжатие видеоинформации и звука: Учебное пособие / Под ред. В.М. Артюшенко. – М.: Издательско-торговая корпорация «Дашков и К°», 2003. – 426 с.
15. Мюррей, Д. Энциклопедия форматов графических файлов / Д. Мюррей, У. ван Райпер; пер. с англ. – К.: Издательская группа BHV, 1997. – 672 с.
16. Красильников, Н. Н. Цифровая обработка 2D- и 3D-изображений: уч. пос. / Н. Н. Красильников. – СПб.: БХВ-Петербург, 2011. – 608 с.
17. Ватолин, Д. С. Использование ДКП для ускорения фрактального сжатия изображений / Д. С. Ватолин // Программирование. – 1999. – № 3. – С. 51–57.
18. Карпов, П. М. Быстрый фрактальный алгоритм сжатия изображений / П. М. Карпов. – Научная сессия МИФИ, 2006. – Т. 15.
19. Шабаршин, А. А. Метод фрактального сжатия изображений / А. А. Шабаршин // Научные школы УПИ-УГТУ. – 1997. – № 1. – С. 70–82.
20. Винокуров, С. В. Методы повышения временной эффективности алгоритмов фрактального сжатия изображений: матер. конф. // Фундаментальные и прикладные проблемы приборостроения, информатики и экономики, сборник Информатика, 2005. – С. 53–59.
21. Винокуров, С. В. Эффективный алгоритм фрактального сжатия изображений с использованием пространственно-чувствительного хеширования / С. В. Винокуров // Открытое образование. – 2006. – Т. 4, № 57. – С. 62–70.
22. Осокин, А. Н. Исследование возможности распараллеливания процесса фрактального сжатия изображений: сб. трудов VIII Всеросс. научно-практ. конф. / А. Н. Осокин, М. П. Шарабайко // Молодежь и современные информационные технологии. – Томск, 2010. – Т. 1, Ч. 2. – С. 212–213.

23. Краснов, М. Л. Интегральные уравнения: введение в теорию / М. Л. Краснов. – М.: Наука, 1975. – 302 с.

24. Хуанг, Т. С. Быстрые алгоритмы в цифровой обработке изображений / Т. С. Хуанг и др.; под ред. Т. С. Хуанга: пер. с англ. – М.: Радио и связь, 1984. – 224 с.

25. Кунт М. Блочное кодирование графических материалов. Обзор / М. Кунт, О. Джонсон // ТИИЭР. – 1980. – Т. 68, № 7. – С. 21–40.

26. Майданюк, В. П. та ін. Кодування зображень в комп'ютерних системах / В. П. Майданюк та ін. – К.,1996. – 16 с.

27. Илюшин С. В., Свет С. Д. Фрактальное сжатие телемедицинских изображений // Электросвязь. – 2009. – № 4. – С. 36–40.

28. Fisher Y. “Fractal Image Compression: Theory and Application to Digital Images”, Springer Verlag, 1995- 452 p.

29. Ватолин Д. С. Алгоритмы сжатия изображений – М.: МГУ, 1999.- 231с.

30. Миано Дж. Форматы и алгоритмы сжатия изображений в действии. Учеб. пособ. – М.: Триумф, 2003. – 336с.

31. Уэлстид С. Фракталы и вейвлеты для сжатия изображений в действии. Учеб. пособ. - М.: Триумф, 2003. – 320с.

32. Корриган Дж. Компьютерная графика: Секреты и решения: Пер. С англ. - М.: Энтроп, 1995.

33. Прэтт У. Цифровая обработка изображений. – Москва: Мир, 1982.

34. Форсайт Д., Понс Д. Компьютерное зрение. Современный подход.- Вильямс, Москва, 2004. - 928 с.

35. Бредихин Д. Ю. Сжатие графики без потерь качества [Електронний ресурс] / Д. Ю. Бредихин. – 2004. – Режим доступу: http://www.compression.ru/download/articles/i_lless/bredikhin_2004_lossless_image_compression_doc.rar. – Назва зекрану.

36. Блінова Т.О., Порєв В.М. Комп’ютерна графіка / За ред. В.М.Горєва. – К.: Видавництво “Юніор”, 2004. – 456 с., іл.

37. С.В.Глушаков, Г.А.Крабе. Компьютерная графика, Харьков, 2002.

38. М. Н. Петров, В. П. Молочков. Компьютерная графика. Учебник. Серия: Учебник для вузов. Издательство Питер, 2004 г., 812 стр.

39. Марк Кэмпбелл. Компьютерная графика. The Complete Idiot's Guide ti Computer Illustration. Серия: The Complete Idiot's Guide. Периодическое издание. Издательство АСТ, 2007 г.

40. П. Я. Пантюхин, А. В. Быков, А. В. Репинская. Компьютерная графика. В 2 частях. Серия: Профессиональное образование. Издательство Инфра-М, 2007 г. 88 стр.

41. Шикин Т.В., Борисков А.В. "Компьютерная графика: динамика и реалистические изображения". - М.: "Диалог-МИФИ", 1996 г.

42. Н. В. Максимов, И. И. Попов, Т. Л. Партыка. Архитектура ЭВМ и вычислительные системы. Серия: Профессиональное образование. Издательство Форум, 2008 г.

43. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. ; пер. с англ. Л. И. Рубанова, П. А. Чочиа – М. : Техносфера, 2012. – 1104 с.

44. Кнут Д. Е. Искусство программирования для ЭВМ. Т. 3: Сортировка и поиск / Д. Е. Кнут. – 2-е изд. – М.: Вильямс, 2008. – 824 с.

45. Лидовский В. В. Теория информации : Учеб. пособ. / В. В. Лидовский. – М.: Компания Спутник+, 2004. – 111 с.

46. Методы компьютерной обработки изображений / [М. В. Гашников, Н. И. Глумов, Н. Ю. Ильясов и др.] ; Под ред. В. А. Сойфера. – 2-е изд., испр. – М.: ФИЗМАТЛИТ, 2003. – 784 с.

47. Ратушняк О. А. Алгоритмы сжатия изображений без потерь с помощью сортировки параллельных блоков / О. А. Ратушняк // Тезисы докладов конференции молодых ученых по математике, мат. моделированию и информатике. – Новосибирск, 2001. – С. 48-49.

48. Ратушняк О. А. Методы сжатия данных без потерь с помощью сортировки параллельных блоков / О. А. Ратушняк. – Дис. ... канд. физ.-мат. наук. – Красноярский государственный технический университет. – Красноярск, 2002. – 87 с.

49. Ватолин Д. С. Тенденции развития алгоритмов архивации графики / Д. С. Ватолин // Открытые системы. – 2010. – № 2. – С. 15-24.

50. Сэломон Д. Сжатие данных, изображений и звука / Д. Сэломон. – М.: Техносфера, 2006. – 368 с. – (Серія: Мир программирования: цифровая обработка сигналов).

51. Умняшкин С. В. Оптимизация кодирования цифровых изображений по методу JPEG / С. В. Умняшкин, М. В. Космач // Известия вузов. Электроника. – 2000. – № 4-5. – С. 139-141.

52. Фомин А. А. Основы сжатия информации : Метод. пос. / А. А. Фомин. – СПб: Изд. СПбГТУ, 1998. – 83 с.

53. Хэмминг Р. В. Теория кодирования и теория информации / Р. В. Хэмминг. – М.: Радио и связь, 1985. – 176 с.

54. Цифровая обработка изображений в информационных системах : Учеб. пос. / И. С. Грузман, В. С. Киричук, В. П. Косых и др. – Новосибирск: НГТУ, 2000. – 168 с.

55. Цифровое преобразование изображений : Учеб. пособ. для вузов / Р. Е. Быков, Р. Фрайер, К. В. Иванов, А. А. Манцветов; Под ред. проф. Р. Е. Быкова. – М.: Горячая линия–Телеком, 2003. – 228 с.

56. Шульгин В. И. Основы теории передачи информации. Ч. I. Экономное кодирование : Учеб. пособ. / В. И. Шульгин. – Харьков: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2003. – 102 с.

57. Кожем’яко В.П. Аналіз та перспективи розвитку кодування зображень / В.П. Кожем’яко, В.П. Майданюк, К.М. Жуков - Вісник ВПІ, 1999, № 3. – 42-48с.

58. Кожем'яко В.П., Майданюк В.П., Жуков К.М., Хамді Р.Р., Піка С.О. Фрактальне стиснення зображень природного походження // Міжнародний науково-технічнийжурнал "Вимірювальна та обчислювальна техніка в технологічних процесах", Хмельницький, 1999, № 2, с. 50-54

59. Kozhemiako V.P., Maidanuk V.P., Pika S.,Zhukov K.M. Speeding up of fractal image compression // Proceeding of SPIE, 2001, Vol. 4425, p. 9-16.

60. Майданюк В. П. Оптимізація роботи з масивами при програмній реалізації фрактального ущільнення зображень // Тези доповідей Четвертої Міжнародної науково-практичної конференції «Інформаційні технології та комп'ютерна інженерія» м. Вінниця, 28-30 травня 2014 року. - http://epsi.vntu.edu.ua/uploads/2014/21566f3cb607ef54fad70ae3695246eabd53cd46.pdf

61. Stanislav A. Fractal research methodology // Nature Neurosci. 1999. 1. 10-140.

62. Уэлстид С. Фракталы и вейвлеты для сжатия изображений в действии.Учебное пособ. – М.: Издательство Триумф, 2003 – 320 с.:ил.

63. Guorui Jiang, Yuzhuo Zhong Fast fractal compression based on HV partition. Departament of computer science, Hebei Teacher's University, Shijiazhuang.

64. Fisher Y. Fractal Image Compression: Theory and Application, Springer-Verlag, New York, 1995.

65. «Практическое применение фрактальных алгоритмов» [Електронний ресурс]. Режим доступу: http://m-rush.ru/theory/item/184-fraktaly-na-praktyke.html (дата обращения: 29.05.2016) – Назва з екрану.

66. Всё о сжатии данных, изображений и видео «Методы сжатия данных: Сжатие изображений» [Електронний ресурс]. – Режим доступу: http://www.compression.ru/book/part2/part2__1.htm – Назва з екрану.

67. Фрактал [Електронний ресурс]. Режим доступу: https://ru.wikipedia.org/wiki/Фрактал – Назва з екрану.

68. Jacquin A. Image coding based on a fractal theory of iterated contractive image transformations / A. Jacquin. // IEEE Transactions on Image Processing. – 1992. – No 1. – Pp. 18-30.

69. М.П. Шарабайко, А.Н. Осокин, Статья «Быстродействующий алгоритм фрактального сжатия изображений» [Електронний ресурс]. Режим доступу: http://fic.bos.ru/articles/ – Назва з екрану.

70. Стаття «Фрактальное сжатие изображений» [Електронний ресурс]. Режим доступу: http://www.compression.ru/arctest/descript/fract-comp.htm – Назва з екрану.

71. А. С. Сибиряков «Фрактальное сжатие изображений» [Електронний ресурс]. Режим доступу: http://cs.usu.edu.ru/study/fractal/#11 – Назва з екрану.

72. Программная реализация дискретного косинусного преобразования алгоритма JPEG [Електронний ресурс]: – Режим доступу: http://metod.vt.tpu.ru/edu/df/ti/pri/lab2c/index.html. – Назва з екрану.

73. Всё о сжатии данных, изображений и видео [Електронний ресурс]. Режим доступу: http://www.compression.ru/book/part2/part2__3.htm – Назва з екрану.

74. Статья «Фрактальное сжатие изображений» [Електронний ресурс]. Режим доступу: http://www.compression.ru/arctest/descript/fract-comp.htm – Назва з екрану.

75. Топорков А. Fractalimagefile – новые горизонты сжатия изображений / А. Топорков // Chip. – 2001. – № 7. – С. 121–123.

76. И.В. Бойченко, С.С. Кулбаев, А.А. Немеров, В.В. Голенков «эксперимент по фрактальному сжатию rgb-изображений на вычислительном кластере» - 2012 год.

77. В.В. Гребеник «Усовершенствованный алгоритм сжатия изображения на основе ИФС» МГУ, Москва - 1998 год.

78. С.В. Илюшин «Разработка алгоритмов быстрого фрактального сжатия цифровых изображений», Москва - 2012 год.

79. В.В. Окунев, А.С. Потапов «Оптимизация разбиения изображения в форме квадродерева по критерию минимальной длины описания во фрактальном сжатии». Санкт-Петербург, - 2011год.

80. Яншин В., Калинин Г. Обработка изображений на языке СИ для IBM PC: Алгоритмы и программы. - Москва: Мир, 1994.

81. Нетравали А. Н. Кодирование изображений: Обзор / А. Н. Нетравали Дж. О. Лимб // ТИИЭР. – 1980. – Т. 68. – № 3. – С. 76–124.

82. Зубарев Ю.Б., Дворкович В.П. Цифровая обработка телевизионных и компьютерных изображений. - Москва, 1997.

83. M.F. Barnsley, Fractals Everywhere. London: Academic Press Inc., 1988.

84. A.E. Jacquin, “Fractal image coding based on a theory of iterated contractive image transformations,” in Proceedings of SPIE Visual Communications and Image Processing ’90, vol. 1360, pp. 227-239 1990.

85. G.E. Qien, Z. Baharav, S. Lepsqy, E. Karnin. A new improved collage theorem with appplications to multiresolution fractal image coding. In Proc. ICASSP, 1994.

86. Saupe, D., Hamzaoui, R., Hartenstein, H., Fractal image compression - An introductory overview, in: Fractal Models for Image Synthesis, Compression and Analysis, D. Saupe, J. Hart (eds.), SIGGRAPH'96 Course Notes, ACM, New Orleans, Louisiana, Aug. 1996.

87. Ансон Л. Фрактальное сжатие изображений / Л. Ансон, М. Барнсли// Мир ПК. – 1992. – № 4. – С. 12–20.

88. Егоров П. Фрактальная компрессия изображений / П. Егоров // Программист. – 2001. – № 12. – С. 53–58.

89. Илюшин С. В. Повышение эффективности фрактального сжатия цифровых изображений / С. В. Илюшин, С. Д. Свет // Проблемы создания и развития информационно-телекоммуникационной системы специального назначения: сб. тр. науч. -практич. конф. – Орел : Академия ФАПСИ, 2003. – Ч. 2. – С. 64–65.

90. Киволвиц П. Сжатие изображений по стандарту JPEG / П. Киволвиш // Мир ПК. – 1992. – № 4. – С. 46–51.

91. Кунт М. Блочное кодирование графических материалов: Обзор/ М. Кунт, О. Джонсен // ТИИЭР. – 1980. – Т. 68. – № 7. – С. 21–40.

92. Морозов А. Д. Введение в теорию фракталов / А. Д. Морозов. – М. : Институт компью-терных исследований, 2002. – 160 с.

93. Зубко Р. А. Фрактали. Східно-Європейський журнал передових технологій. 2012. Т. 6, № 4(60). С. 13-14.

94. Зубко Р. А. Алгоритми стиснення зображень. Східно-Європейський журнал передових технологій. 2013. Т. 1, № 2(61). С. 40-44.

95. Зубко Р. А. Стиснення зображень фрактальним методом. Східно-Європейський журнал передових технологій. 2014. Т. 6, № 2(72). С. 23-28.

96. Зубко Р. А. Методи оптимізації фрактального стиснення зображень. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2015. №1 (17). С. 167-176.

97. Зубко Р. А. Покращення ефективності стиснення зображень фрактальним методом. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2017. №1 (18). С. 190-196.

98. Зубко Р. А. Варіанти підвищення швидкодії стиснення зображень фрактальним методом. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2018. №2 (21/2). С. 128-135.

99. Зубко Р. А. Дослідження можливості покращення часової ефективності фрактального стиснення зображень. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2019. № 1 (22). С. 256-266.

100. Roman Zubko. Improving the efficiency of fractal image compression by the criterion of detail of rank and domain blocks. «Danish Scientific Journal» (DSJ) Kobenhavn. Denmark. 2021. Vol. № 45 (2). Pp. 38-43. ISSN 3375-2389.

101. Зубко Р. А. Фрактальний метод стиснення зображень Матеріали VII Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 1-3 листопада 2012. С. 95-97.

Додаток 1. Список публікацій здобувача

1. Зубко Р. А. Фрактали. Східно-Європейський журнал передових технологій. 2012. Т. 6, № 4(60). С. 13-14.

2. Зубко Р. А. Алгоритми стиснення зображень. Східно-Європейський журнал передових технологій. 2013. Т. 1, № 2(61). С. 40-44.

3. Зубко Р. А. Стиснення зображень фрактальним методом. Східно-Європейський журнал передових технологій. 2014. Т. 6, № 2(72). С. 23-28.

4. Зубко Р. А. Методи оптимізації фрактального стиснення зображень. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2015. №1 (17). С. 167-176.

5. Зубко Р. А. Покращення ефективності стиснення зображень фрактальним методом. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2017. №1 (18). С. 190-196.

6. Зубко Р. А. Варіанти підвищення швидкодії стиснення зображень фрактальним методом. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2018. №2 (21/2). С. 128-135.

7. Зубко Р. А. Дослідження можливості покращення часової ефективності фрактального стиснення зображень. Вісник Університету «Україна», серія: інформатика, обчислювальна техніка та кібернетика, Київ, 2019. № 1 (22). С. 256-266.

8. Roman Zubko. Improving the efficiency of fractal image compression by the criterion of detail of rank and domain blocks. «Danish Scientific Journal» (DSJ) Kobenhavn. Denmark. 2021. Vol. № 45 (2). Pp. 38-43. ISSN 3375-2389.

9. Зубко Р. А. Фрактальний метод стиснення зображень Матеріали VII Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 1-3 листопада 2012. С. 95-97.

10. Зубко Р. А. Алгоритм швидкого фрактального стиснення цифрових зображень. Матеріали VIII Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 7-9 листопада 2013. С. 72-73.

11. Зубко Р. А. Стиснення зображень фрактальним методом. Матеріали XII Всеукраїнської наукової конференції студентів і молодих вчених «Молодь: освіта, наука, духовність», Україна, Київ, 21-22 квітня 2015. С. 250-252.

12. Зубко Р. А. Методи оптимізації фрактального стиснення зображень. Матеріали IX Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 27-28 листопада 2015. С. 41-42.

13. Зубко Р. А. Покращення часової ефективності базового методу фрактального стиснення зображень. Матеріали XIII Всеукраїнської наукової конференції студентів і молодих вчених «Молодь: освіта, наука, духовність», Україна, Київ, 12-14 квітня 2016. С. 224-225.

14. Зубко Р. А. Табличний метод підвищення швидкодії фрактального стиснення зображень. Матеріали X Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 30-31 березня 2017. С. 86-88.

15. Зубко Р. А. Покращення ефективності стиснення зображень фрактальним методом з використанням табличного пошуку. Матеріали XI Всеукраїнської науково-практичної конференції «Комп'ютерні технології: наука і освіта», Україна, Київ, 23-24 березня 2018. С. 23-27.

16. Зубко Р. А. Дослідження можливості покращення часової ефективності фрактального стиснення зображень. Матеріали XVII Всеукраїнської наукової конференції студентів і молодих вчених «Молодь: освіта, наука, духовність», Україна, Київ, 27-28 травня 2020. Ч. 3. С. 401-403.

Додаток 2. Лістинг програми

Код програми підвищення швидкодії фрактального стиснення

```
unit Main;

interface

uses

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, Optimization, StdCtrls, Math;

type
  TMainForm = class(TForm)
    Image: TImage;
    rgMode: TRadioGroup;
    btCompress: TButton;
    meResults: TMemo;
    gbYImage: TGroupBox;
    laRegionSize: TLabel;
    laDomainOffset: TLabel;
    edRegionSize: TEdit;
    edDomainOffset: TEdit;
    gbYVImage: TGroupBox;
    laRegionSizeUV: TLabel;
    laDomainOffsetUV: TLabel;
    edRegionSizeUV: TEdit;
    edDomainOffsetUV: TEdit;
    procedure btCompressClick(Sender: TObject);
    procedure rgModeClick(Sender: TObject);
  private
    { Private declarations }
    procedure CompressImage;
    procedure DecompressImage;
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;
```

```
implementation

{$R *.dfm}

function Round5(X: Extended): Extended;
begin
  Result := Round(X * 100000) / 100000;
end;

function GetFileSize(const aFilename: String): Int64;
var
  Info: TWin32FileAttributeData;
begin
  if GetFileAttributesEx(PChar(aFileName), GetFileExInfoStandard, @Info) then
    Result := Int64(Info.nFileSizeLow) or Int64(Info.nFileSizeHigh shl 32)
  else
    Result := 0;
end;

procedure TMainForm.CompressImage;
var
  FC: TOptimizedIFS;
  RegionSize, DomainOffset, RegionSizeUV, DomainOffsetUV: Integer;
begin
  RegionSize := StrToIntDef(edRegionSize.Text, 0);
  DomainOffset := StrToIntDef(edDomainOffset.Text, 0);

  if (RegionSize <> EnsureRange(RegionSize, 4, 16)) then
  begin
    ShowMessage('Invalid region size');
    Exit;
  end;

  if (DomainOffset <> EnsureRange(DomainOffset, 1, 20)) then
  begin
    ShowMessage('Invalid domain offset');
    Exit;
  end;

  if RGBEnabled then
  begin
```

```
  RegionSizeUV := StrToIntDef(edRegionSizeUV.Text, 0);
  DomainOffsetUV := StrToIntDef(edDomainOffsetUV.Text, 0);

  if (RegionSizeUV <> EnsureRange(RegionSizeUV, 2, 16)) then
  begin
    ShowMessage('Invalid region size');
    Exit;
  end;

  if (DomainOffsetUV <> EnsureRange(DomainOffsetUV, 1, 20)) then
  begin
    ShowMessage('Invalid domain offset');
    Exit;
  end;
 end;

 FC := TOptimizedIFS.Create;
 try
  FC.LoadFromBMPFile('lenna512.bmp');
  FC.SaveToJPEGFile('lenna512.jpg', 20);
  FC.Configure(RegionSize, DomainOffset, RegionSizeUV, DomainOffsetUV);

  FC.SaveToIFSFile('lenna512.ifs');

  {FC.DumpMatrix(FC.ImageY.Pixels, 'upakovka.1.ishodnoe-izobrazhenie-
Y.bmp');
  if RGBEnabled then
  begin
    FC.DumpMatrix(FC.ImageU.Pixels, 'upakovka.2.ishodnoe-izobrazhenie-
U.bmp');
    FC.DumpMatrix(FC.ImageV.Pixels, 'upakovka.3.ishodnoe-izobrazhenie-
V.bmp');
  end;}
 finally
  FC.Free;
 end;
end;

procedure TMainForm.DecompressImage;
var
 FC: TOptimizedIFS;
```

```
begin
  FC := TOptimizedIFS.Create;
  try
    FC.LoadFromIFSFile('lenna512.ifs');
    Image.Picture.Assign(nil);
    FC.SaveToBitmap(Image.Picture.Bitmap);

    {FC.DumpMatrix(FC.ImageY.Pixels, 'rozpakovka.1.ishodnoe-izobrazhenie-
Y.bmp');
    if RGBEnabled then
    begin
      FC.DumpMatrix(FC.ImageU.Pixels, 'rozpakovka.2.ishodnoe-izobrazhenie-
U.bmp');
      FC.DumpMatrix(FC.ImageV.Pixels, 'rozpakovka.3.ishodnoe-izobrazhenie-
V.bmp');
    end;}
  finally
    FC.Free;
  end;
end;

procedure TMainForm.btCompressClick(Sender: TObject);
var
  TickCount, CompressionTime, DecompressionTime, CompressedSize,
DecompressedSize: Integer;
begin
  meResults.Lines.Clear;
  Application.ProcessMessages;

  RGBEnabled := rgMode.ItemIndex > 0;

  TickCount := GetTickCount;
  CompressImage;
  CompressionTime := GetTickCount - TickCount;

  TickCount := GetTickCount;
  DecompressImage;
  DecompressionTime := GetTickCount - TickCount;

  CompressedSize := GetFileSize('lenna512.ifs');
  DecompressedSize := 512 * 512 * 3;
```

```
  meResults.Lines.Add('Compression time:');
  meResults.Lines.Add(FloatToStr(CompressionTime / 1000) + ' sec');
  meResults.Lines.Add('');

  meResults.Lines.Add('Decompression time:');
  meResults.Lines.Add(FloatToStr(DecompressionTime / 1000) + ' sec');
  meResults.Lines.Add('');

  meResults.Lines.Add('Compressed size:');
  meResults.Lines.Add(FloatToStr(Round5(CompressedSize / 1024)) + ' KB');
  meResults.Lines.Add('');

  meResults.Lines.Add('Decompressed size (RGB24):');
  meResults.Lines.Add(FloatToStr(Round5(DecompressedSize / 1024)) + ' KB');
  meResults.Lines.Add('');

  meResults.Lines.Add('Compression value:');
  meResults.Lines.Add(FloatToStr(Round5(DecompressedSize / CompressedSize))
+ 'x');
  meResults.Lines.Add('');
end;

procedure TMainForm.rgModeClick(Sender: TObject);
begin
  edRegionSizeUV.Enabled := (rgMode.ItemIndex > 0);
  edDomainOffsetUV.Enabled := (rgMode.ItemIndex > 0);
end;

end.

************************
unit Optimization;

interface

uses
  Windows, Messages, SysUtils, Graphics, Classes, Math, jpeg;

type
  TMatrix = array of array of Integer;
```

```
  //     TMemIfsRec = packed record
    FormNum: Byte; //
    Betta: ShortInt; //
    DomCoordX, DomCoordY: Word; //
  end;

  TRegionRec = packed record
    MeanColor: Integer; //
    Ifs: TMemIfsRec; //
  end;

  PDomainEntry = ^TDomainEntry;
  TDomainEntry = record
    X, Y: Integer;
    Next: PDomainEntry;
  end;

  TTransformType = (ttRot0, ttRot90, ttRot180, ttRot270, ttSimmX, ttSimmY,
ttSimmDiag1, ttSimmDiag2);

  TOptimizedFractal = class
  public
    DomainImage: TMatrix;   //
    DomainWidth: Integer;
    DomainHeight: Integer;
    Domains: TMatrix; //
    SourContours: TMatrix;
    DomainContours: TMatrix;
    RegionsWeights: TMatrix;
    DomainsWeights: TMatrix;
    DomainTable: array[0..255] of PDomainEntry;
    Pixels: TMatrix;
    Width: Integer;
    Height: Integer;
    RegionSize: Integer;
    DomainOffset: Integer;
    Regions: array of array of TRegionRec;

    procedure SetSize(W, H: Integer);
    procedure CreateDomainImage;
```

```
    function GetMeanBrigth(Img: TMatrix; X, Y: Integer): Byte;
    function GetDifference(Region: TMatrix; DomCoordX, DomCoordY, Betta:
Integer): Integer;

    procedure Compress;
    procedure Decompress;
  end;

type
  TOptimizedIFS = class
    ImageY: TOptimizedFractal;
    ImageU: TOptimizedFractal;
    ImageV: TOptimizedFractal;
    constructor Create;
    destructor Destroy; override;
    procedure Configure(RegionSizeY, DomainOffsetY, RegionSizeUV,
DomainOffsetUV: Integer);
    procedure LoadFromBitmap(Bitmap: TBitmap);
    procedure LoadFromBMPFile(FileName: string);
    procedure LoadFromJPEGFile(FileName: string);
    procedure LoadFromIFSFile(FileName: string);
    procedure SaveToBitmap(Bitmap: TBitmap);
    procedure SaveToBMPFile(FileName: string);
    procedure SaveToJPEGFile(FileName: string; Quality: Integer);
    procedure SaveToIFSFile(FileName: string);
    procedure DumpMatrix(Matrix: TMatrix; FileName: string);
  end;

var
  RGBEnabled: Boolean;

implementation

function KeepByte(Value: Integer): Byte;
begin
  if Value > 255 then
    Result := 255
  else if Value < 0 then
    Result := 0
  else
```

```
    Result := Value;
end;

procedure ApplyLaplacian(Sour, Dest: TMatrix; W, H: Integer);
var
  X, Y, k1, k2, k3, k4, k5, k6, k7, k8, k9, n, Sum: Integer;
begin
  k1 := -1;  k2 := -1;  k3 := -1;
  k4 := -1;  k5 := 8;  k6 := -1;
  k7 := -1;  k8 := -1;  k9 := -1;

  n := k1+k2+k3+k4+k5+k6+k7+k8+k9;
  if n = 0 then
    n := 1;

  for X := 1 to W - 2 do
  for Y := 1 to H - 2 do
  begin
    Dest[Y, X] := KeepByte((
      k1*Sour[Y-1, X-1] + k2*Sour[Y-1, X] + k3*Sour[Y-1, X+1] +
      k4*Sour[Y, X-1] + k5*Sour[Y, X] + k6*Sour[Y, X+1] +
      k7*Sour[Y+1, X-1] + k8*Sour[Y+1, X] + k9*Sour[Y+1, X+1]
      ) div n);
  end;
end;

procedure Transform(Sour, Dest: TMatrix; TransformType: TTransformType;
RegionSize: Integer);
var
  I, J: Integer;
begin
  case TransformType of
    ttRot0: // Поворот на 0 градусів
      begin
        for I := 0 to RegionSize - 1 do
          for J := 0 to RegionSize - 1 do
            Dest[J, I] := Sour[I, J];
      end;

    ttRot90: // Поворот на 90 градусів
      begin
```

```
    for I := 0 to RegionSize - 1 do
      for J := 0 to RegionSize - 1 do
        Dest[(RegionSize - 1 - J), I] := Sour[I, J];
  end;

ttRot180: // Поворот на 180 градусів
  begin
    for I := 0 to RegionSize - 1 do
      for J := 0 to RegionSize - 1 do
        Dest[(RegionSize - 1 - I), (RegionSize - 1 - J)] := Sour[I, J];
  end;

ttRot270: // Поворот на 270 градусів
  begin
    for I := 0 to RegionSize - 1 do
      for J := 0 to RegionSize - 1 do
        Dest[J, (RegionSize - 1 - I)] := Sour[I, J];
  end;

ttSimmX: // Симетрія відносно X
  begin
    for I := 0 to RegionSize - 1 do
      for J := 0 to RegionSize - 1 do
        Dest[(RegionSize - 1 - I), + J] := Sour[I, J];
  end;

ttSimmY: // Симетрия відносно У
  begin
    for I := 0 to RegionSize - 1 do
      for J := 0 to RegionSize - 1 do
        Dest[I, (RegionSize - 1 - J)] := Sour[I, J];
  end;

ttSimmDiag1: // Симетрия від головної діагоналі
  begin
    for I := 0 to RegionSize - 1 do
      for J := 0 to RegionSize - 1 do
        Dest[J, I] := Sour[I, J];
  end;

ttSimmDiag2: // Симетрия від другорядної головної діагоналі
```

```
      begin
        for I := 0 to RegionSize - 1 do
          for J := 0 to RegionSize - 1 do
            Dest[(RegionSize - 1 - J), (RegionSize - 1 - I)] := Sour[I, J];
      end;
  end;
end;

procedure CopyRegion(Dest, Sour: TMatrix; X, Y: Integer; DstW, DstH, SrcW,
SrcH: Integer);
var
  I, J, X0, Y0: Integer;
begin
  for J := 0 to DstH - 1 do
  for I := 0 to DstW - 1 do
  begin
    X0 := X + I;
    Y0 := Y + J;
    if (X0 < SrcW) and (Y0 < SrcH) then
      Dest[J, I] := Sour[Y0, X0];
  end;
end;

{ TOptimizedFractal }

function TOptimizedFractal.GetDifference(Region: TMatrix; DomCoordX,
  DomCoordY, Betta: Integer): Integer;
var
  X1, Y1, Diff: Integer;
begin
  Result := 0;
  for Y1 := 0 to RegionSize - 1 do
  for X1 := 0 to RegionSize - 1 do
  begin
    Diff := Region[Y1, X1] - DomainImage[DomCoordY + Y1, DomCoordX +
X1];
    Inc(Result, Abs(Diff - Betta));
  end;
end;

function TOptimizedFractal.GetMeanBrigth(Img: TMatrix; X, Y: Integer): Byte;
```

```
var
  I, J, Bufer: Integer;
begin
  Bufer := 0;
  for I := Y to Y + RegionSize - 1 do
  for J := X to X + RegionSize - 1 do
    Inc(Bufer, Img[I, J]);
  Result := Trunc(Bufer / (RegionSize * RegionSize));
end;

procedure TOptimizedFractal.Compress;
var
  RegX, RegY, DomX, DomY, Error, BestError, Betta, TransNum: Integer;
  Region, BaseRegion: TMatrix;
  DCoordX, DCoordY, BestFormNum, BestDomX, BestDomY, BestBetta: Integer;
  W, X, Y: Integer;
  I, J, XRegions, YRegions, XDomains, YDomains: Integer;
  Sum: Integer;
  Entry, Iter: PDomainEntry;
begin
  CreateDomainImage;

  XDomains := ((Width div 2) - RegionSize) div DomainOffset;
  YDomains := ((Height div 2) - RegionSize) div DomainOffset;
  XRegions := Width div RegionSize;
  YRegions := Height div RegionSize;

  SetLength(Regions, XRegions, YRegions);
  for Y := 0 to YRegions - 1 do
  for X := 0 to XRegions - 1 do
    Regions[X, Y].MeanColor := GetMeanBrigth(Pixels, X * RegionSize, Y *
RegionSize);

  SetLength(Domains, XDomains, YDomains);
  for Y := 0 to YDomains - 1 do
  for X := 0 to XDomains - 1 do
    Domains[X, Y] := GetMeanBrigth(DomainImage, X * DomainOffset, Y *
DomainOffset);

  SetLength(SourContours, Width, Height);
  ApplyLaplacian(Pixels, SourContours, Width, Height);
```

```
  SetLength(DomainContours, DomainWidth, DomainHeight);
  ApplyLaplacian(DomainImage, DomainContours, DomainWidth,
DomainHeight);

  SetLength(RegionsWeights, XRegions, YRegions);

  for Y := 0 to YRegions - 1 do
  for X := 0 to XRegions - 1 do
  begin
    Sum := 0;
    for I := 0 to RegionSize - 1 do
    for J := 0 to RegionSize - 1 do
      Sum := Sum + SourContours[Y * RegionSize + J, X * RegionSize + I];

    RegionsWeights[Y, X] := Sum div (RegionSize * RegionSize);
  end;

  SetLength(DomainsWeights, XDomains, YDomains);

  for Y := 0 to YDomains - 1 do
  for X := 0 to XDomains - 1 do
  begin
    Sum := 0;
    for I := 0 to RegionSize - 1 do
    for J := 0 to RegionSize - 1 do
      Sum := Sum + DomainContours[Y * DomainOffset + J, X * DomainOffset +
I];

    DomainsWeights[Y, X] := Sum div (RegionSize * RegionSize);
  end;

  for Y := 0 to YDomains - 1 do
  for X := 0 to XDomains - 1 do
  begin
    New(Entry);
    Entry.X := X;
    Entry.Y := Y;
    Entry.Next := nil;

    // ves tekuschego domena
```

```
    W := DomainsWeights[Y, X];

    // dobavliaem element v tablicu vesov
    if DomainTable[W] = nil then
      DomainTable[W] := Entry
    else
    begin
      Iter := DomainTable[W];
      while Iter.Next <> nil do
        Iter := Iter.Next;
      Iter.Next := Entry;
    end;
  end;

  // (RegionSize x RegionSize)
  SetLength(Region, RegionSize, RegionSize);
  SetLength(BaseRegion, RegionSize, RegionSize);

  // Перебираємо рангові блоки
  for RegY := 0 to YRegions - 1 do
  for RegX := 0 to XRegions - 1 do
  begin
    //     //
    BestError := $7FFFFFFF;
    BestFormNum := -1;
    BestDomX := -1;
    BestDomY := -1;
    BestBetta := 0;

    //
    CopyRegion(BaseRegion, Pixels, RegX * RegionSize, RegY * RegionSize,
RegionSize, RegionSize, Width, Height);

    W := RegionsWeights[RegY, RegX];
    Iter := DomainTable[W];
    while Iter <> nil do
    begin
      DomX := Iter.X;
      DomY := Iter.Y;

      //
```

```
  Betta := Regions[RegX, RegY].MeanColor - Domains[DomX, DomY];


  DCoordX := DomX * DomainOffset;
  DCoordY := DomY * DomainOffset;

  //
  for TransNum := 0 to 7 do
  begin

    Transform(BaseRegion, Region, TTransformType(TransNum), RegionSize);


    Error := GetDifference(Region, DCoordX, DCoordY, Betta);

    //
    if Error < BestError then
    begin
      BestError := Error;
      BestFormNum := TransNum;
      BestDomX := DCoordX;
      BestDomY := DCoordY;
      BestBetta := Betta;
    end;
  end;
  Iter := Iter.Next;
end;

//
with Regions[RegX, RegY].Ifs do
begin
  DomCoordX := BestDomX;
  DomCoordY := BestDomY;


  BestBetta := BestBetta div 2;

  Betta := BestBetta;
  if BestFormNum = 1 then BestFormNum := 3 else // 90 -> 270
    if BestFormNum = 3 then BestFormNum := 1;  // 270 -> 90
  FormNum := BestFormNum;
```

```
    end; //
  end;
end;

procedure TOptimizedFractal.Decompress;
var
  I, J, X, Y, Pixel, Iter, XRegions, YRegions: Integer;
  Domain1, Domain2: TMatrix;
begin
    SetLength(Pixels, Width, Height);

  SetLength(Domain1, RegionSize, RegionSize);
  SetLength(Domain2, RegionSize, RegionSize);

  XRegions := Width div RegionSize;
  YRegions := Height div RegionSize;

  for Iter := 1 to 15 do
  begin
    //
    CreateDomainImage;

     for J := 0 to YRegions - 1 do
    for I := 0 to XRegions - 1 do
    begin
          CopyRegion(Domain1, DomainImage, Trunc(Regions[I,
J].Ifs.DomCoordX), Trunc(Regions[I, J].Ifs.DomCoordY), RegionSize,
RegionSize, DomainWidth, DomainHeight);

      //
      Transform(Domain1, Domain2, TTransformType(Regions[I, J].Ifs.FormNum),
RegionSize);

      for Y := 0 to RegionSize - 1 do
      for X := 0 to RegionSize - 1 do
      begin
        Pixel := KeepByte(Domain2[Y, X] + Regions[I, J].Ifs.Betta * 2);
        Pixels[J * RegionSize + Y, I * RegionSize + X] := Pixel;
      end;
    end;
  end;
```

```
end;

procedure TOptimizedFractal.CreateDomainImage;
var
  X, Y, PixColor: Integer;
begin
  DomainWidth := Width div 2;
  DomainHeight := Height div 2;
  SetLength(DomainImage, DomainWidth, DomainHeight);

  for Y := 0 to DomainHeight - 1 do
  for X := 0 to DomainHeight - 1 do
  begin
       PixColor :=
     Pixels[Y * 2, X * 2] + Pixels[Y * 2, X * 2 + 1] +          // 1 2
     Pixels[Y * 2 + 1, X * 2] + Pixels[Y * 2 + 1, X * 2 + 1];    // 3 4

    DomainImage[Y, X] := Trunc(PixColor / 4 * 0.75);
  end;
end;

procedure TOptimizedFractal.SetSize(W, H: Integer);
begin
  Width := W;
  Height := H;
  SetLength(Pixels, W, H);
end;

type
    TOptimizedIFSHeader = packed record
    RegSize: Byte; // Розмір рангового блоку
    XRegions, YRegions: Word; // Кіл-ть рангових блоків по X и У
  end;


  TFileIfsRec = packed record
    FormNum: Byte; // Номер перетворення
    Betta: ShortInt; //

    //     // Координати лівого верхнього кута домена
    // 0й байт – молодший байт от X
```

```
    // 1й байт - молодший байт от У
    DomCoordX: Byte;
    DomCoordY: Byte;
  end;

  TRGB32 = packed record
    B, G, R, A: Byte;
  end;
  PRGB32 = ^TRGB32;

procedure RGB2Y(Bitmap: TBitmap; ImageY: TOptimizedFractal);
var
  W, H, X, Y: Integer;
  P: PRGB32;
begin
  W := Bitmap.Width;
  H := Bitmap.Height;

  ImageY.SetSize(W, H);
  for Y := 0 to H - 1 do
  begin
    P := PRGB32(Bitmap.ScanLine[Y]);
    for X := 0 to W - 1 do
    begin
      ImageY.Pixels[Y, X] := (P.R + P.G + P.B) div 3;
      P := PRGB32(Integer(P) + 3);
    end;
  end;
end;

procedure Y2RGB(ImageY: TOptimizedFractal; Bitmap: TBitmap);
var
  W, H, X, Y, V: Integer;
  P: PRGB32;
begin
  W := ImageY.Width;
  H := ImageY.Height;
  Bitmap.Width := W;
  Bitmap.Height := H;
  Bitmap.PixelFormat := pf24bit;
  for Y := 0 to H - 1 do
```

```
  begin
    P := PRGB32(Bitmap.ScanLine[Y]);
    for X := 0 to W - 1 do
    begin
      V := ImageY.Pixels[Y, X];
      P.R := V;
      P.G := V;
      P.B := V;
      P := PRGB32(Integer(P) + 3);
    end;
  end;
end;

//=============================================================
=============================================================
=
//       RGB to YUV Converter
//
//       Y  = (0.257 * R) + (0.504 * G) + (0.098 * B) + 16
//       U = -(0.148 * R) - (0.291 * G) + (0.439 * B) + 128
//       V =  (0.439 * R) - (0.368 * G) - (0.071 * B) + 128
//=============================================================
=============================================================
=
procedure RGB2YUV(Bitmap: TBitmap; ImageY, ImageU, ImageV:
TOptimizedFractal);
var
  W, H, W2, H2, I, X, Y: Integer;
  YR, YG, YB, UR, UG, UB, VR, VG, VB: array[0..255] of Integer;
  P: PRGB32;
begin
  W := Bitmap.Width;
  H := Bitmap.Height;
  W2 := W div 2;
  H2 := H div 2;

  ImageY.SetSize(W, H);
  ImageU.SetSize(W2, H2);
  ImageV.SetSize(W2, H2);

  for I := 0 to 255 do
```

```
  begin
    YR[I] := Trunc(0.257 * I * 1024.0);
    YG[I] := Trunc(0.504 * I * 1024.0);
    YB[I] := Trunc(0.098 * I * 1024.0);

    UR[I] := Trunc(-0.148 * I * 1024.0);
                UG[I] := Trunc(-0.291 * I * 1024.0);
                UB[I] := Trunc(0.439 * I * 1024.0);

    VR[I] := Trunc(0.439 * I * 1024.0);
                VG[I] := Trunc(-0.368 * I * 1024.0);
                VB[I] := Trunc(-0.071 * I * 1024.0);
  end;

  for Y := 0 to H - 1 do
  begin
    P := PRGB32(Bitmap.ScanLine[Y]);
    for X := 0 to W - 1 do
    begin
      ImageY.Pixels[Y, X] := KeepByte(((YR[P.R] + YG[P.G] + YB[P.B]) div
1024) + 16);
      if ((X and 1 = 0) and (Y and 1 = 0)) then
        ImageU.Pixels[Y div 2, X div 2] := KeepByte(((UR[P.R] + UG[P.G] +
UB[P.B]) div 1024) + 128);
      if ((X and 1 <> 0) and (Y and 1 <> 0)) then
        ImageV.Pixels[Y div 2, X div 2] := KeepByte(((VR[P.R] + VG[P.G] +
VB[P.B]) div 1024) + 128);
      P := PRGB32(Integer(P) + 3);
    end;
  end;
end;

//=========================================================
=========================================================
=//     YUV to RGB Converter
//
//       R = 1.164*(Y - 16) + 1.596*(V - 128)
//       G = 1.164*(Y - 16) - 0.813*(V - 128) - 0.391*(U - 128)
//       B = 1.164*(Y - 16) + 2.018*(U - 128)
```

```
//=============================================================
=============================================================
=
procedure YUV2RGB(ImageY, ImageU, ImageV: TOptimizedFractal; Bitmap:
TBitmap);
var
  I, X, Y, X2, Y2, W, H, YC, VC, UC: Integer;
  F: Double;
  YY, UG, UB, VR, VG: array[0..255] of Integer;
  P: PRGB32;
begin
  for I := 0 to 255 do
  begin
    F := (I - 128) * 1024.0;
    YY[I] := Trunc(1.164 * (I - 16) * 1024.0);
    UG[I] := Trunc(0.391 * F);
    UB[I] := Trunc(2.018 * F);
    VR[I] := Trunc(1.596 * F);
    VG[I] := Trunc(0.813 * F);
  end;

  W := ImageY.Width;
  H := ImageY.Height;
  Bitmap.Width := W;
  Bitmap.Height := H;
  Bitmap.PixelFormat := pf24bit;
  Y2 := 0;
  for Y := 0 to H - 1 do
  begin
    X2 := 0;
    P := PRGB32(Bitmap.ScanLine[Y]);
    for X := 0 to W - 1 do
    begin
      YC := YY[ImageY.Pixels[Y, X]];
      UC := ImageU.Pixels[Y2, X2];
      VC := ImageV.Pixels[Y2, X2];

      P.R := KeepByte((YC + VR[VC]) div 1024);
      P.G := KeepByte((YC - VG[VC] - UG[UC]) div 1024);
      P.B := KeepByte((YC + UB[UC]) div 1024);
```

```
      P := PRGB32(Integer(P) + 3);

      if X and 1 <> 0 then
        Inc(X2);
    end;
    if Y and 1 <> 0 then
      Inc(Y2);
  end;
end;

constructor TOptimizedIFS.Create;
begin
  ImageY := TOptimizedFractal.Create;
  ImageU := TOptimizedFractal.Create;
  ImageV := TOptimizedFractal.Create;
end;

destructor TOptimizedIFS.Destroy;
begin
  ImageY.Free;
  ImageU.Free;
  ImageV.Free;
  inherited Destroy;
end;

procedure TOptimizedIFS.Configure(RegionSizeY, DomainOffsetY,
RegionSizeUV, DomainOffsetUV: Integer);
begin
  ImageY.RegionSize := RegionSizeY;
  ImageU.RegionSize := RegionSizeUV;
  ImageV.RegionSize := RegionSizeUV;

  ImageY.DomainOffset := DomainOffsetY;
  ImageU.DomainOffset := DomainOffsetUV;
  ImageV.DomainOffset := DomainOffsetUV;
end;

procedure TOptimizedIFS.LoadFromBitmap(Bitmap: TBitmap);
begin
  Bitmap.PixelFormat := pf24bit;
  if RGBEnabled then
```

```
    RGB2YUV(Bitmap, ImageY, ImageU, ImageV)
  else
    RGB2Y(Bitmap, ImageY);
end;

procedure TOptimizedIFS.LoadFromBMPFile(FileName: string);
var
  Bitmap: TBitmap;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile(FileName);
    LoadFromBitmap(Bitmap);
  finally
    Bitmap.Free;
  end;
end;

procedure TOptimizedIFS.LoadFromJPEGFile(FileName: string);
var
  JPEG: TJPEGImage;
  Bitmap: TBitmap;
begin
  JPEG := TJPEGImage.Create;
  Bitmap := TBitmap.Create;
  try
    JPEG.LoadFromFile(FileName);
    Bitmap.Assign(JPEG);
    LoadFromBitmap(Bitmap);
  finally
    JPEG.Free;
    Bitmap.Free;
  end;
end;

procedure TOptimizedIFS.LoadFromIFSFile(FileName: string);
var
  X, Y, XRegions, YRegions, RegionSize: Integer;
  Header: TOptimizedIFSHeader;
  OldIfs: TMemIfsRec;
  FileIfs: TFileIfsRec;
```

```
  Stream: TMemoryStream;
begin
  Stream := TMemoryStream.Create;
  try
    Stream.LoadFromFile(FileName);
    Stream.Read(Header, SizeOf(TOptimizedIFSHeader));

    XRegions := Header.XRegions;
    YRegions := Header.YRegions;
    RegionSize := Header.RegSize;

    ImageY.SetSize(XRegions * RegionSize, YRegions * RegionSize);
    ImageY.RegionSize := RegionSize;
    SetLength(ImageY.Regions, XRegions, YRegions);
    for Y := 0 to YRegions - 1 do
    for X := 0 to XRegions - 1 do
    begin
      Stream.Read(FileIfs, SizeOf(FileIfs));
      OldIfs.FormNum := FileIfs.FormNum;
      OldIfs.Betta := FileIfs.Betta;
      OldIfs.DomCoordX := FileIfs.DomCoordX;
      OldIfs.DomCoordY := FileIfs.DomCoordY;
      ImageY.Regions[X, Y].Ifs := OldIfs;
    end;

    if RGBEnabled then
    begin
      Stream.Read(Header, SizeOf(TOptimizedIFSHeader));
      XRegions := Header.XRegions;
      YRegions := Header.YRegions;
      RegionSize := Header.RegSize;

      ImageU.SetSize(XRegions * RegionSize, YRegions * RegionSize);
      ImageU.RegionSize := RegionSize;
      SetLength(ImageU.Regions, XRegions, YRegions);
      for Y := 0 to YRegions - 1 do
      for X := 0 to XRegions - 1 do
      begin
        Stream.Read(FileIfs, SizeOf(FileIfs));
        OldIfs.FormNum := FileIfs.FormNum;
        OldIfs.Betta := FileIfs.Betta;
```

```
        OldIfs.DomCoordX := FileIfs.DomCoordX;
        OldIfs.DomCoordY := FileIfs.DomCoordY;
        ImageU.Regions[X, Y].Ifs := OldIfs;
      end;

      ImageV.SetSize(XRegions * RegionSize, YRegions * RegionSize);
      ImageV.RegionSize := RegionSize;
      SetLength(ImageV.Regions, XRegions, YRegions);
      for Y := 0 to YRegions - 1 do
      for X := 0 to XRegions - 1 do
      begin
        Stream.Read(FileIfs, SizeOf(FileIfs));
        OldIfs.FormNum := FileIfs.FormNum;
        OldIfs.Betta := FileIfs.Betta;
        OldIfs.DomCoordX := FileIfs.DomCoordX;
        OldIfs.DomCoordY := FileIfs.DomCoordY;
        ImageV.Regions[X, Y].Ifs := OldIfs;
      end;
    end;

  finally
    Stream.Free;
  end;

  ImageY.Decompress;
  if RGBEnabled then
  begin
    ImageU.Decompress;
    ImageV.Decompress;
  end;
end;

procedure TOptimizedIFS.SaveToBitmap(Bitmap: TBitmap);
begin
  if RGBEnabled then
    YUV2RGB(ImageY, ImageU, ImageV, Bitmap)
  else
    Y2RGB(ImageY, Bitmap);
end;

procedure TOptimizedIFS.SaveToBMPFile(FileName: string);
```

```
var
  Bitmap: TBitmap;
begin
  Bitmap := TBitmap.Create;
  try
    SaveToBitmap(Bitmap);
    Bitmap.SaveToFile(FileName);
  finally
    Bitmap.Free;
  end;
end;

procedure TOptimizedIFS.SaveToJPEGFile(FileName: string; Quality: Integer);
var
  Bitmap: TBitmap;
  JPEG: TJPEGImage;
begin
  Bitmap := TBitmap.Create;
  JPEG := TJPEGImage.Create;
  try
    SaveToBitmap(Bitmap);
    JPEG.Assign(Bitmap);
    JPEG.CompressionQuality := Quality;
    JPEG.SaveToFile(FileName);
  finally
    Bitmap.Free;
    JPEG.Free;
  end;
end;

procedure TOptimizedIFS.SaveToIFSFile(FileName: string);
var
  X, Y, XRegions, YRegions, RegionSize: Integer;
  Header: TOptimizedIFSHeader;
  FileIfs: TFileIfsRec;
  AByte: Byte;
  Stream: TMemoryStream;
begin
  ImageY.CreateDomainImage;
  ImageY.Compress;
```

```
if RGBEnabled then
begin
  ImageU.CreateDomainImage;
  ImageU.Compress;

  ImageV.CreateDomainImage;
  ImageV.Compress;
end;

Stream := TMemoryStream.Create();
try
  XRegions := ImageY.Width div ImageY.RegionSize;
  YRegions := ImageY.Height div ImageY.RegionSize;

  //     // Зберігаємо заголовну інформацію
  Header.RegSize := RegionSize;
  Header.XRegions := XRegions;
  Header.YRegions := YRegions;
  Stream.Write(Header, SizeOf(TOptimizedIFSHeader));

  for Y := 0 to YRegions - 1 do
  for X := 0 to XRegions - 1 do
  begin
    with ImageY.Regions[X, Y].Ifs do
    begin
      FileIfs.FormNum := FormNum;
      FileIfs.Betta := Betta;
      FileIfs.DomCoordX := Byte(DomCoordX);
      FileIfs.DomCoordY := Byte(DomCoordY);
      Stream.Write(FileIfs, SizeOf(FileIfs));
    end;
  end;

  if RGBEnabled then
  begin
    RegionSize := ImageU.RegionSize;
    XRegions := ImageU.Width div RegionSize;
    YRegions := ImageU.Height div RegionSize;

    Header.RegSize := RegionSize;
    Header.XRegions := XRegions;
```

```
    Header.YRegions := YRegions;
    Stream.Write(Header, SizeOf(TOptimizedIFSHeader));

    for Y := 0 to YRegions - 1 do
    for X := 0 to XRegions - 1 do
    begin
      with ImageU.Regions[X, Y].Ifs do
      begin
        FileIfs.FormNum := FormNum;
        FileIfs.Betta := Betta;
        FileIfs.DomCoordX := Byte(DomCoordX);
        FileIfs.DomCoordY := Byte(DomCoordY);
        Stream.Write(FileIfs, SizeOf(FileIfs));
      end;
    end;

    for Y := 0 to YRegions - 1 do
    for X := 0 to XRegions - 1 do
    begin
      with ImageV.Regions[X, Y].Ifs do
      begin
        FileIfs.FormNum := FormNum;
        FileIfs.Betta := Betta;
        FileIfs.DomCoordX := Byte(DomCoordX);
        FileIfs.DomCoordY := Byte(DomCoordY);
        Stream.Write(FileIfs, SizeOf(FileIfs));
      end;
    end;
  end;

  Stream.SaveToFile(FileName);
 finally
  Stream.Free;
 end;
end;

procedure TOptimizedIFS.DumpMatrix(Matrix: TMatrix; FileName: string);
var
 Bitmap: TBitmap;
 X, Y, V: Integer;
 P: PRGB32;
```

```
begin
  Bitmap := TBitmap.Create;
  Bitmap.PixelFormat := pf24bit;
  Bitmap.Width := Length(Matrix);
  Bitmap.Height := Length(Matrix[0]);
  for Y := 0 to Bitmap.Height - 1 do
  begin
    P := Bitmap.ScanLine[Y];
    for X := 0 to Bitmap.Width - 1 do
    begin
      V := Byte(Matrix[Y, X]);
      P.R := V;
      P.G := V;
      P.B := V;
      P := PRGB32(Integer(P) + 3);
    end;
  end;

  Bitmap.SaveToFile(FileName);
  Bitmap.Free;
end;

end.
```